第 1 章

精华

Good Parts

……我不过略有一些讨人喜欢的地方而已，怎么会有什么迷人的魔力？

——威廉·莎士比亚，《温莎的风流娘儿们（The Merry Wives of Windsor）》

当我还是一个初出茅庐的程序员时，我想了解所用到的语言的每个特性。我写程序时会尝试去使用所有的特性。我觉得这是炫耀的好方法，而我也的确出了不少风头，因为我对各个特性了如指掌，谁有问题我都能解答。

最终，我认定这些特性中有一部分特性带来的麻烦远远超出它们的价值。其中，一些特性是因为规范很不完善，从而可能导致可移植性的问题；一些特性会导致生成难以理解和修改的代码；一些特性促使我的代码风格过于复杂且易于出错；还有一些特性就是设计错误。有时候语言的设计者也会犯错。

大多数编程语言都有精华部分和鸡肋部分。我发现如果只使用精华部分而避免使用鸡肋的部分，我可以成为一个更好的程序员。毕竟，用糟糕的部件怎么可能构建出好东西呢？

标准委员会想要移除一门语言中的缺陷部分，这几乎是不可能的，因为这样做会损害所有依赖于那些鸡肋部分的糟糕的程序。除了在已存在的一大堆缺陷上堆积更多的特性，他们通常无能为力。并且新旧特性并不总是能和谐共处，可能从而产生出更多的鸡肋部分。

但是，你有权力定义你自己的子集。你完全可以基于精华部分去编写更好的程序。

JavaScript 中鸡肋部分的比重超出了预料。在短到令人吃惊的时间里，它从不存在发展到全球采用。它从来没有在实验室里被试用和打磨。当它还非常粗糙时，它就被直接集成到网景的 Navigator 2 浏览器中。随着 Java™的小应用程序（Java™ applets）的失败，JavaScript 变成了默认的“网页语言”。作为一门编程语言，JavaScript 的流行几乎完全不受它的质量的影响。

好在 JavaScript 有一些非常精华的部分。在 JavaScript 中，美丽的、优雅的、富有表现力的语言特性就像珍珠和一堆鱼目混杂在一起一样。JavaScript 最本质的部分被深深地隐藏着，以至于多年来对它的主流观点是：JavaScript 就是一个丑陋的、没用的玩具。本书的目的就是要揭示 JavaScript 中的精华，让大家知道它是一门杰出的动态编程语言。JavaScript 就像一块大理石，我要切除那些不好的特性直到这门语言的真实的本质自我显露出来。我相信

我精雕细琢出来的优雅子集大大地优于这门语言的整体，它更可靠、更易读、更易于维护。

这本书不打算全面描述这门语言。反之，它将专注在精华部分上，同时会偶尔警告要去避免鸡肋的部分。这里将被描述的子集可以用来构造可靠的、易读的大小程序。通过仅专注于精华部分，我们就可以缩短学习时间，增强健壮性，并且还能拯救一些树木（译注1）。

或许只学习精华部分的最大好处就是你可以不用考虑鸡肋的部分。忘掉不好的模式是非常困难的。这是一个非常痛苦的工作，我们中的大多数人都会很不愿意面对。有时候，制定语言的子集是为了让学生更好地学习。但在这里，我制定的 JavaScript 子集是为了让专业人员更好的工作。

1.1 为什么要使用 JavaScript

Why JavaScript?

JavaScript 是一门重要的语言，因为它是 web 浏览器的语言。它与浏览器的结合使它成为世界上最流行的编程语言之一。同时，它也是世界上最被轻视的编程语言之一。浏览器的 API 和文档对象模型（DOM）相当糟糕，导致 JavaScript 遭到不公平的指责。在任何语言中处理 DOM 都是一件痛苦的事情，它的规范制定得很拙劣并且实现互不一致。本书很少涉及 DOM，我认为写一本关于 DOM 的精华的书就像执行一项不可能完成的任务。

JavaScript是最被轻视的语言，因为它不是所谓的主流语言（SOME OTHER LANGUAGE）（译注2）。如果你擅长某些主流语言，但却在一个只支持JavaScript的环境中编程，那么被迫使用JavaScript确是相当令人厌烦的。在那种情形下，大多数人觉得没必要去先学好JavaScript，但结果他们会惊讶地发现，JavaScript跟他们宁愿使用的主流语言有很大不同，而且这些不同至为关键。

JavaScript 令人惊异的事情是，在对这门语言没有太多了解，甚至对编程都没有太多了解的情况下，你也能用它来完成工作。它是一门拥有极强表达能力的语言。当你知道要做什么时，它甚至能表现得更好。编程是很困难的事情。绝不应该在对此一无所知时便开始你的工作。

译注 1：作者这里幽默的暗示这本书只关注精华部分，所以书变薄了，用的纸张少了，就可以少砍伐一些树木。

译注 2：专指一些主流语言，像 C、C++、Java、Perl、Python 等。

1.2 分析 JavaScript

Analyzing JavaScript

JavaScript 建立在一些非常好的想法和少数非常坏的想法之上。

那些非常好的想法包括函数、弱类型、动态对象和一个富有表现力的对象字面量表示法。

那些坏的想法包括基于全局变量的编程模型。

JavaScript的函数是(主要)基于词法作用域(lexical scoping)(译注3)的顶级对象。JavaScript是第一个成为主流的lambda（译注4）语言。实际上，相对Java而言，JavaScript与Lisp（译注5）和Scheme（译注6）有更多的共同点。它是披着C外衣的Lisp。这使得JavaScript成为一个非常强大的语言。

