轻量级脚本引擎的设计与实现

张贝克,符盛宝
(北京化工大学 安全科学与监控工程中心,北京 100029)

摘 要:通过嵌入脚本引擎为应用程序提供脚本支持是实现应用程序可定制和可扩展的有效方法,但现存的脚本语言难于掌握,引擎庞大使应用程序的效率降低。为了解决该问题,设计了语法简单易学的脚本语言 Vblet,实现了 Vblet 的轻量级脚本引擎。该引擎支持脚本无缝地使用应用程序实现的类和函数,并具有很好的性能。

关键词:脚本引擎;脚本语言;可扩展;可定制;二次开发
中图分类号:TP314 文献标识码:A 文章编号:1674-7720(2011)15-0008-04

Design and implementation of a lightweight scripting engine

Zhang Beike, Fu Shengbao
(Safety Science and Monitoring Engineering Center, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: Scripting support for an application by embedding a scripting engine is an effective method to make the application more extendable and more customizable, but either the existing scripting languages are too hard to learn, or their scripting engines are too bulky and effect the application's performance. To try to resolve this problem, this paper present the design and implementation of a easy learning scripting language Vblet and it's lightweight scripting engine. The scripting engine support Vblet scripts to call the functions and create the instances of classes transparently which are registered by the application and has high performance.

Key words: scripting engine; scripting language; extendibility; customizability; secondary development

脚本语言凭借强大的描述能力和灵活的语法结构,使得为应用程序提供脚本支持从而进行混合语言开发成为实现可扩展和可定制的有效方案[1]。出于稳定性和开发时间限制的考虑,开发人员倾向于嵌入现有脚本引擎的方法为应用程序提供脚本支持,如嵌入 Python 引擎为应用程序提供 Python 脚本支持,或使用 Microsoft 提供的 ActiveX Scripting 技术为应用程序嵌入 VBScript 引擎或 JavaScript 引擎提供相应的脚本支持。但是这样方法灵活性较差,应用程序必须接受现有脚本引擎的体积和性能要求,这对运行在低硬件条件下的应用程序,或者是只要求进行简单规则计算的小型应用程序来说,这种方法在效率上没有优势[2]。而且有些现有脚本语言比较难学,使得用户把太多时间花在语言的学习上。因此,需要一个轻型的脚本引擎,能够解释运行一门语法简单易学的脚本语言,该脚本语言对于工程应用领域的非正式程序员,可以经过短时间的学习培训或者不经过学习就能掌握并使用。

针对以上问题,本文在自行设计的脚本语言 Vblet 的基础上,开发实现了 Vblet 的轻型脚本引擎,支持脚本引擎被嵌入在 C++实现的应用程序上。Vblet 语言语法简单,继承了在非专业程序员中具有较高声誉的 VBA 语言,并且借鉴了 Python 语言的部分功能,使得用户能够专注于问题的解决而不是语法的学习上。

1 脚本引擎概述

脚本引擎[3]是一个加载、解释执行脚本,并负责与外界进行交互的程序。脚本引擎一般很少独立存在,而是要嵌入应用程序中以扩展应用程序的行为,这个被嵌入脚本引擎的应用程序称为宿主程序。

嵌入的脚本引擎如图 1 所示。图 1 中,脚本引擎通过某种交互接口,根据脚本源程序描述的逻辑来控制应用系统。根据宿主程序和脚本引擎之间的紧密层次不同,可将通信方式分为:(1)基于二进制接口的通信。(2)

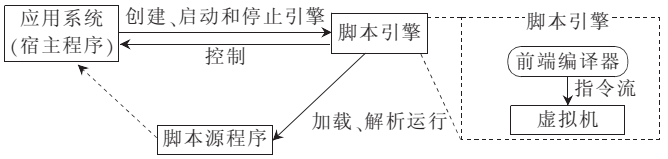

图 1 嵌入的脚本引擎

基于公共运行时环境的通信。(3)基于源码接口的通信。本文实现的基于源码的交互接口,即通信双方基于共同的实现语言,在源代码级上相互调用。

除了交互接口,作为脚本的解释运行平台,脚本引擎包含了一个编译器前端程序,前端程序负责将脚本源代码经过词法分析、语法语义分析后生成字节码格式的指令序列,然而这些指令序列是不能在目标机器上执行的。因此,在脚本引擎的最底层还要一个执行字节码指令的程序,这个程序即称为虚拟机。

