
2015 年 12 条
专业的 JS 规则

前言

2015年 12条专业的JavaScript规则

> 原文出处：Coder资源网

免责声明：下面的内容为了简洁说的有些绝对，是的，在编程中所有的“规则”都有例外。

学习JavaScript是困难的。它发展的如此之快，以至于在任何一个特定的时刻，你都不清楚自己是否“做错了”。有些时候，感觉像是坏的部分超过了好的部分。然而，讨论这些并没有意义，JavaScript正在征服世界，所以，我们也只能这么做了。

下面是我的一些建议：

1. JS应该放到 .js 文件中

1. JS应该放到 .js 文件中

“额，只有那么几行而已…”，是的，我的意思是所有的 JS 都应该放在 `.js` 文件中。为什么呢？因为这有助于可读性，节省带宽。行内 JavaScript 在每次页面加载时都会重新下载，相反的，单独的 `.js` 文件则会被缓存起来。正如你所看到的，这个规则有助于支持如下一长串的其他规则。这就是为什么它的规则#1。

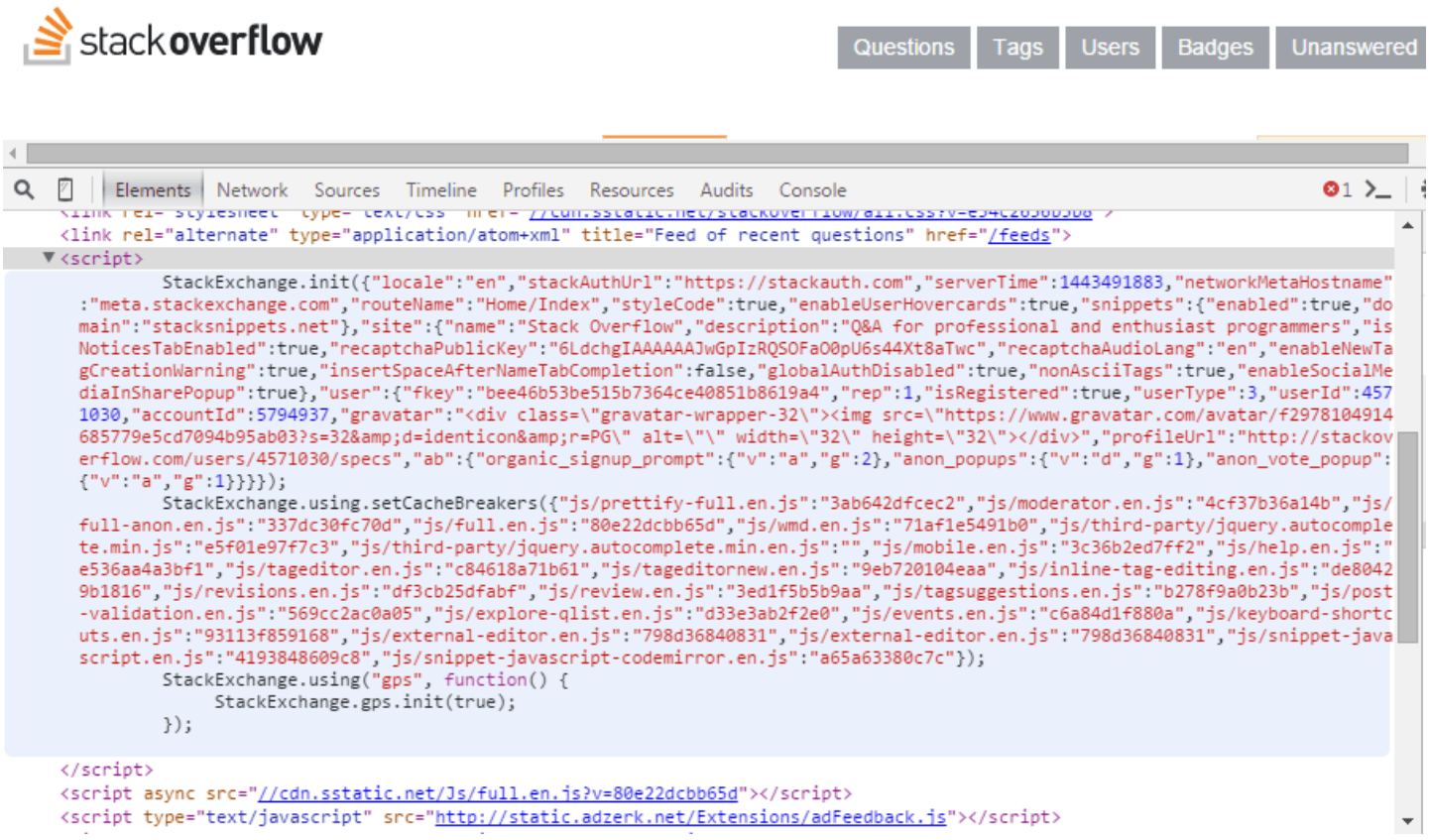
2. JS 应该是静态的

2. JS 应该是静态的

我看到过很多程序员喜欢动态的使用JavaScript。他们喜欢像使用服务器端语言如C#, Ruby, Java那样来动态的使用JavaScript。千万不要这么做。你失去了代码着色、语法高亮显示和智能感知的支持。记住，JavaScript 应该属于一个 `.js` 文件(见规则 #1)。

然而，使用JSON引入动态行为。我把这称为JavaScript配置对象模式。具体方法如下：把JSON注入到你应用程序的头部，并根据业务逻辑的需要利用这些数据。你可能会想：“嘿，这违背了规则 #1”。我把 JSON 看作是数据，而不是代码，所以我破例，为了支持静态的、单独的JavaScript文件。

