# JavaScript规范

## 内容列表



## 类型

- **原始值**: 相当于传值(JavaScript对象都提供了字面量)，使用字面量创建对象。
    - string
    - number
    - boolean
    - null
    - undefined

```
var foo = 1,
    bar = foo;

bar = 9;

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

- **复杂类型**: 相当于传引用
    - object
    - array
    - function

```
var foo = [1, 2],
    bar = foo;

bar[0] = 9;

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

# 对象

- 使用字面值创建对象

```
// bad
var item = new Object();

// good
var item = {};
```

- 不要使用保留字 reserved words 作为键

```
// bad
var superman = {
  class: 'superhero',
  default: { clark: 'kent' },
  private: true
};

// good
var superman = {
  klass: 'superhero',
  defaults: { clark: 'kent' },
  hidden: true
};
```

# 数组

- 使用字面值创建数组

```
// bad
var items = new Array();

// good
var items = [];
```

- 如果你不知道数组的长度，使用push

```
var someStack = [];


// bad
someStack[someStack.length] = 'abracadabra';

// good
someStack.push('abracadabra');
```

- 当你需要拷贝数组时使用slice. jsPerf

```
var len = items.length,
    itemsCopy = [],
    i;
```

```
// bad
for (i = 0; i < len; i++) {
  itemsCopy[i] = items[i];
}

// good
itemsCopy = items.slice();
```

- 使用slice将类数组的对象转成数组.

```
function trigger() {
  var args = [].slice.apply(arguments);
  ...
}
```

# 字符串

- 对字符串使用单引号 '' (因为大多时候我们的字符串。特别html会出现 " )

```
// bad
var name = "Bob Parr";

// good
var name = 'Bob Parr';

// bad
var fullName = "Bob " + this.lastName;

// good
var fullName = 'Bob ' + this.lastName;
```

- 超过80(也有规定140的，项目具体可制定)个字符的字符串应该使用字符串连接换行
- 注: 如果过度使用，长字符串连接可能会对性能有影响. jsPerf & Discussion

```
// bad
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.';
```

```
// bad
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.';


// good
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.';
```

- 编程时使用join而不是字符串连接来构建字符串，特别是IE: jsPerf.

```
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;

// bad
function inbox(messages) {
  items = '<ul>';

  for (i = 0; i < length; i++) {
    items += '<li>' + messages[i].message + '</li>';
  }

  return items + '</ul>';
}
```

```
// good
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

- 函数表达式:

```
// 匿名函数表达式
var anonymous = function() {
  return true;
};

// 有名函数表达式
var named = function named() {
  return true;
};

// 立即调用函数表达式
(function() {
  console.log('Welcome to the Internet. Please follow me.');
})();
```

- 绝对不要在一个非函数块里声明一个函数，把那个函数赋给一个变量。浏览器允许你这么做，但是它们解析不同。

- **注:** ECMA-262定义把 块 定义为一组语句，函数声明不是一个语句。阅读ECMA-262对这个问题的说明.

```
// bad
if (currentUser) {
  function test() {
    console.log('Nope.');
  }
}
```

```
// good
if (currentUser) {
  var test = function test() {
    console.log('Yup.');
  };
}
```

- 绝对不要把参数命名为 `arguments`,这将会逾越函数作用域内传过来的 `arguments` 对象.

```
// bad
function nope(name, options, arguments) {
  // ...stuff...
}

// good
function yup(name, options, args) {
  // ...stuff...
}
```

# 属性

- 当使用变量和特殊非法变量名时，访问属性时可以使用中括号( . 优先).

```
var luke = {
  jedi: true,
  age: 28
};

function getProp(prop) {
  return luke[prop];
}

var isJedi = getProp('jedi');
```

# 变量

- 总是使用 var 来声明变量，如果不这么做将导致产生全局变量，我们要避免污染全局命名空间。

```
// bad
superPower = new SuperPower();

// good
var superPower = new SuperPower();
```

- 使用一个 var 以及新行声明多个变量，缩进4个空格。

```
// bad
var items = getItems();
var goSportsTeam = true;
var dragonball = 'z';

// good
var items = getItems(),
    goSportsTeam = true,
    dragonball = 'z';
```

- 最后再声明未赋值的变量，当你想引用之前已赋值变量的时候很有用。

