基于手机中间件的 JavaScript 解释器设计与实现①

Design and Implementation of JavaScript Interpreter Based on Phone Middleware

刘仕坤　金　瓯　(中南大学 信息科学与工程学院 湖南 长沙 410083)

摘　要： 封闭手机系统之上的一种手机中间件平台，采用 Doug Lea 内存管理算法对非常有限的系统内存进行管理，小内存块的有效利用率比较低，针对该手机中间件的内存特点，开发了基于封闭手机系统手机中间件上的一款 JavaScript 解释器系统，该系统分为解释前端和解释后端两部分，采取语法分析驱动词法分析和语义动作策略，并基于语法制导思想构造了 JavaScript 生成抽象语法树的属性文法。相对于其他解释器系统而言，有效地降低了小内存块的大规模使用，提高了内存利用率，效果良好。

关键词： JavaScript 解释器 手机中间件 Doug Lea 内存管理算法 抽象语法树 属性文法

1　引言

目前国外与国内市场上手机可以分为两类：SMART PHONE 和 FEATURE PHONE，SMART PHONE 硬件配置高，特点在于采用了开放式的智能的操作系统，例如 SYMBIAN、WINDOWS MOBILE，用户在终端可以享受多层次的应用体验。而 FEATURE PHONE 则采用了相对封闭的实时操作系统，硬件配置相对较低，终端应用方面也乏善可陈。

但是基于一种新型的手机平台中间件技术(VRE 手机中间件)，可以在 FEATURE PHONE 平台上扩展丰富的应用，其中基于 FEATURE PHONE 手机平台的浏览器就是其中之一。而通过 JavaScript 脚本解释器来支持对 JavaScript 脚本语言的解释执行是手机平台浏览器的核心功能之一。

JavaScript 是一种轻型的、解释型的程序设计语言，而且具有面向对象的能力。该语言的通用核心已经嵌入了 Netscape、Internet Explorer 和其他的 PC 平台的 Web 浏览器中，而且它能用表示 Web 浏览器窗口及其内容的对象使 Web 程序设计增色不少。JavaScript 的客户端版本把可执行的内容添加到了网页中，这样一来，网页就不再是静态的 HTML 了，而是包含与用户进行交互的程序、控制浏览器的程序以及动态创建 HTML 内容的程序 [1]。

2　中间件内存特点

在 FEATURE PHONE 上，一般 RAM/ROM 资源非常有限，除去手机系统自身需要之外，可供第三方手机应用使用的内存资源非常有限，供 VRE 中间件可支配的 RAM 一般只有 600K 连续内存空间，甚至更少。同时，为了对基于 VRE 之上的应用提供灵活的内存使用接口，VRE 借助 Doug Lea 内存管理算法对自己的连续内存空间进行动态分配管理。

在 Doug Lea 内存管理算法中，分配的每一个内存块，都要耗费 8 字节的簿记开销(Book Keep Overhead)，并且内存块大小 8 字节对齐。

它的内存块头部结构如下：

```
struct malloc_chunk {
size_t                prev_foot;
size_t                head;
struct malloc_chunk*      fd;
struct malloc_chunk*      bk;
};
```

① 收稿时间:2008-10-10

Pre_foot 和 head 是内存块头部管理结构。

比如，当申请的内存小于 8 字节时，实际申请得到的字节大小是 16 字节，内存有效利用率仅为 50%，Doug Lea 内存管理算法的这种特性，使得在大规模使用小内存节点时会引起严重的内存浪费。