2 脚本语言的设计

在开发实现脚本引擎之前,先要确定引擎要解释执行的对象,即脚本语言。本文设计的脚本语言 Vblet 是 VBA(Visual Basic for Applications)的子集,而 VBA 的语法简单易学,在非专业程序员中有很大的用户量,享有很高的声誉。Vblet 简化了 VBA 语法,去掉了 VBA 语法中的一些限制,此外还根据需要扩展了部分功能。下面是 Vblet 不同于 VBA 的一些重要的语法特性:

(1)交互执行。这是借鉴了 Python 语言交互执行的语法特点,使得程序员可以单行执行语句或计算表达式,而不限于一定要把代码封装在代码块中。

(2)不需要变量和参数声明。VBlet 是动态语言,变量的类型由脚本引擎从上下文中确定,变量可以不经过声明就可以使用。

(3)去掉了部分运算符,比如冒号运算符和逗号运算符。

为了提高性能,Vblet 去除了 VBA 中一些库函数的支持,只保留一些在工程应用领域比较常用的数学计算函数。

3 脚本引擎实现方案

Vblet 引擎除了 IDE 的开发使用了 MFC 类库之外,其他模块的实现都是使用标准 C++编写的,这使得 Vblet 引擎只需要重新编写 IDE,或者修改小部分的核心代码就能够移植到其他平台上。

3.1 前端编译程序的实现

前端编译程序将脚本源程序的字符流经过词法分析、语法分析和语义分析后,生成字节码表示的指令流,同时进行语法检查,对语法错误给出提示信息。另外,为了支持断点调试和异常信息显示,每行源程序和生成的字节码指令的对应关系也要在这里建立。

前端编译器一般可以通过一些自动生成工具生成,但是这些自动生成的代码效率都不够高或者不好阅读,因此本文采用手写的方式实现前端编译程序。前端程序由 Scanner 类和 Compiler 类两大主要模块组成。Scanner 类主要负责源程序的词法分析,它根据规定的词法规则把源程序拆分成词法单元,并进行词法检查。Compiler 类则充当语法分析、语义分析和字节码生成,而且这三者一步完成,中间不产生任何数据。另外,语法分析和语义分析出现的错误由类 Parse_error 负责处理。前端编译器序列图如图 2 所示。

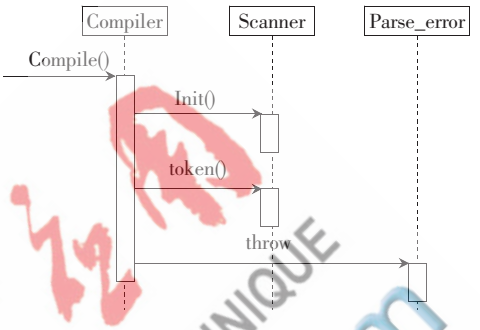

图 2 前端编译器序列图

Vblet 语法分析采用自顶向下的预测分析法,驱动 Scanner 对象的 token()成员函数为其产生一个词法单元,当需要后退时使用 Scanner 对象的 stoken () 方法保存一个不符合当前产生式规则的词法单元,以便下一个产生式规则的分析。类 Compiler 只包含一个 public 权限的成员函数 Compile(),当 Compile()被调用时,将生成的对应编译单元的字节码序列和符号信息、常量数据封装在 ByteCode 对象中,并返回给被调用者。

3.2 虚拟机的实现

Vblet 虚拟机是一个模拟的运行时环境,是对 Vblet 脚本逻辑做出响应的地方。根据体系结构的不同,虚拟机分为如下两种类型:(1) 寄存器虚拟机;(2) 堆栈虚拟机。寄存器虚拟机具有相对较高的执行效率,但实现机制复杂。而堆栈虚拟机实现起来则相对比较简单,但是需要付出一定的性能代价。堆栈虚拟机由于 Java 和 Python 的成功而被证明它在模拟计算平台上的优势[4-5]。因此,本文也将采用堆栈虚拟机作为 Vblet 脚本引擎的计算平台。

如图 3 所示,除了模拟处理器执行字节码指令外,Vblet 虚拟机还包含了 1 个堆栈和 PC、SP、FP 3 个寄存器。堆栈是虚拟机的运行时栈,是函数调用和保存中间变量的地方,是整个虚拟机最核心的数据结构。程序计

