分类：IT技术

上一篇：2009年我的网志总结

下一篇：16世纪的英国，21世纪的中国

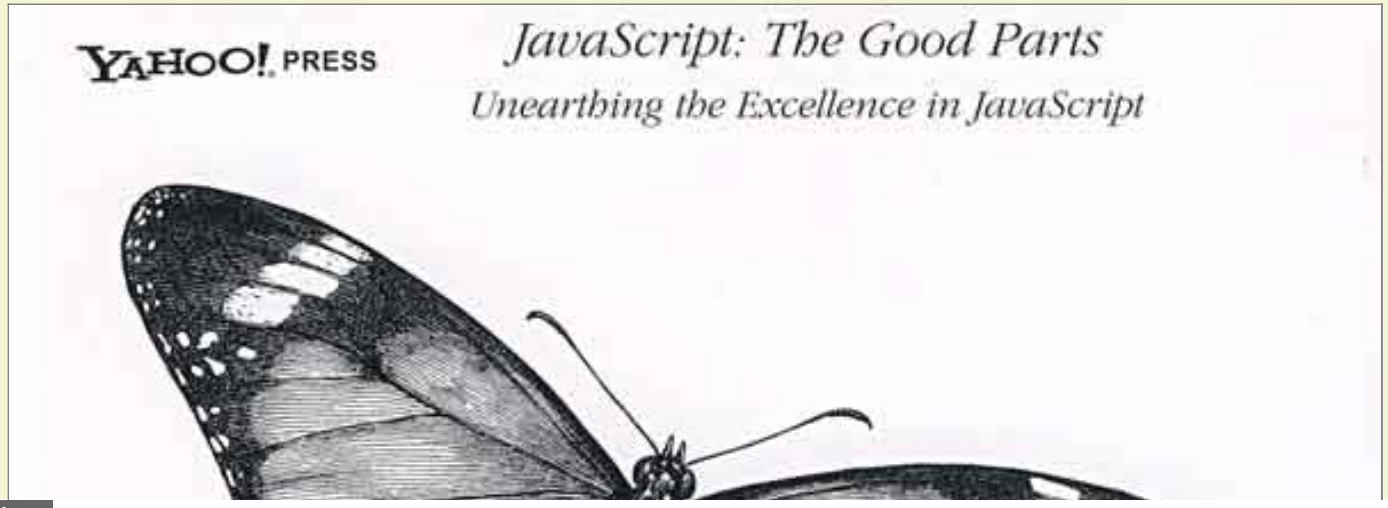
12种不宜使用的Javascript语法

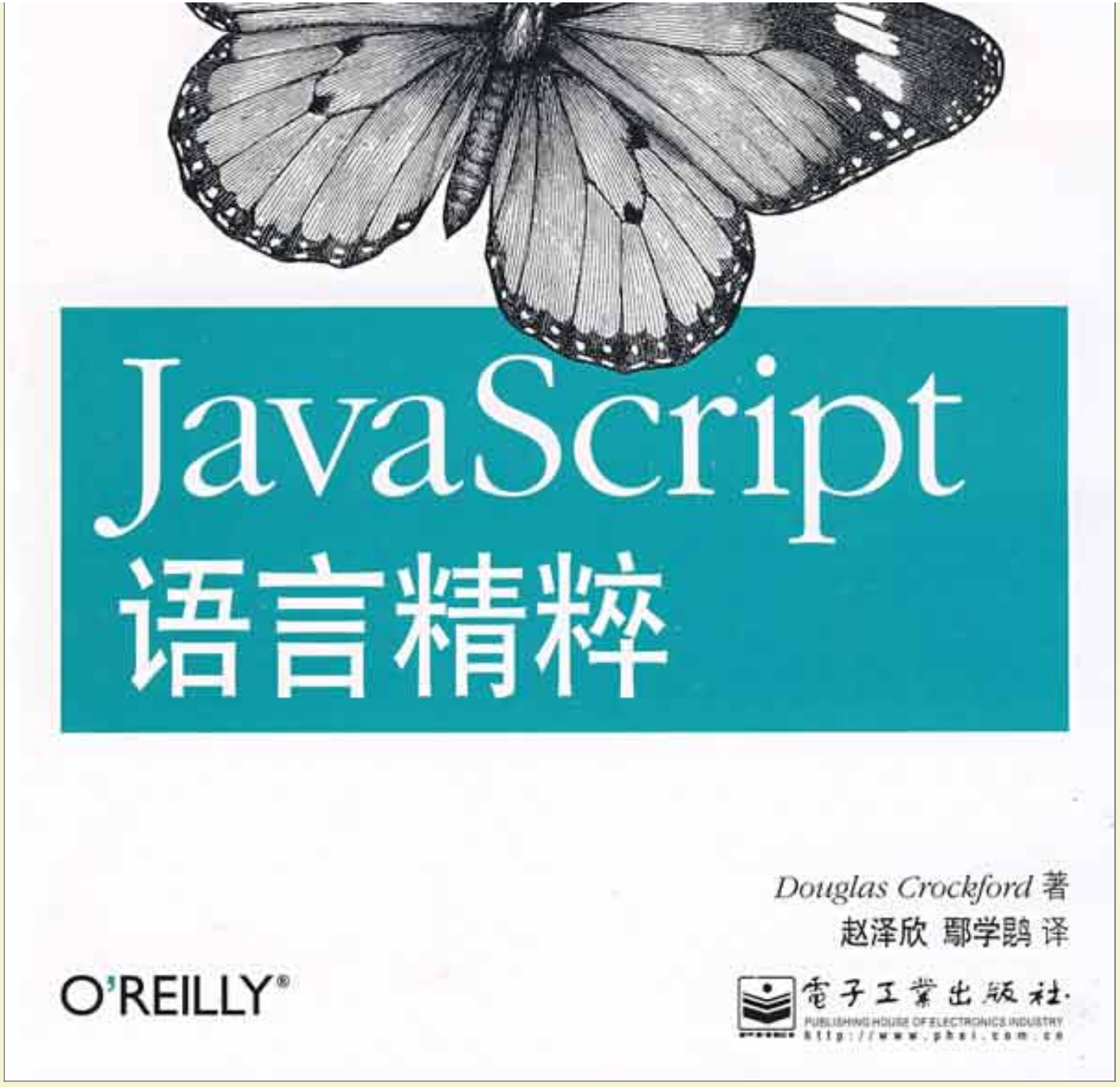
作者： 阮一峰

日期： 2010年1月 3日

这几天，我在读《Javascript语言精粹》。

这本书很薄，100多页，正好假日里翻翻。





该书的作者是Douglas Crockford，他是目前世界上最精通Javascript的人之一，也是Json格式的创造者。

他认为Javascript有很多糟粕。因为1995年Brendan Eich设计这种语言的时候，只用了三个月，很多语言特性没有经过深思熟虑，就推向了市场。结果等到人们意识到这些问题的时候，已经有100万程序员在使用它了，不可能再大幅修改语言本身了。所以，Douglas Crockford决定，他要告诉大家，Javascript中哪些部分是精粹，哪些部分是糟粕和鸡肋。

这个想法非常好，但是我不得不说，这本书写得不够好，不适合新手阅读。原因如下：1）Douglas Crockford叙述得不清晰，更像与同行讨论问题，而不是由浅入深地讲解问题。这本书的重点不是解释，所以读完后，我觉得Javascript好像变得更复杂了。2）他固执地使用铁路图（railroad diagram）解释每一条语句。全世界似乎只有他一个人使用这种比Javascript更难看懂的图。3）该书基本上是一本简化的Javascript语法手册，缺乏足够的新内容。4）该书举例过少，而且在最难的函数和对象部分，使用的例子都是环环相套、层层递进的例子，导致阅读起来很吃力。

该书最有价值的内容不是正文，反而是附录。在附录B中，Douglas Crockford列出了12种应该避免使用的Javascript语法，我觉得非常值得推广。

==========

1. ==

Javascript有两组相等运算符，一组是==和!=，另一组是===和!==。前者只比较值的相等，后者除了值以外，还比较类型是否相同。

请尽量不要使用前一组，永远只使用===和!==。因为==默认会进行类型转换，规则十分难记。如果你不相信的话，请回答下面五个判断式的值是true还是false：

```
false == 'false'

false == undefined

false == null

null == undefined

0 == ''
```