```
// bad
var i, len, dragonball,
    items = getItems(),
    goSportsTeam = true;

// bad
var i, items = getItems(),
    dragonball,
    goSportsTeam = true,
    len;

// good
var items = getItems(),
    goSportsTeam = true,
    dragonball,
    length,
    i;
```

- 在作用域顶部声明变量，避免变量声明和赋值引起的相关问题。

```
// bad
```

```
function() {
  test();
  console.log('doing stuff..');

  //..other stuff..

  var name = getName();

  if (name === 'test') {
    return false;
  }

  return name;
}

// good
function() {
  var name = getName();

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

  //..other stuff..

  if (name === 'test') {
    return false;
  }

  return name;
}

// bad
function() {
  var name = getName();

  if (!arguments.length) {
    return false;
  }

  return true;
}

// good
function() {
  if (!arguments.length) {
    return false;
  }

  var name = getName();
```

```
  return true;
}
```

# 条件表达式和等号

- 合理使用 `===` 和 `!==` 以及 `==` 和 `!=`.
- 合理使用表达式逻辑操作运算.
- 条件表达式的强制类型转换遵循以下规则:
    - **对象** 被计算为 **true**
    - **Undefined** 被计算为 **false**
    - **Null** 被计算为 **false**
    - **布尔值** 被计算为 **布尔的值**
    - **数字** 如果是 **+0, -0, or NaN** 被计算为 **false** , 否则为 **true**
    - **字符串** 如果是空字符串 `''` 则被计算为 **false**, 否则为 **true**

```
if ([0]) {
  // true
  // An array is an object, objects evaluate to true
}
```

- 使用快捷方式.

```
// bad
if (name !== '') {
  // ...stuff...
}

// good
if (name) {
  // ...stuff...
}

// bad
if (collection.length > 0) {
  // ...stuff...
}

// good
if (collection.length) {
  // ...stuff...
```

```
}
```

- 阅读 Truth Equality and JavaScript 了解更多

# 块

- 给所有多行的块使用大括号

```
// bad
if (test)
  return false;

// good
if (test) return false;

// good
if (test) {
  return false;
}

// bad
function() { return false; }

// good
function() {
  return false;
}
```

# 注释

- 使用 `/** ... */` 进行多行注释，包括描述，指定类型以及参数值和返回值

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

```
function make(tag) {

  // ...stuff...

  return element;
}

// good
/**
 * 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，在评论对象的上面进行单行注释，注释前放一个空行.

```
// bad
var active = true;  // is current tab

// good
// is current tab
var active = true;

// bad
function getType() {
  console.log('fetching type...');
  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}

// good
function getType() {
  console.log('fetching type...');

  // set the default type to 'no type'
  var type = this._type || 'no type';
```

```
  return type;
}
```

- 如果你有一个问题需要重新来看一下或如果你建议一个需要被实现的解决方法的话需要在你的注释前面加上 `FIXME` 或 `TODO` 帮助其他人迅速理解

```
function Calculator() {

  // FIXME: shouldn't use a global here
  total = 0;

  return this;
}
```

```
function Calculator() {

  // TODO: total should be configurable by an options param
  this.total = 0;

  return this;
}
```

- 满足规范的文档，在需要文档的时候，可以尝试jsdoc.

# 空白

- 缩进、格式化能帮助团队更快得定位修复代码BUG.
- 将tab设为4个空格

```
// bad
function() {
··var name;
}

// bad
function() {
·var name;
}

// good
function() {
```

```
····var name;
}
```

- 大括号前放一个空格

```
// bad
function test(){
  console.log('test');
}

// good
function test() {
  console.log('test');
}

// bad
dog.set('attr',{
  age: '1 year',
  breed: 'Bernese Mountain Dog'
});

// good
dog.set('attr', {
  age: '1 year',
  breed: 'Bernese Mountain Dog'
});
```

- 在做长方法链时使用缩进.

```
// bad
$('#items').find('.selected').highlight().end().find('.open').updat
eCount();

// good
$('#items')
  .find('.selected')
    .highlight()
    .end()
  .find('.open')
    .updateCount();

// bad
var leds =
stage.selectAll('.led').data(data).enter().append('svg:svg').class(
'led', true)
    .attr('width',  (radius + margin) * 2).append('svg:g')
```

```
    .attr('transform', 'translate(' + (radius + margin) + ',' +
(radius + margin) + ')')
    .call(tron.led);

// good
var leds = stage.selectAll('.led')
    .data(data)
  .enter().append('svg:svg')
    .class('led', true)
    .attr('width',  (radius + margin) * 2)
  .append('svg:g')
    .attr('transform', 'translate(' + (radius + margin) + ',' +
(radius + margin) + ')')
    .call(tron.led);
```