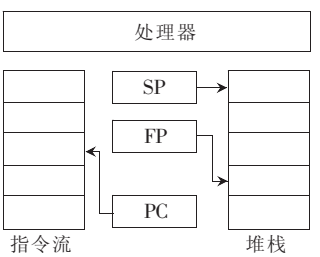

图 3 Vblet 虚拟机的体系结构

数器寄存器 PC 是记录下一条要执行指令的序号；栈顶寄存器 SP 是在堆栈变化的过程中保存堆栈的顶部位置；帧指针寄存器 FP 则相当于真实处理机的基址寄存器 BX，保存当前函数工作栈的栈底位置。这些数据结构表示了整个 Vblet 虚拟机的运行时环境。

Vblet 是一种动态语言，所有数据类型的数据值在 Vblet 虚拟机中只以一种数据结构 VALUE 存在，VALUE 的定义如下：

```
struct VALUE
{
    short int v_type;                    //数据类型
    union V
    {
        bool v_bool;
        int v_integer;
        double v_float;
        string* v_string;          //指向字符串类型数据
        ARRAY v_array;
     //指向数组数据，ARRAY 是封装 VALUE 数组的类
        CODE* v_code;
            //指向字节码；CODE 是封装字节码的类
        VlExtObject* v_extobj;
                              //指向用户注册类的对象
    } v;
    …  //操作 VALUE 和各种数据类型间转换的函数
}
```

在 VALUE 结构体中，成员 v_type 表示了当前 VALUE 对象保存的数据类型。VALUE 不仅封装了 Vblet 的所有基本数据类型，也封装了数组和字节码指令流等。实际上，虚拟机的堆栈和 SP、FP 寄存器所保存的就是 VALUE 对象或是 VALUE 对象的引用。

Vblet 虚拟机在执行字节码指令的过程中，要经常读写堆栈数据。因此，为了提高堆栈读写操作的速度，从而提高虚拟机性能，在实现过程中，定义了如下宏来进行堆栈操作：

```
#define PUSH(v)    (*(++sp))=v      //将值 v 压入栈顶
#define POP        (--sp)                  //弹 s 出栈顶
#define PUSHN(n)   (sp+=(n))   //将 n 个未定义的值压栈
#define POPN(n)    (sp-=(n))   //从栈中弹出 n 个元素
#define SP(n)      (*(sp-(n)))
                    //取栈顶一个第 n 个元素的值
#define FP(n)      (*(fp-(n)))
                  //取帧基地址以下第 n 个元素的值
```

Vitual 虚拟机指令系统共有 16 条数据传输指令、21 条运算指令、5 条转移指令和 5 条支持调试、异常和错误处理的指令，而执行指令的机构——处理器则由函数 interpret()来模拟。interpret()函数从指令序列中逐条取得操作码指令，根据操作码的不同调用各自的处理函数。整个虚拟机的实现封装在 VVM 类中。

3.3 集成接口的实现

脚本引擎的集成接口是指将脚本引擎嵌入应用程序中扩展后者功能时，负责两者之间通信的 API。所谓脚本引擎与应用程序的通信，是指脚本引擎以动态库或静态库的形式被加载进应用程序中，应用程序向脚本引擎开放特定的全局函数和类及其属性、方法，使脚本引擎可以调用这些全局函数、创建这些类的实例，并且通过该实例实现属性的访问和方法的调用。

为了实现全局函数和类的注册，需要一些数据结构来表示注册对象的信息，以便脚本引擎能够识别并使用注册对象。相对来说，描述函数的信息比较简单，只需要一个函数名和一个函数指针，函数指针的定义如下：

```
#define VALUE(*extfuncptr)(ARRAY)
```

可见，extfuncptr 是指向以 VALUE 数组为参数、返回一个 VALUE 值的函数，extfuncptr 函数指针统一了参数类型、个数和返回值类型不同的所有函数声明。因此，需要把用一个能够被 extfuncptr 指向的全局函数将注册函数“包装”起来。在包装函数中，需要将 Vblet 脚本传递过来的 VALUE 参数转换为 C++数据类型的参数，并调用注册函数取得 C++数据类型的返回值，再将返回值转换成 VALUE 值返回给脚本引擎。