StackOverflow 使用的这种模式，Google 也是。你可以看下他们的代码：



正如你看到的，StackOverflow 注入了一些个人的设置，如 isNoticesTabEnabled。这个简单的JSON代码片段为你使用静态JavaScript文件自定义行为提供了必要的数据支持。为了实现这一点，需要序列号服务器端类为JSON，然后放置在 `<head>` 中。然后你可以在静态的JavaScript文件中根据需要参考这个数据结构，能够使用它，是因为它被注入到 `<head>` 中。

3. JS 应该被压缩

3. JS 应该被压缩

压缩可以减小文件体积，从而提升页面加载速度。记住，性能也是一项功能。因为，为了压缩，你需要把 JS 放到一个单独的文件中(见规则 #1)。压缩JS曾经很麻烦，但现在完全是简单自动化的。有一打的方式可以做到，而Gulp 和 gulp-uglify 是一种低摩擦和自动化的办法。

4. JS 应该位于页面底部

4. JS 应该位于页面底部

如果你把 `<script>` 标签放在 `<head>` 中，它会阻碍页面渲染。位于 `<head>` 中的脚本必须在页面显示前加载，因此把 `<script>` 放在底部的 前面可以先显示页面，而不用等 JS 文件下载完毕。这有助于提升感知性能。如果你的JavaSctipe必须位于 `<head>` 中，可以考虑使用 jQuery 的 `$(document).ready` 这样你的脚本可以等到 DOM 加载完毕后再执行。

5. JS 应该实时的 Linted

5. JS 应该实时的 Linted

Linting 遵循代码风格、发现错别字、有助于避免错误。有很多这样的工具，我建议使用 ESLint。你可以使用 Gulp 的 gulp-eslint 来运行它。Gulp 可以查看你所有的 JS 文件，并在你每次保存的时候运行 linter。另外，你需要把你的 JS 代码放在单独的 .js 文件中才能运行 linter 。

6. JS应该有自动化测试

6. JS应该有自动化测试

在过去的几年中，我们知道了测试的重要性。但它在很大程度上忽略了在JavaScript，直到最近才被重视。现在典型的JavaScript应用需要测试的部分远比你实际手动测试到的要多。使用JavaScript处理这么多的逻辑，关键的是具有自动测试。

您可以通过工具，如 Selenium 自动化集成测试。然而，集成测试往往是脆弱的，所以我建议专注于自动化单元测试。自动化单元测试有多种选择。如果你是新手， 我建议你使用Jasmine，而如果你想要终极配置，可以使用Mocha with Chai。

7. JS 需要封装

7. JS 需要封装

前些年我们了解了全局变量的风险，值得庆幸的是，现在有很多的方法来封装JS：

- Immediately Invoked Function Expressions (aka IIFE)
- Revealing Modules
- AMD (typically via RequireJS)
- CommonJS (used by Node.js, use in browser via Browserify or Webpack)
- ES6 modules

ES6模块是未来。好消息是，虽然在浏览器中还不能很好的支持，但你可以用 Babel 来使用它。

如果你不想 transpile， CommonJS可能是你最佳的选择。由于 Node 使用的 CommonJS 模式，所以你可以使用npm 来下载数千个包。CommonJS 不能在浏览器中运行，所以你可能需要 Browserify，Webpack, or JSPM.

8. JS 依赖应当明确

8. JS 依赖应当明确

这条规则与上述规则紧密相关。一旦你开始封装JavaScript，您需要一个简单的方法来引用其他模块。这就是常说的现代模块系统CommonJS和ES6模块的好处。你只需要在文件顶部指定依赖，就像 Java 或 C# 那样一句声明：

```
//CommonJS
var react = require('react');
//ES6 Modules
import React from 'React'
```

9. Transpile to JS

9. Transpile to JS

最新版本的JavaScript，EcmaScript 2015(被大家熟知的名字是ES6) 官方版本在 6月份发布了。浏览器还不能很好的支持它的很多特性，但这并无关紧要。你可以用 Babel 来体验它的新特性。Babel 把 ES6 transpile 到 ES5，如果你能忍受这么做，你现在就可以享受 ES6 的新特性。JavaScript预计一年发布一次的新版本了，所以你可能一直需要transpiling 。

或者你喜欢强类型？那么你可以考虑 TypeScript。

10. JS应该自动构建

10. JS应该自动构建

我们已经谈到了 linting、压缩、transpilation 和测试。但如何才能让这一切自动发生？很简单：使用自动构建。Gulp 就是这样一个结合了所有功能的工具。不过你也可以选择 Grunt 和 Webpack。或者如果你是一个高手，你也可以使用 npm 来构建。问题的关键是，不要指望人记得手动运行这些东西的，自动化是一个非常棒的选择。

11. 使用框架或者库

11. 使用框架或者库

拿一些现成的东西来用。想保持轻量级？试试Backbone 或 Knockout。 或者 jQuery就够了。想要更多更全功能的？试试 Angular， Ember,，或者 React with Flux。

关键是：

> **不要试图从头开始。**站在巨人的肩膀上。

不管你选择哪个框架，都应该分开你的关注，这就是下一点..

12. JS Should Separate Concerns

12. JS Should Separate Concerns

把 JS代码放到一个文件中的习惯很容易养成，或者盲目跟从你的框架的意见。当你移动到客户端的时候，不要忘记你在服务器端学到的经验教训。

这里并不仅仅意味着就像你在Angular 和 Knockout等 MVC 框架中那样分离模型、视图、控制器。编写JavaScript的时候应该像服务器端开发者那样思考问题。把你的业务逻辑和数据访问分离出来。

这意味着AJAX调用都应该在一个地方。创建一个集中的客户端“数据访问层”。业务逻辑模块应包含纯JavaScript的。这使得逻辑易于重用，易于测试，升级也不受影响。