# 逗号

- 不要将逗号放前面

```
// bad
var once
  , upon
  , aTime;

// good
var once,
    upon,
    aTime;

// bad
var hero = {
    firstName: 'Bob'
  , lastName: 'Parr'
  , heroName: 'Mr. Incredible'
  , superPower: 'strength'
};

// good
var hero = {
  firstName: 'Bob',
  lastName: 'Parr',
  heroName: 'Mr. Incredible',
  superPower: 'strength'
```

```
};
```

- 不要加多余的逗号，这可能会在IE下引起错误，同时如果多一个逗号某些ES3的实现会计算多数组的长度。

```
// bad
var hero = {
  firstName: 'Kevin',
  lastName: 'Flynn',
};

var heroes = [
  'Batman',
  'Superman',
];

// good
var hero = {
  firstName: 'Kevin',
  lastName: 'Flynn'
};

var heroes = [
  'Batman',
  'Superman'
];
```

# 分号

- 语句结束一定要加分号

```
// bad
(function() {
  var name = 'Skywalker'
  return name
})()

// good
(function() {
  var name = 'Skywalker';
  return name;
})();
```

```
// good
;(function() {
  var name = 'Skywalker';
  return name;
})();
```

# 类型转换

- 在语句的开始执行类型转换.
- 字符串:

```
//  => this.reviewScore = 9;

// bad
var totalScore = this.reviewScore + '';

// good
var totalScore = '' + this.reviewScore;

// bad
var totalScore = '' + this.reviewScore + ' total score';

// good
var totalScore = this.reviewScore + ' total score';
```

- 对数字使用 parseInt 并且总是带上类型转换的基数.，如 parseInt(value, 10)

```
var inputValue = '4';

// bad
var val = new Number(inputValue);

// bad
var val = +inputValue;

// bad
var val = inputValue >> 0;

// bad
var val = parseInt(inputValue);
```

```
// good
var val = Number(inputValue);

// good
var val = parseInt(inputValue, 10);

// good
/**
 * parseInt was the reason my code was slow.
 * Bitshifting the String to coerce it to a
 * Number made it a lot faster.
 */
var val = inputValue >> 0;
```

- 布尔值:

```
var age = 0;

// bad
var hasAge = new Boolean(age);

// good
var hasAge = Boolean(age);

// good
var hasAge = !!age;
```

# 命名约定

- 避免单个字符名，让你的变量名有描述意义。

```
// bad
function q() {
  // ...stuff...
}

// good
function query() {
  // ..stuff..
}
```

- 当命名对象、函数和实例时使用驼峰命名规则

```
// bad
var OBJEcttssssss = {};
var this_is_my_object = {};
var this-is-my-object = {};
function c() {};
var u = new user({
  name: 'Bob Parr'
});

// good
var thisIsMyObject = {};
function thisIsMyFunction() {};
var user = new User({
  name: 'Bob Parr'
});
```

- 当命名构造函数或类时使用驼峰式大写

```
// bad
function user(options) {
  this.name = options.name;
}

var bad = new user({
  name: 'nope'
});

// good
function User(options) {
  this.name = options.name;
}

var good = new User({
  name: 'yup'
});
```

- 命名私有属性时前面加个下划线 _

```
// bad
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';

// good
```

```
this._firstName = 'Panda';
```

- 当保存对 this 的引用时使用 self(python 风格),避免 this issue .Angular建议使用vm(MVVM模式中view-model)

```
// good
function() {
  var self = this;
  return function() {
    console.log(self);
  };
}
```

# 存取器

- 属性的存取器函数不是必需的
- 如果你确实有存取器函数的话使用getVal() 和 setVal('hello'), java getter、setter风格 或者 jQuery风格

- 如果属性是布尔值，使用isVal() 或 hasVal()

```
// bad
if (!dragon.age()) {
  return false;
}

// good
if (!dragon.hasAge()) {
  return false;
}
```

