基于JavaScript的编译算法演示系统

招蕴豪

January 19, 2012

目录

前言

对编译原理的一点感觉

早在两年前，我听说大三有编译原理的课，于是就毫不犹豫地随着减价流买了《编译原理》这本书。这本书一直安安静静地躺在我的书柜上，我好像从来就没有想过主动去翻阅它，生怕它里面深奥的内容会触动我敏感的神经。到了大三，终于得接受这门课的洗礼，这本尘封两年的龙书终于重见天日。

对于编译原理，我有说不出的喜爱。原因有很多，最主要的还是因为我对语言的喜爱。这里的语言偏向于人类语言，我喜欢倾听不同国家的确所特有的韵律，并喜欢观察其语法特点。纵观多种语言，由于中文基本元素（汉字）很多，所以语法结构是比较混乱的。正是因为我们是中国人，我们没有必要过于严谨地去学习中文语法，所以才没有能体会到对于一个完全不会中文的人接触到中文，会是如何地艰辛。当时我就在想，有没有什么办法能真正系统地将这些语法总结出来呢？对于其它语言，我们都可以在书店看到很多相关的语法书，可以说这些语法书在一定程度上总结了该语言总体的语法特点，但是实际上，那是及其不严谨的语法说明，其中存在着诸多特例，以及二义性的描述。但同时我又想，或许正是人类语言中的那些充满二义性的表达，才真正使得人类语言如此迷人。

语法作为指导语言元素使用者能够尽量准确使用语言元素的规则，在人类学习某一门语言的时候充当着及其重要的作用。使得语言初学者能够将其学到的语言元素组合起来，并且能够指引语言学习者能够以一种宏观的角度来观察语言的结构以及性质。从机器的角度而言，语法充当这类似的作用。所以当我第一次接触跟编译相关概念的时候，我非常惊讶，对机器能完成跟人脑类似的工作感到相当神奇。虽然机器所需要识别的语法结构跟人类语言的结构差异巨大，但能向前迈出这样的一步，实在是不容易啊。当然，机器所做的事情比人脑做的事情还多了一些，那就是在理解输入以后还得将其转换为目标代码。

编译原理的课程带给了我很大的快乐，感谢老师允许我用自己的方式来阐述我对编译原理的喜爱以及理解；再有是感谢为编译理论做出过贡献的为人们，你们所

创造出来的知识使我身心愉悦，心情舒畅。

有关本文档

自从大一接触LaTeX之后，我就深深爱上了这个排版系统。无论是我的数学作业还是博客，都是靠它来帮我排版。但由于LaTeX对中文支持比较差，在Archlinux下面配置比较痛苦，所以本来打算用英语来完成这个文档。但考虑到各个方面的原因，还是勉强使用中文来完成。

Chapter 1

系统概述

1.1 动机

由于编译原理这门课程非常重要，能从里面学到的思想有很多，例如流水线以及局部处理局部优化等思想，所以我想更认真地去学习这一门课程。但由于课程内容本身非常抽象，所以我并不能够单单靠阅读书本去理解。虽然用纸笔演算可以解决很多问题，但是当我在演算过程中感受到神奇并不能真切地表现出来，以至于过一段时间以后，我依然对书中的算法感到非常模糊。当然，对算法进行演算无疑是有助于理解，但是由于算法太多，并且复杂，所以有必要借助另外的途径来寻求对其的更深刻的理解。

于是我有了一个想法，那就是将书中的算法实实在在地用程序语言来实现，并找到一种能比较好展示算法演算过程的表达方式展现出来。通过形象的过程描述，或许能使我对算法的流程更为深刻，同时，对算法的实现又可以使我对算法有更深刻的理解。我向来是一个希望究其源而不满足于现象的人，为了这样的目标，我开始了这趟旅途。

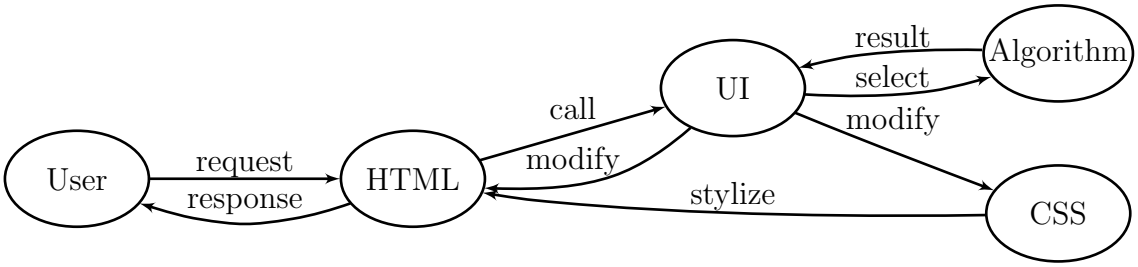
1.2 语言的选择

既然下定了决心，我就开始考虑实现的语言了。由于我的目标并不是去实现一个实实在在的编译器，也不是为了将这些算法的效率提高多少个数量级，我的目标只是为了用一种形象的算法来展示这些算法，所以我没有使用一些比较低级的语言，比如C或者是C++。因为基于这些语言的高效编译器已经有很多了。考虑到我需要的是语言的展示能力，我决定使用JavaScript来完成这个任务。