前三个是false，后两个是true。

2. with

with的本意是减少键盘输入。比如

```
obj.a = obj.b;

obj.c = obj.d;
```

可以简写成

```
with(obj) {
    a = b;
    c = d;
}
```

但是，在实际运行时，解释器会首先判断obj.b和obj.d是否存在，如果不存在的话，再

判断全局变量b和d是否存在。这样就导致了低效率，而且可能会导致意外，因此最好不要使用with语句。

3. eval

eval用来直接执行一个字符串。这条语句也是不应该使用的，因为它有性能和安全性的问题，并且使得代码更难阅读。

eval能够做到的事情，不用它也能做到。比如

```
eval("myValue = myObject." + myKey + ";");
```

可以直接写成

```
myValue = myObject[myKey];
```

至于ajax操作返回的json字符串，可以使用官方网站提供的解析器json_parse.js运行。

4. continue

这条命令的作用是返回到循环的头部，但是循环本来就会返回到头部。所以通过适当的构造，完全可以避免使用这条命令，使得效率得到改善。

5. switch 贯穿

switch结构中的case语句，默认是顺序执行，除非遇到break，return和throw。有的程序员喜欢利用这个特点，比如

```
switch(n) {
```

```
        case 1:
        case 2:
            break;
    }
```

这样写容易出错，而且难以发现。因此建议避免switch贯穿，凡是有case的地方，一律加上break。

```
    switch(n) {
        case 1:
            break;
        case 2:
            break;
    }
```

6. 单行的块结构

if、while、do和for，都是块结构语句，但是也可以接受单行命令。比如

```
    if (ok) t = true;
```

甚至写成

```
    if (ok)
        t = true;
```

这样不利于阅读代码，而且将来添加语句时非常容易出错。建议不管是否只有一行命

令，都一律加上大括号。

```
if (ok){
    t = true;
}
```

7. ++和--

递增运算符++和递减运算符--，直接来自C语言，表面上可以让代码变得很紧凑，但是实际上会让代码看上去更复杂和更晦涩。因此为了代码的整洁性和易读性，不用为好。

8. 位运算符

Javascript完全套用了Java的位运算符，包括按位与&、按位或|、按位异或^、按位非~、左移<<、带符号的右移>>和用0补足的右移>>>。

这套运算符针对的是整数，所以对Javascript完全无用，因为Javascript内部，所有数字都保存为双精度浮点数。如果使用它们的话，Javascript不得不将运算数先转为整数，然后再进行运算，这样就降低了速度。而且“按位与运算符”&同“逻辑与运算符”&&，很容易混淆。

9. function语句

在Javascript中定义一个函数，有两种写法：

```
function foo() { }
```

和

```
var foo = function () { }
```

两种写法完全等价。但是在解析的时候，前一种写法会被解析器自动提升到代码的头部，因此违背了函数应该先定义后使用的要求，所以建议定义函数时，全部采用后一种写法。

10. 基本数据类型的包装对象

Javascript的基本数据类型包括字符串、数字、布尔值，它们都有对应的包装对象String、Number和Boolean。所以，有人会这样定义相关值：

```
new String("Hello World");

new Number(2000);

new Boolean(false);
```

这样写完全没有必要，而且非常费解，因此建议不要使用。

另外，new Object和new Array也不建议使用，可以用{}和[]代替。

11. new语句

Javascript是世界上第一个被大量使用的支持Lambda函数的语言，本质上属于与Lisp同类的函数式编程语言。但是当前世界，90%以上的程序员都是使用面向对象编程。为了靠近主流，Javascript做出了妥协，采纳了类的概念，允许根据类生成对象。

类是这样定义的：

```
var Cat = function (name) {
    this.name = name;
    this.saying = 'meow' ;
}
```

然后，再生成一个对象

```
var myCat = new Cat('mimi');
```

这种利用函数生成类、利用new生成对象的语法，其实非常奇怪，一点都不符合直觉。而且，使用的时候，很容易忘记加上new，就会变成执行函数，然后莫名其妙多出几个全局变量。所以，建议不要这样创建对象，而采用一种变通方法。

Douglas Crockford给出了一个函数：

```
Object.beget = function (o) {
    var F = function (o) {};
    F.prototype = o ;
    return new F;
};
```

创建对象时就利用这个函数，对原型对象进行操作：

```
var Cat = {
    name:'',
    saying:'meow'
```

```
};

var myCat = Object.beget(Cat);
```

对象生成后，可以自行对相关属性进行赋值：

```
myCat.name = 'mimi';
```

12. void

在大多数语言中，void都是一种类型，表示没有值。但是在Javascript中，void是一个运算符，接受一个运算数，并返回undefined。

```
void 0; // undefined
```

这个命令没什么用，而且很令人困惑，建议避免使用。

（完）

文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名 | Creative Commons BY-NC-ND 3.0
- 原文网址：http://www.ruanyifeng.com/blog/2010/01/12_javascript_syntax_structures_you_should_not_use.html
- 最后修改时间：2010年1月 3日 15:39

相关文章

- **2010.12.27:** PHP最佳实践

  > 今天下午，我在读下面这篇文章。

- **2010.12.07:** 《黑客英雄》书摘

  > 《黑客与画家》翻译完成后，我还欠出版社一篇译者序。

功能链接

- 前一篇：2009年我的网志总结
- 后一篇：16世纪的英国，21世纪的中国
- 更多内容请访问：首页 » 档案 » IT技术
- 站内搜索：javascript GO！
- **Feed**订阅：

广告

留言（19条）

1/137 说：

第11条不错，拿去用了...

