dead-horse / node-style-guide

forked from felixge/node-style-guide

A guide for styling your node.js / JavaScript code. Fork & adjust to your taste.

37 commits 1 branch 0 releases 7 contributors

Branch: master node-style-guide / +

This branch is 15 commits ahead, 51 commits behind felixge:master. Pull request Compare

dead-horse Merge pull request #1 from fengmk2/typo-fix ··· Latest commit 548880b on Jan 10, 2014

Readme.en.md	translate to chinese	2 years ago
Readme.md	尽量每行不超过80字符, 修复所有英文标点符号	2 years ago

Code
Pull requests 0
Wiki
Pulse
Graphs

Readme.md

Node.js Style Guide

这是一份关于如何写出一致且美观的 Node.js 代码的风格指南。 它从社区最流行的写法中抽取出来，同时融入了部分个人观点。

这份指南由 Felix 编写。 通过 CC BY-SA 3.0 发布。 你可以 fork 这份指南，并针对自己的需求的进行部分的调整。

中文指南由 dead_horse 翻译，同时根据个人和团队经验有部分增加条目。

2空格缩进

使用2个空格而不是 tab 来进行代码缩进，同时绝对不要混用空格和 tab 。

Sublime Text 2 设置(perfernces > Settings - User)：

```
"tab_size": 2,
"translate_tabs_to_spaces": true
```

换行

使用 UNIX 风格的换行符 (`\n`)，同时在每个文件的结尾添加一个换行符。 Windows 风格的换行符 (`\r\n`) 是绝对禁止出现在任何项目中的。

Sublime Text 2 设置(perfernces > Settings - User)：

```
"default_line_ending": "unix"
```

去除行末尾的多余空格

就像吃完饭要刷牙一样，在提交 (commit) 代码之前你需要清理掉所有的不必要的空格。

Sublime Text2 设置(perfernces > Settings - User)：

```
"trim_trailing_white_space_on_save": true
```

使用分号

是否使用分号，在社区争论已久。 isaac 也写过一篇讨论的文章， 但是，当可以用廉价的语法来消除一些可能引入的错误的时候，请当一个保守派。

每行80个字符

限制你每行代码不超过80个字符。尽管现在的显示器越来越大，但是你的大脑并没有变大，并且你还可以把你的大显示器切分成多屏来显示。

Sublime Text 2 设置(perfernces > Settings - User)：

```
"rulers": [80]
```

多屏： `view > Layout > Columns 2`

使用单引号

除非编写.json文件，其他时候都请用单引号包裹字符串。

Right:

```
var foo = 'bar';
```

Wrong:

```
var foo = "bar";
```

大括号位置

请把你的所有的左大括号都放在语句开始的这一行。

Right:

```
if (true) {
  console.log('winning');
}
```

Wrong:

```
if (true)
{
  console.log('losing');
}
```

同时，请注意在条件判断前后都添加一个空格。

每个变量声明都带一个 var

每个变量声明都带一个 var ， 这样删除或者调整变量声明的顺序会更加容易。 不要把变量都声明在最前面，而是声明在它最有意义的地方。

Right:

```
var keys   = ['foo', 'bar'];
var values = [23, 42];

var object = {};
while (items.length) {
  var key = keys.pop();
  object[key] = values.pop();
}
```

Wrong:

```
var keys = ['foo', 'bar'],
    values = [23, 42],
    object = {},
    key;

while (items.length) {
  key = keys.pop();
  object[key] = values.pop();
}
```

变量、属性和函数名都采用小驼峰

变量、属性和函数的命名风格都需要遵循小驼峰风格。 同时所有的命名都是有意义的。 尽量避免用单字符变量和少见单词来命名。

Right:

```
var adminUser = db.query('SELECT * FROM users ...');
```

Wrong:

```
var admin_user = db.query('SELECT * FROM users ...');
var a = db.query('SELECT * FROM users ...');
```

类名采用大驼峰

类名都采用大驼峰风格来命名。

Right:

```
function BankAccount() {
}
```

Wrong:

```
function bank_Account() {
}
```

用大写来标识常量

常量变量和对象的静态常量属性都需要特殊表明，通过全部大写的方式来表明。

尽管 Node.js / V8 支持 mozilla 的 const 关键字， 但是不幸的是，对象的属性并不支持这个关键字，而且 const 没有包含于任何一个 ECMA 规范中。

Right:

```
var SECOND = 1 * 1000;

function File() {
}
File.FULL_PERMISSIONS = 0777;
```

Wrong:

```
const SECOND = 1 * 1000;

function File() {
}
File.fullPermissions = 0777;
```

对象、数组的创建

使用尾随逗号，尽量用一行来声明，只有在编译器不接受的情况下才把对象的 key 用单引号包裹。 使用字面表达式，用 `{}, []` 代替 `new Array, new Object` 。

Right:

```
var a = ['hello', 'world'];
var b = {
  good: 'code',
  'is generally': 'pretty',
};
```

Wrong:

```
var a = [
  'hello', 'world'
];
var b = {"good": 'code'
        , is generally: 'pretty'
        };
```