- 可以创建get()和set()函数，但是要保持一致

```
function Jedi(options) {
  options || (options = {});
  var lightsaber = options.lightsaber || 'blue';
  this.set('lightsaber', lightsaber);
}

Jedi.prototype.set = function(key, val) {
  this[key] = val;
```

```
};

Jedi.prototype.get = function(key) {
  return this[key];
};
```

# 构造器

- 给对象原型分配方法，而不是用一个新的对象覆盖原型，覆盖原型会使继承出现问题。

```
function Jedi() {
  console.log('new jedi');
}

// bad
Jedi.prototype = {
  fight: function fight() {
    console.log('fighting');
  },

  block: function block() {
    console.log('blocking');
  }
};

// good
Jedi.prototype.fight = function fight() {
  console.log('fighting');
};

Jedi.prototype.block = function block() {
  console.log('blocking');
};
```

- 方法可以返回 `this` 帮助方法可链。

```
// bad
Jedi.prototype.jump = function() {
  this.jumping = true;
  return true;
};
```

```
Jedi.prototype.setHeight = function(height) {
  this.height = height;
};

var luke = new Jedi();
luke.jump(); // => true
luke.setHeight(20) // => undefined

// good
Jedi.prototype.jump = function() {
  this.jumping = true;
  return this;
};

Jedi.prototype.setHeight = function(height) {
  this.height = height;
  return this;
};

var luke = new Jedi();

luke.jump()
  .setHeight(20);
```

- 可以写一个自定义的toString()方法，但是确保它工作正常并且不会有副作用。

```
function Jedi(options) {
  options || (options = {});
  this.name = options.name || 'no name';
}

Jedi.prototype.getName = function getName() {
  return this.name;
};

Jedi.prototype.toString = function toString() {
  return 'Jedi - ' + this.getName();
};
```

# 事件

- 当给事件附加数据时，传入一个哈希而不是原始值，这可以让后面的贡献者加入更多数据到事件数据里而不用找出并更新那个事件的事件处理器

```
// bad
$(this).trigger('listingUpdated', listing.id);

...

$(this).on('listingUpdated', function(e, listingId) {
  // do something with listingId
});
```

更好:

```
// good
$(this).trigger('listingUpdated', { listingId : listing.id });

...

$(this).on('listingUpdated', function(e, data) {
  // do something with data.listingId
});
```

# 模块

- 这个文件应该以驼峰命名，并在同名文件夹下，同时导出的时候名字一致
- 对于公开API库可以考虑加入一个名为noConflict()的方法来设置导出的模块为之前的版本并返回它
- 总是在模块顶部声明 `'use strict';`，引入 [jslint 规范](https://github.com/douglascrockford/JSLint)

```
// fancyInput/fancyInput.js

(function(global) {
  'use strict';

  var previousFancyInput = global.FancyInput;

  function FancyInput(options) {
    this.options = options || {};
  }

  FancyInput.noConflict = function noConflict() {
    global.FancyInput = previousFancyInput;
```

```
    return FancyInput;
  };

  global.FancyInput = FancyInput;
})(this);
```

# jQuery

- 缓存jQuery查询

```
// bad
function setSidebar() {
  $('.sidebar').hide();

  // ...stuff...

  $('.sidebar').css({
    'background-color': 'pink'
  });
}

// good
function setSidebar() {
  var $sidebar = $('.sidebar');
  $sidebar.hide();

  // ...stuff...

  $sidebar.css({
    'background-color': 'pink'
  });
}
```

- 对DOM查询使用级联的 `$('.sidebar ul')` 或 `$('.sidebar ul')` , jsPerf
- 对有作用域的jQuery对象查询使用 `find`

```
// bad
$('.sidebar', 'ul').hide();

// bad
$('.sidebar').find('ul').hide();
```

```
// good
$('.sidebar ul').hide();

// good
$('.sidebar > ul').hide();

// good (slower)
$sidebar.find('ul');

// good (faster)
$($sidebar[0]).find('ul');
```

# ECMAScript 5兼容性

尽量采用ES5方法，特别数组map、filter、forEach方法简化日常开发。在老式IE浏览器中引入ES5-shim。或者也可以考虑引入underscore、lodash 常用辅助库.
- 参考Kangax的 ES5 compatibility table
- JavaScript工具库之Lodash
- Babel-现在开始使用 ES6