2010年1月 3日 17:22 | 档案 | 引用

keshin 说：

第一部分介绍语法时铁路图比较多，除了第一遍的时候看过一遍外，之后翻就直接跳过了.....
这本书我觉得挺好，这几天要准备一个前端的面试，又拿出来翻了一遍。

至于这十二条，确实是眞知灼见，只是第一条我持保留意见，个人感觉如果完全不用==或者!=，就完全丧失了js弱类型的优势。

2010年1月 3日 17:30 | 档案 | 引用

半就业 说：

第7条有些不明白，阅读起来挺容易的。
在C语言里i++要比i=i+1运算速度快，Java就不知道了

2010年1月 3日 17:47 | 档案 | 引用

路过 说：

第一条我也不赞成，JavaScript本身就是弱类型语言，对于熟悉JavaScript的人来说反倒是尽量少用“全等于”...

2010年1月 3日 20:10 | 档案 | 引用

Kaelzhang 说：

1. 第一条，有些时候，全部用===或者!==会出现致命的错误，比如经典的对于domready的判断，碰到IE，当你闭着眼睛写===，你就掉坑了。(IE下self===window为false)，很多时候，还是需要清醒的判断，而不是“不宜”写什么。

7.个人愿意用-- ++，这个运算符在很多浏览器中的效率都很高。

11.恰巧，mootools的应用风格就是使用new，有时候我觉得把js看成类定义，更符合通常程序员的编程思维。

2，3，4，6，8，9，10同意
eval is evil，极低的运算效率，是访问一个简单函数运算执行时间几百倍。eval内容越长，耗费越多时间。

8. 很同意，之前还准备位运算符，结果发现js的位运算符不支持浮点数，把我害得很惨。

2010年1月 3日 21:21 | 档案 | 引用

dylanklc 说：

10.&11. 在V8里面 有极大的优化了
鉴于javascript依赖于不同的解释器来实现,具体还是要看浏览器所采用的解释器了.他这12条也是为了避免javascript在不同浏览器运行时跑出各种奇异结果所写.
ps:Javascript要发展必须要靠V8等更强力的虚拟机来解释了.

2010年1月 3日 21:30 | 档案 | 引用

keshin 说：

> 引用**Kaelzhang**的发言：
>
> 11.恰巧，mootools的应用风格就是使用new，有时候我觉得把js看成类定义，更符合通常程序员的编程思维。

更符合通常程序员的编程思维这点并不赞同，JavaScript的继承是基于原型，虽然有类的影子或者说可以模拟类的方式，但那样反而会很累。jQuery之所以流行，个人认为就在于他让开发者抛开了类的概念。

2010年1月 3日 22:11 | 档案 | 引用

keshin 说：

呵呵，貌似我对这个问题产生兴趣了，重翻了一下ECMA 262的规范，里面对于==的解释却是够繁琐的……，大致整理如下（参见11.9.1）：

1、类型相同的情况（略过）

2、null和undefined比较，返回true

3、数字和字符串比较，先将字符串转为数字（toNumber），然后比较

4、布尔值和其他类型比较，先将布尔值转为数字（toNumber），然后比较

5、字符串或者数字和对象比较，先将对象转为基本类型（toPrimitive）

至于如何toNumber和toPrimitive，则又是另外一个地方的论述……

可能ECMA的人也觉得实在太过复杂，规则下面另外做了提示，在比较前如何强制转型，比如：

1、强制转字符串：a + "" 这个大家都知道

2、强制转数字：+a 呵呵，我以前习惯用a - 0

3、强制转布尔： !a

回过头来说，恐怕我要修正一下我的观点，平常我实际上很少用==，逻辑判断是基本上直接

```
if (someexpress) {
// do
```

}

而不是

if (someexpress == true) {

// do

}

我的做法可以利用js自动转布尔值的机制，规则相对简单，而后者确实会有比较多的问题，如果要用到，可能用===相等性测试确实更为稳妥。

惭愧惭愧，之前一直忽略这个问题了。

2010年1月 4日 10:21 | 档案 | 引用

gg 说：

In spare time, I like reading book, surfing internet, watching movie and taking a leisurely walk outdoors.

