第 10 章

优美的特性

Beautiful Features

我让你的脚玷污我的嘴唇，让你的肖像玷污我的眼睛，让你的每一部分玷污我的心，等候着你的答复。你的最忠实的……

——威廉·莎士比亚，《空爱一场（Love's Labor's Lost）》

去年我被邀请为Andy Oram和Greg Wilson的"《Beautiful Code》（译注1）"（O'Reilly出版）一书写一篇文章，这是一本以计算机程序的表达之美为主题的选集。我负责的章节将介绍JavaScript，通过那一部分来证明JavaScript不虚其名，它的确是抽象、强大且有用的。然而，我想避开不谈浏览器和其他适合使用JavaScript的地方。我想要强调其更有分量的内容，以显示它是值得尊敬的语言。

我立即想到Vaughn Pratt的自顶向下的运算符优先级解析器（译注2）（Top Down Operator Precedence parser），我在JSLint中运用了它。在计算机的信息处理技术中，解析是一个重要的主题。一门语言是否具备为其自身编写一个编译器的能力，仍然是对这门语言完整性的一个测试。

我想用 JavaScript 编写的解析 JavaScript 的解析器的全部代码都涵盖在文章中。但是那章仅是 30 章或 40 章之一，我感觉被束缚在能占用的那几页中。更大的困难是大部分读者没有使用 JavaScript 的经验，我也不得不介绍这门语言和它的特色。

所以，我决定提炼这门语言的子集。这样，我就不必解析整个语言，并且也就不需要描述整个语言了。我把这个子集叫做精简的 JavaScript（Simplified JavaScript）。提炼子集并不难：它包括的就是我编写解析器所需要的特性。我在《Beautiful Code（美丽的代码）》一书中是

译注 1：《Beautiful Code》是O'Reilly2007 年 6 月出版的书籍，参见*http://oreilly.com/catalog/9780596510046/*。

译注 2：详细内容请参阅作者的文章 Top Down Operator Precedence parser——*http://javascript.crockford. com/tdop/tdop.html*。

这样描述的。

精简的 JavaScript 里都是好东西，包括以下主要内容。

函数是头等对象

在精简 JavaScript 中，函数是有词法作用域的闭包（lambda）。

基于原型继承的动态对象

对象是无类别的。我们可以通过普通的赋值给任何对象增加一个新成员元素。一个对象可以从另一个对象继承成员元素。

对象字面量和数组字面量

这对创建新的对象和数组来说是一种非常方便的表示法。JavaScript 字面量是数据交换格式 JSON 的灵感之源。

子集包含了精华中最好的部分。尽管它是一门小巧的语言，但它很强大且非常富有表现力。JavaScript 有许多本不应该加入的额外特性，正如你在随后的附录中将看到的那样，它有大量的会带来负面价值的特性。在子集中没有丑陋或糟糕的内容，它们全部都被筛除了。

精简的 JavaScript 不是一个严格的子集。我添加了少许新特性。最简单的是增加了 pi 作为一个简单的常量。 我这么做是为了证明解析器的一个特性。我也展示了一个更好的保留词策略并证明哪些保留词是多余的。在一个函数中，一个单词不能既被用作变量或参数名，又被用作一个语言特性。你可以让某个单词用在其中之一上，并允许程序员自己选择。这会使一门语言易于学习，因为你不必知道你没有使用的特性。并且它会使这门语言易于扩展，因为它无须保留更多的词来增加新特性。

我也增加了块级作用域。块级作用域不是一个必需的特性，但没有它会让有经验的程序员感到困惑。包含块级作用域是因为我预先考虑到解析器可能被用于解析非 JavaScript 语言，并且那些语言能正确地界定作用域。我为这个解析器所写的代码风格不关心块作用域是否可用。我推荐你也采用这种方式来写。

当开始构思本书的时候，我想进一步地发展这个子集，展示如何通过除了排除低价值特性外不做任何改变来获得一个现有的编程语言，并且使它得到有效的改进。

我们看到大量的特性驱动的产品设计，其中特性的成本没有被正确地计算。对于用户来说，某些特性可能有一些负面价值，因为它们使产品更加难以理解和使用。我们发现人们想要的产品其实只要能工作即可。事实证明产生恰好可以工作的设计比集合一大串特性的设计要困难得多。

特性有规定成本、设计成本和开发成本。还有测试成本和可靠性成本。特性越多，某个特性出现问题，或者和其他特性相互干扰的可能性就越大。在软件系统中，存储成本是无足轻重的，但在移动应用中，它又变得重要了。它们抬高了电池的效能成本，因为摩尔定律并不适用于电池。

特性有文档成本。每个特性都会让产品指南变得更厚，从而增加了培训成本。为少数用户提供价值的特性增加了所有用户的成本。所以在设计产品和编程语言时，我们希望直接使用核心的精华部分，是因为这些精华创造了大部分的价值。

在我们使用的产品中，总能找到好的部分。我们喜欢简单，追求简洁易用，但是当产品缺乏这种特性时，就要自己去创造它。微波炉有一大堆特性，但是我只会用烹调和定时，使用定时功能就足够麻烦的了。对于特性驱动型的设计，我们唯有靠找出它的精华并坚持使用，才能更好地应对其复杂性。

如果产品和编程语言被设计得仅有优点，那就太好了。