现今大部分编程语言中都流行要求强类型。其原理在于强类型允许编译器在编译时检测错误。我们能越早检测和修复错误，付出的代价就越小。JavaScript 是一门弱类型的语言，所以 JavaScript 编译器不能检测出类型错误。这可能让从强类型语言转向 JavaScript 的开发人员感到恐慌。但事实证明，强类型并不会让你的测试工作轻松。并且我在工作中发现，强类型检查找到的那种错误并不是令我头痛的错误。另一方面，我发现弱类型是自由的。我无须建立复杂的类层次，我永远不用做强制造型，也不用疲于应付类型系统以得到想要的行为。

JavaScript有非常强大的对象字面量表示法。通过列出对象的组成部分，它们就能简单地被创建出来。这种表示法是促使我创立流行的数据交换格式——JSON的灵感（译注7）。（附录E中将会有更多关于JSON的内容。）

原型继承是 JavaScript 中一个有争议的特性。JavaScript 有一个无类别的（class-free）对象系统，在这个系统中，对象直接从其他对象继承属性。这真的很强大，但是对那些被训练使用类去创建对象的程序员们来说，原型继承是一个陌生的概念。如果你尝试对 JavaScript 直接应用基于类的设计模式，你将会遭受挫折。但是，如果你学习使用 JavaScript 的原型本质，那么你的努力将会有所回报。

JavaScript 在关键思想的选择上饱受非议。虽然在大多数情况下，这些选择是合适的。但是有一个选择相当糟糕：JavaScript 依赖于全局变量来进行连接。所有编译单元的所有顶级变量被撮合到一个被称为全局对象的公共命名空间中。这是一件糟糕的事情，因为全局变量是魔鬼，并且在 JavaScript 中它们是基础性的。幸好，我们接下来会看到，JavaScript 也给我们提供了缓解这个问题的处理方法。

译注 3：JavaScript 中的函数是根据词法来划分作用域的，而不是动态地划分作用域的。具体内容参见《JavaScript 权威指南》中译第五版相关章节——8.8.1　词法作用域。

译注 4：Lambda 演算是一套用于研究函数定义、函数应用和递归的形式系统。它对函数式编程有巨大的影响，比如 Lisp 语言、ML 语言和 Haskell 语言。更多详细内容请参见 *http://zh.wikipedia.org/wiki/λ 演算*。

译注 5：LISP（全名 LISt Processor，即链表处理语言），由约翰·麦卡锡在 1960 年左右创造的一种基于 λ 演算的函数式编程语言。更多详细内容请参见 *http://zh.wikipedia.org/wiki/Lisp*。

译注 6：Scheme，一种多范型的编程语言，它是两种 Lisp 主要的方言之一。更多详细内容请见 *http://zh.wikipedia.org/wiki/Scheme*。

译注 7：本书作者也是 JSON(JavaScript Object Notation)的创立者。官方网站中文版网址是 *http://json.org/json-zh.html*。

在少数情况下，我们不能忽略鸡肋的部分。另外还有一些不可避免的糟粕，当涉及这些部分时，我会将它们指出来。它们也被总结在附录 A 中。但是我们将成功地避免本书中提到的大多数鸡肋的部分，它们中的大部分被总结在附录 B 中。如果你想学习那些鸡肋的部分及如何拙劣地使用它们，请参阅任何其他的 JavaScript 书籍。

定义 JavaScript(又称 JScript)的标准是 ECMAScript 编程语言第 3 版 ,它可以从 *http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf* 获得。本书中描述的语言是 ECMAScript 的一个适当的子集。本书并不描述整个语言，因为它不包含鸡肋的部分。这里的处理并没有穷尽，一些模棱两可的情况被回避了。在实践中，你也应该这样。模棱两可的情况会带来风险和麻烦事。

附录 C 描述了一个叫 JSLint 的编程工具，它是一个 JavaScript 解析器，它能分析 JavaScript 问题并报告它包含的缺点。JSLint 提供了一般在 JavaScript 开发中缺少的严谨性。它能让你确信你的程序只包含精华的部分。

JavaScript 是一门有许多差异的语言。它包含很多错误和尖锐的边角 (sharp edges) , 所以你可能会疑惑 :"为什么我要使用 JavaScript？"有两个答案。第一个是你没有选择。Web 已变成一个重要的应用开发平台 ,而 JavaScript 是唯一一门所有浏览器都可以识别的语言。很不幸，Java 在浏览器环境中失败了，否则想用强类型语言的人来说就有其他选择了。但是 Java 确实失败了，而 JavaScript 仍蓬勃发展着，这恰恰说明 JavaScript 确有其过人之处。

另一个答案是，尽管 JavaScript 有缺陷，但是它真的很优秀。它既轻量级又富有表现力。并且一旦你熟练掌握了它，就会发现函数式编程是一件很有趣的事。

但是为了更好地使用这门语言，你必须知道它的局限。我将会无情地揭示它们。不要因此而气馁。这门语言的精华部分足以弥补它鸡肋的不足。

1.3 一个简单的试验场

A Simple Testing Ground

如果你有一个 Web 浏览器和任意一个文本编辑器，那么你就有了运行 JavaScript 程序所需要的一切。首先，请创建一个 HTML 文件，可以命名为 *program.html*：

```
<html><body><pre><script src="program.js">
</script></pre></body></html>
```

接下来，在同一个文件夹内，创建一个脚本文件，可以命名为 *program.js*：

```
document.writeln('Hello, world!');
```

下一步，用你的浏览器打开你的 HTML 文件去查看结果。本书贯彻始终都会用到一个 `method` 方法去定义新方法。下面是它的定义：

```
Function.prototype.method = function (name, func) {
    this.prototype[name] = func;
    return this;
};
```

我会在第 4 章解释它。