使用 === 比较符

写代码并不是在背这些 stupid rules 。使用 `===` 操作符来进行比较操作，它会完全按照你的期望来执行。

Right:

```
var a = 0;
if (a === '') {
  console.log('winning');
}
```

Wrong:

```
var a = 0;
if (a == '') {
  console.log('losing');
}
```

三元操作符分多行

三元操作符不应该写在一行，将它分割到多行。

Right:

```
var foo = (a === b)
  ? 1
  : 2;
```

Wrong:

```
var foo = (a === b) ? 1 : 2;
```

不要扩展内建类型

不要扩展 javascript 内建对象的方法。将来的你会感谢你这个做法的。

Right:

```
var a = [];
if (!a.length) {
  console.log('winning');
}
```

Wrong:

```
Array.prototype.empty = function() {
  return !this.length;
}

var a = [];
if (a.empty()) {
  console.log('losing');
}
```

使用有意义的判断条件

所有复杂的条件判断都需要赋予一个有意义的名字或者方法。

Right:

```
var isValidPassword = password.length >= 4 && /^(?=.*\d).{4,}$/.test(password);

if (isValidPassword) {
  console.log('winning');
}
```

Wrong:

```
if (password.length >= 4 && /^(?=.*\d).{4,}$/.test(password)) {
  console.log('losing');
}
```

写精简的函数

保持你的函数尽可能的精简。 一个好的函数应该能够在幻灯片上一屏显示，并且让坐在教室最后一排的人看清楚。 别再去数你的每一个函数并控制在15行以内了。

尽早的从函数中返回

为了避免深入嵌套的 if 语句，请尽早的从函数中返回。

Right:

```
function isPercentage(val) {
  if (val < 0) {
    return false;
  }

  if (val > 100) {
    return false;
  }

  return true;
}
```

Wrong:

```
function isPercentage(val) {
  if (val >= 0) {
    if (val < 100) {
      return true;
    } else {
      return false;
    }
  } else {
    return false;
  }
}
```

针对这个示例，甚至可以进一步精简优化：

```
function isPercentage(val) {
  var isInRange = (val >= 0 && val <= 100);
  return isInRange;
}
```

给你的闭包命名

请尽量给你的闭包、匿名函数命名。 这让人知道你在意这个函数，更重要的是，这将会产生可读性更好的堆栈跟踪和CPU调用信息等。

Right:

```
req.on('end', function onEnd() {
  console.log('winning');
});
```

Wrong:

```
req.on('end', function() {
  console.log('losing');
});
```

不要嵌套闭包

使用闭包，但是不要嵌套他们，否则你的代码将会一团糟。

Right:

```
setTimeout(function() {
  client.connect(afterConnect);
}, 1000);

function afterConnect() {
  console.log('winning');
}
```

Wrong:

```
setTimeout(function() {
  client.connect(function() {
    console.log('losing');
  });
}, 1000);
```

使用单行注释风格

不管是单行注释还是多行注释，都使用 `//` 。 同时请尝试在更高层次来编写注释（解释函数整体的思路）， 只在解释一些难以理解代码的时候添加注释，而不是给一些琐碎的东西加上注释。

Right:

```
// 'ID_SOMETHING=VALUE' -> ['ID_SOMETHING=VALUE'', 'SOMETHING', 'VALUE']
var matches = item.match(/ID_([^\n]+)=([^\n]+)/));

// This function has a nasty side effect where a failure to increment a
// redis counter used for statistics will cause an exception. This needs
// to be fixed in a later iteration.
function loadUser(id, cb) {
  // ...
}

var isSessionValid = (session.expires < Date.now());
if (isSessionValid) {
  // ...
}
```

Wrong:

```
// Execute a regex
var matches = item.match(/ID_([^\n]+)=([^\n]+)/));

// Usage: loadUser(5, function() { ... })
function loadUser(id, cb) {
  // ...
```

```
}

// Check if the session is valid
var isSessionValid = (session.expires < Date.now());
// If the session is valid
if (isSessionValid) {
  // ...
}
```

Object.freeze, Object.preventExtensions, Object.seal, with, eval

这一堆屎一样的东西，你永远都不会需要他们。

Getters 和 Setters

不要使用 setters ，他们会引发一些使用你的代码的人无法解决的问题。 当没有副作用的时候，可以使用 getters，例如提供一个集合类的长度属性的时候。

异步回调函数

Node 的异步回调函数的第一个参数应该是错误指示，只有这样才能够享受许多流程控制模块的福利。

Right:

```
function cb(err, data , ...) {...}
```

Wrong:

```
function cb(data, ...) {...}
```

继承

尽管有许多的方法来实现继承，但是最为推荐的是 Node 的标准写法：

```
function Socket(options) {
  // ...
  stream.Stream.call(this);
  // ...
}

util.inherits(Socket, stream.Stream);
```

文件命名

单词之间使用 `_` underscore 来分割，如果你不想暴露某个文件给用户，你也可以用 `_` 来开头

Right:

```
child_process.js
string_decoder.js
_linklist.js
```

Wrong:

```
childProcess.js
stringDecoder.js
```

空格

在所有的操作符前后都添加空格，`function` 关键字后面添加空格

Right:

```
var add = function (a, b) {
  return a + b;
};
```

Wrong:

```
var add=function(a,b){
  return a+b;
}
```

尽量参照 Node.js 源码的编码风格

- node 源码
- Google’s JavaScript style guide

入乡随俗

给别人的项目提交 pull request 的时候，要注意遵循项目的编码规范，保持项目编码风格的统一。