基于 Doug Lea 内存管理算法开发 JavaScript 解释器，应尽量避免小内存块的大规模使用。

3 解释器总体结构

基于手机中间件平台 VRE 设计的 JavaScript 解释器系统的结构如图 1 所示。

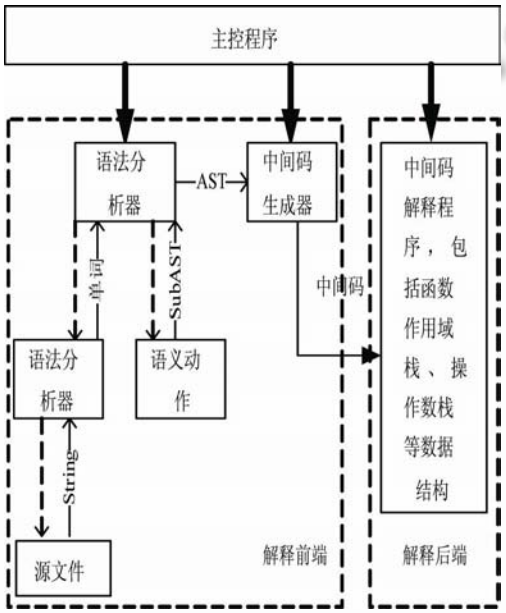

图 1　JavaScript 解释器系统结构图

解释器系统分为前端和后端两部分。系统前端中，选用词法分析模块驱动词法分析和语法动作模块，相对于有些解释器结构中采用的单词链表和语句链表作为前端解释的载体的策略[2]，极大地节省了小内存大规模的申请使用。前端系统通过语法分析模块中分析动作的驱动，调用词法分析模块中产生式归约所需要的单词符号，当一个产生式归约成功后，调用语法动作模块生成抽象语法子树。当一个可执行语句集归约完毕后，与之对应的抽象语法树(AST)构造完毕。由 JavaScript 解释器主控模块对抽象语法树进行遍历，生成自定义格式的中间字节码。对后端系统而言，前端系统产生的中间代码即为所需的目标码，对其进行解释执行并输出结果即可。

前端系统的词法分析模块和语法分析模块所使用的文法是 ECMA-262 第二版中的文法。ECMA-262 是欧洲计算机制造商协会(ECMA)制定的脚本预研的国际标准。JavaScript V1.3 完全兼容 ECMA-262，而 ECMA-262 第二版只是修改了第一版中的一些错误。

在前端系统中，根据 JavaScript 词法规则构造确定有限自动机(DFA)，由确定有限自动机可以很容易地构造词法分析器。语法分析和语义部分则采用 LALR 分析方法和语法制导翻译方法。

前端系统产生的中间代码采用自定义的字节码。采用字节码的好处是字节码与平台无关，在不同的平台上使用不同的解释器对它进行解释执行，即可实现在字节码级与各平台兼容，不仅仅局限于 VRE 手机平台，不必对字节码做任何修改 [3]。

4 系统实现中的关键技术

虽然利用 Lex 和 Yacc 等自动化工具可以自动生成可编译执行的 c 语言词法和语法分析器，但是在运行的过程中需要 Lex、Yacc 工具自带的静态库文件，而 Lex、Yacc 工具库文件相对较大，例如工具 Parser Generator Win32 版本的 Lib 文件大约 7M，鉴于 VRE 可用内存尺寸方面的限制，不适合移植。因此要在内存资源非常有限的手机设备中实现词法语法分析器，不能借助于自动工具。

要实现一个 JavaScript 解释器，从本质上来说，也就是实现图 1 所示的体系结构。要实现这个解释器，需要解决各种技术问题，下面加以介绍。

4.1 词法分析器构造

根据 ECMA-262 中 JavaScript 词法规则定义，构造出 DFA 的状态转换矩阵表，可以比较容易地构造出词法分析器。

词法分析器以 JavaScript 源代码字符串作为输入，每次将源码字符串中的下一个单词符号信息返回给语法分析器。鉴于 JavaScript 语言基于其对象的特性，词法分析器返回给语法分析器的单词符号信息应

该兼容基本数据类型和对象类型。符号信息数据类型定义如下：

```
typedef struct lexer_word_info{
    int   type;     //单词符号类型
    union{
      int keyword;     //关键字
      int intvalue;     //整型数值
      float fvalue;     //浮点型数值
      char * strvalue; //变量名、对象名、函数名或字符串等指针
      }wvalue;
      }lexer_word_info;
```

词法分析器在返回给语法分析模块结果之前，将识别结果写入符号表中。

在该 JavaScript 解释器系统中，词法分析器被语法分析器驱动。

4.2 语法分析器构造

解释器的前端系统的构架设计有多种方案，有的解决方法是把词法分析器、语法分析器和语义动作部件作为平行的部件来对待，即首先对整个源文件进行词法分析得到单词串链表，然后语法分析器逐个读取单词串进行词法分析并得到语句序列链表，最后语义动作对整个语句序列链表采取对应的语义处理[2]。这种设计思路需要单词链表和语句链表作为中间数据结构来支撑，会造成小内存块的大规模占用，鉴于 Doug Lea 内存管理算法的特性，这种策略并不适合在手机中间件采用。

该 JavaScript 解释器在解释前端的设计上，语法分析器在逻辑上高于词法分析器和语义动作部件，采取“即用即取”的策略，节省了单词链表等中间数据结构的内存开销。

语法分析模块负责驱动语法分析和语义动作两个模块，在通过 LALR 分析法进行产生式归约的过程中，逐个获取 JavaScript 单词符号并根据每条产生式的语义规则调用对应的语义动作。

面向词法分析与语义动作驱动的 LALR 语法分析器算法如下：

```
Begin
 初始化LALR归约状态为0状态；
 Do
  调用词法分析，得到一个单词符号word；
  If(word有效&&LALR归约未到acc态)
   按照LALR分析表，
   基于符号栈做移进、归约和跳转等操作，
   有归约时调用对应语义动作；
 Else
  If(LALR归约到acc态)
   Break；
  Else
   Error；
 While(word有效&&LALR归约未到acc态)
End
```