用JavaScript有一些好处，那就是JavaScript的表达能力比低级语言要强，这样在我实现的过程中就可以忽略掉一些并不需要在算法中关注的细微的现实问题，比如一些低级数据结构的实现；再有，JavaScript可以跟HTML5/CSS3相结合，使得界面的设计更为的便捷与轻松，这样就可以把经历真正放到了对算法的理解上。

1.3 系统结构

这个编译算法演示系统可以分为两个独立的部分，一是核心算法，一是用户界面。而用户界面又可以分为结构，样式以及行为三个部分。鉴于本份文档的任务所在，用户界面并不会花太多的篇幅进行介绍。



从上图可以看出整个演示系统的工作流程。用户向界面发送请求，界面根据用户的输入或者选择来适当调用相应的算法，并将参数传给算法。当相应的算法运算结束后，将结果返回给界面接口，该接口通过修改页面的结构以及样式，来将结果以一种合适的形式来显示给用户，从而完成一次交互。

这种结构实际上是参照网络应用框架的MVC模型，这种模型的好处是容易管理，扩展性强，各部分相互独立，可以单独进行编写以及测试。而整个开发过程则得益于这种模式，可以以增量的方式进行开发，使得基本上没有浪费太多的时间。

Chapter 2

语法分析

语法分析是编译流水线的第二个部分，语法分析器接受词法分析器所提供的词法单元流，根据给定的语法判断词法单元流是否符合语法。语法分析有两种主要的方法，一种是自顶向下语法分析，另一种是自底向上语法分析。在自顶向下语法分析中，语法分析器从语法的开始符号出发，构造一棵词法单元流的语法分析树；而在自底向上的语法分析中，语法分析器根据移入归约原则，将词法单元流转化为语法的初始符号，其过程中的每一步都对应着该词法单元流最右推导的中间过程。

在语法分析部分，对于自顶向下的语法分析，我实现了$LL(1)$预测分析；对于自底向上的语法分析，我实现了SLR以及$LR(1)$。在这些方法的实现过程中，需要到很多的辅助函数，而这些辅助函数都起到了至关重要的作用，下面结合我对语法分析的理解，逐一介绍它们的实现过程。

2.1 语法模型

在语法分析的实现过程中，首当其冲的问题就是为语法选择一个合适的数据结构，一个高效的数据结构非常重要，但高效的数据结构同时又可能难以令人理解。为了均衡高效以及良好阅读性的矛盾，数据结构必须仔细地进行设计。由于JavaScript并没有提供什么数据结构，所以必须自己根据需要来实现。同时，由于数据结构是根据自身来进行实现的，所以比较灵活。下面围绕着书中的表达式文法作为例子，来阐述我的设计过程。一个具有非左递归的文法如下：

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE'|\epsilon \\
T &\rightarrow FT' \\
T' &\rightarrow *FT'|\epsilon \\
F &\rightarrow (E)|id
\end{aligned}
$$

从上面的语法可以看出，语法最直观的一个模型就是数组，或者是链表，其基本元素是产生式。其实在JavaScript里面，只提供了对象以及数组两个比较高级的数据结构，其实数组也是对象，所以在我的整个系统里面，数据结构基本上都是根据简单的数组组合构建而成的。既然语法是一个产生式的数组，那么产生式又应该如何表示呢？观察下面的产生式：

$$
\underbrace{E'}_{\text{产生式头}} \rightarrow \underbrace{+TE'}_{\text{产生式体1}} | \underbrace{\epsilon}_{\text{产生式体2}}
$$

可以观察到产生式由两个部分组成，一个是产生式头，另一个是产生式体，而由于一个产生式可以有多个产生式体，所以可以用数组来存放一个产生式的所有产生式体。在这里有一个问题是需要注意的，那就是**在这里假定每个非终结符号对应一个产生式的数据结构。**简单地说，就是不会出现下面的结构：

$$
\begin{aligned}
E' &\rightarrow +TE' \\
E' &\rightarrow \epsilon
\end{aligned}
$$

当然，这也是一种合法的语法表示形式，是没有理由禁止的，但为了处理的方便，当用户以这样的方式进行输入的时候，一个称为$reduceRedundancy$的函数会消除这种冗余，即将这样的情况转化为上面的那种用“|”来表示的形式。

当然，上面的描述足以表示产生式以及语法两个抽象概念，但是考虑到运算时有一些额外的量是需要的，将这些量绑定到这两个数据结构有助于提高运算效率。例如，在求$First$和$Follow$集的时候，需要查看语法中的终结符号，所以有必要把终结符集合也绑定到语法的数据结构中；又如，在计算预测分析表的时候，需要

随时用到某一个非终结符的$First$和$Follow$集的结果，所以把这两个集合也绑定到产生式中也是非常重要的。基于这些考虑，可以得到下面的语法数据结构和产生式数据结构的伪代码。

```
Grammar:
    terminals   -> Array(String)     # 终结符集合
    productions -> Array(Production) # 产生式集合

Production:
    head   -> Char            # 产生式头
    bodies -> Array(String)   # 产生式体集合
    first  -> Array(String)   # First集
    follow -> Array(String)   # Follow集
```