似乎应该是reading books, surfing the internet, watching movies

2010年1月 4日 19:01 | 档案 | 引用

roxhaiy 说：

阮先生，有篇文章推荐您看看：

http://www.dapenti.com/blog/more.asp?name=xilei&id=25558

2010年1月 4日 21:18 | 档案 | 引用

passion 说：

很像as 看过来也有不少收益

特别with和原形链

2010年1月 5日 17:05 | 档案 | 引用

AlexYuan 说：

越来越喜欢这里。

比之王小和和菜的blog，好多了。

- 1. 只要控制好逻辑、类型，我觉得==、!=已经足够了，个人浅陋的想法。

相等运算符 （==、!=）

- 如果两表达式的类型不同，则试图将它们转换为字符串、数字或

Boolean 量。

- NaN 与包括其本身在内的任何值都不相等。
- 负零等于正零。
- null 与 null 和 undefined 相等。

- 相同的字符串、数值上相等的数字、相同的对象、相同的 Boolean 值或者（当类型不同时）能被强制转化为上述情况之一，均被认为是相等的。

- 其他比较均被认为是不相等的。

- 2. 不完全同意，有时候用到with，可读性会好些，减少键盘输入只是一方面
- 3. 同意，从来不用eval
- 4. 基本同意，几乎不用continue
- 5. 同意，并自勉：凡是有case的地方，一律加上break。
- 6. 可读性，怎么说呢？google的一些api和诸如jquery-1.3.2.min.js、ext-all.js整个代码全都写成了单

行，release版本吧。最起码debug的时候还是格式化看着舒服。

- 7. i++ 或i-- 复杂和晦涩吗？ 那不要用js好了
- 8. 很少用到
- 9.长知识了。看不懂这句话“前一种写法会被解析器自动提升到代码的头部，因此违背了函数应该先定义后使用的要求”
- 10.的确，完全没有必要
- 11. 哦
- 12.viod几乎不用。看到这么一句：void 运算符对

表达式求值，并返回 undefined。在希望求表达式的值，但又不希望脚本的剩余部分看见这个结果时，该运算符最有用。

2010年1月 6日 10:42 | 档案 | 引用

熊眼看世界 说：

看到11条，才有点感觉

2010年1月 6日 11:35 | 档案 | 引用

oldsha 说：

> 引用**AlexYuan**的发言：
>
> 9.长知识了。看不懂这句话“前一种写法会被解析器自动提升到代码的头部，因此违背了函数应该先定义后使用的要求”

function foo() { } 和 var foo = function () { } 这两种写法的区别是，第一种写法在变量和表达式初始化的时候foo就被定义了，所以可以在任何地方调用foo函数，而第二种写法在JavaScript引擎执行到这一行代码之前foo是未定义的，所以调用函数的

话，只能在函数代码的后面调用，就是所说的“先定义后使用”，不过我感觉第一种方法更不容易犯错。

2010年1月 7日 14:43 | 档案 | 引用

George Wing 说：

第11种中：

var F = function (o) {};

这行是空的构造函数，这个参数o作什么用的呢？

2010年1月 7日 16:34 | 档案 | 引用

netwjx 说：

感觉prototype是一种很容易让代码陷入不可控制的特性

应该要有限制的使用

2010年1月10日 00:08 | 档案 | 引用

netwjx 说：

> 引用**George Wing**的发言：
>
> 第11种中：
>
> var F = function (o) {};
>
> 这行是空的构造函数，这个参数o作什么用的呢？

目前没作用 作用就是F函数如果提供了一个参数,那么这个参数在F函数内部可以使用o访问

2010年1月10日 00:14 | 档案 | 引用

longbill 说：

更加坚定我不买这本书的决心。。。。说的乱七八糟的，没有明确目的。 有的是为了效率，有的是为了便于理解。 在不同的环境下程序员对效率和便于理解有不同的需求。

对js了解比较透彻的看上面这些到没什么，如果初学者看到了，会让人产生迷惑，让人误入歧途。。。。。

2010年1月11日 10:35 | 档案 | 引用

shuil 说：

给推荐本适合新手学习js的书吧

2010年1月27日 11:56 | 档案 | 引用

我要发表看法

您的留言 （HTML标签部分可用）

您的大名：

«-必填

电子邮件：

«-必填，不公开

个人网址：

«-我信任你，不会填写广告链接

记住个人信息？☐

发表 «- 点击按钮