LALR 语法分析过程中，基于符号栈和语法树指针栈进行操作。终结符非终结符的操作都基于一个符号栈来进行；在分析过程中生成的抽象语法树指针放在与符号栈对应的语法树指针栈中。

4.3 语义动作模块构造

语法分析器运行过程中，当某个产生式被归约时，调用语义动作模块生成产生式的抽象语法子树(Sub AST)。

本系统参照语法制导语义分析方法，构造了 JavaScript 语法生成抽象语法树的属性文法，以 JavaScript 中对象加法运算语法的属性文法为例，举例如下：

```
A -> A + A
   A.ptr = NewASTNode(+,2,A1.ptr,A2.ptr)
A -> A.b
   b.ptr = NewASTNode(.,1,null,null)
   *(b.ptr) .value = b.value
   A.ptr = NewASTNode(.,2,A1.ptr,b.ptr)
A -> b
  A.ptr = NewASTNode(null,1,null,null)
  *(A.ptr) .value = b.value
```

其中，A 表示对象，b 表示对象或属性。属性文法中，如果产生式右侧有与左侧同名的非终结符，则对右侧的非终结符依次缀以编号表示。函数原型 ASTNode * NewASTNode(int op_type, int op_count, ASTNode *oprand_node, …)，它的功能是生成抽象语法树的子节点并返回节点指针。

基于语法制导翻译法的思想，在每个产生式进行归约时，依照它们对应的语义文法，每归约产生一个非终结符，它所对应的抽象语法子树都要弹入语法分析器中维护的指针栈中。一旦一个产生式被归约，符号栈顶几个元素即被弹出，归约产生的非终结符弹入，同时抽象语法树指针栈顶对应几个元素也被弹出后指针赋值后再弹入。

对抽象语法树进行后序遍历就可以生成中间码，在此不做赘述。

4.4 解释后端设计

解释器的后端系统实际上是一个解释自定义字节码的虚拟机。后端虚拟机系统是由字节码解释控制程序和函数调用关系栈、操作数栈、变量数据栈和对象信息记录表五部分组成的。

其中，字节码字节控制程序根据字节码的操作符码，对操作数进行操作，比如进栈出栈等。

函数调用关系栈记录了 JavaScript 各种可执行代码的相互调用关系，每当进入一个新的可执行代码，就在函数调用关系栈栈顶压入当前可执行代码的的执行上下文环境信息。

操作数栈主要用于进行各种计算和中间结果的保存。中间字节码的所有计算均基于它来进行。

变量数据栈用于记录字节码执行时的变量信息。每当进入一个新的可执行代码，就在变量数据栈压入当前可执行代码的局部变量信息(包括变量名称、类型、值等)，一旦当前的可执行代码执行完毕，变量数据栈顶元素即被弹出，同时函数调用关系栈和操作数栈中相关元素弹出。

对象信息记录表是存储各种 JavaScript 对象的内存区。由于手机中间件平台的可用内存资源有限和小内存块利用不充分的特点，将固定长度的结构数组作为对象信息记录表。对象记录表中记录包括对象的属性个数、对象的属性信息等。在 JavaScript 中，对象的方法也被视作对象的属性。

5 结束语

本文中介绍的基于封闭手机系统手机中间件的 JavaScript 解释器系统采用了解释前端与解释后端分离的策略，中间代码采用自定义的字节码。有更好的移植性，与手机中间件跨平台的特性恰好吻合。前端系统中，针对该中间件小内存块有效利用率低的特点，采用了语法分析器驱动词法分析和语义动作的策略，避免了小内存块的大规模使用，语义分析部分采用了面向构造抽象语法树的语法制导翻译法。同时，如何设计内存利用紧凑的抽象语法树节点结构以及高效的抽象语法树遍历算法是一个可以继续研究的问题，并且中间字节码的设计及后端解释系统都有值得优化的地方。

参考文献

1 Flanagan D. 张铭泽,等译.JavaScript 权威指南.第 4 版.北京:机械工业出版社, 2003:7－8.

2 张军林,阳富民,胡贯荣.JavaScript 语言解释器的设计与实现.计算机工程与应用,2003,30:124－125.

3 王宜春,李蜀瑜,曹清,吴健.ECMAScript 解释器的中间代码生成技术研究.计算机工程与应用,2003,18:135－136.

4 吴作顺,窦文华.几个常用解释器的性能分析.计算机工程与科学, 2002,24(4):83－84.

5 ECMA Standardizing Information and Communication Systems. 3rd Edition.1999.

6 陈火旺,刘春林,谭庆平,赵克佳,刘越.程序设计语言编译原理.第 3 版.北京:国防工业出版社, 2000.