一个类的类信息包括类名、大小、初始化函数指针、类注册的方法和属性，用结构体 VLEXTCLASSINFO 表示，如图 4 所示。

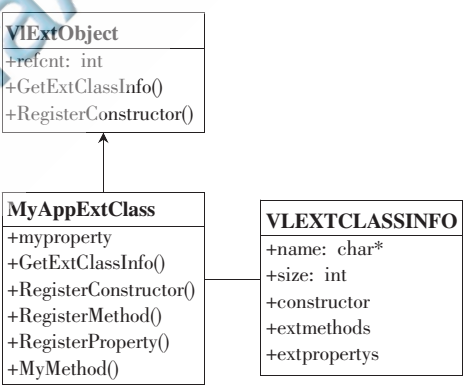

图 4 注册类 MyAppExtClass 与 VlExtObject、VLEXTCLASSINFO 三者之间关系的类图

此外，还需要一些机制使得脚本引擎能够引用脚本创建的注册类的实例对象。把这个表示所有注册类对象的“始祖”称为 VlExtObject。VlExtObject 包含一个表示引用计数的成员变量 refcnt 和两个纯虚函数 GetExtClassInfo()和 RegisterConstructor()。任何应用程序需要向脚本引擎注册的自定义类都必须继承自类 VlExtObject，每个注册类包含一个静态的 VLEXTCLASSINFO 实例，函数 GetExtClassInfo () 返回该 VLEXTCLASSINFO 实例，函数 RegisterConstructor()将向该 VLEXTCLASSINFO 实例指定类实例的初始化函数——构造器的包装函数。此外，应用

程序注册类还可以有自己的函数来向VLEXTCLASSINFO 实例对象添加自己的方法和属性。在脚本引擎内部,通过使用指针来操作注册类的实例。因此,在脚本中当这样的对象被复制时,实际上复制的是对象的指针,并且该对象的引用计数refcnt 加 1, 而当对象的引用在脚本中离开其作用域时,并不立即销毁对象,而是将 refcnt 减 1 后如果为零才会使用 delete 将其销毁。

整个脚本引擎被封装成单件模式的 CVbletEngine 类中,该类包含了脚本引擎的启动、初始化、脚本的运行和停止等操作。CVbletEngine 和集成接口的声明对应用程序可见, 而集成接口始终与脚本引擎的虚拟机部分连接,在虚拟机指令集足够完善的情况下,前端编译器和集成接口的分离使得前端编译器对应用程序是透明的,这样,当需要增加脚本功能的时候,应用程序可以不做修改。

4 与 Python 的性能比较

为了测试 Vblet 脚本引擎是否达到轻量级的要求,在 Intel 586 PC 机上用该脚本引擎解释执行如下这段 Vblet 脚本代码:

```
function main(void)
  i=0
  a=121
  b=212
  while i<100000000
    a=a+b
    a=a-b
    a=a*b
    a=a/b
    i=i+1
  ewhile
end funtion
```

同时在同一平台上用 Cpython2.6 解释执行对应的 Python 程序,最后通过对两者的初始化时间、编译时间、运行时间、内存使用量和生成字节码个数进行了比较,结果如表 1 所示。

表 1 Vblet 脚本引擎和 CPython2.6 的性能比较

	初始化时间 /10^{-6}s	编译时间 /10^{-6}s	运行时间/s	内存使用量/KB	生成字节码个数
Vblet	602.869 9	615.720 7	55.83 543 942	1 484	32
CPython	31 874.721 5	251.987 3	61.16 282 963	4 524	19

从表 1 可以看出, 虽然 Vblet 的编译时间高于 CPython,但是初始化时间要远远优于 Cpython。这是因为 Python 包含了强大的功能模块,引擎运行前需要加载这些模块,并初始化复杂的运行时环境和类型环境。另外,由于 Vblet 的类型机制比 Python 简单, 虚拟机在数据的存取上比 CPython 更快, 即使生成的字节码个数略多于 CPython, 也能达到更优的运行时间。再加上 Vblet 在内存上的优势, 表明 Vblet 完全可以作为一个轻型的脚本引擎嵌入在应用程序中。

基于嵌入或扩展脚本的混合语言编程是实现可定制工程应用系统的有效方法。本文通过借鉴 VBA 和 Python 的语法特点设计了语法简单易学的脚本语言 Vblet, 设计并实现了 Vblet 的基于堆栈虚拟机的轻量级脚本引擎。测试结果表明,脚本引擎能够正确运行 Vblet 代码,占有较小的内存空间,在进行简单的规则计算时具有明显的执行效率。同时,脚本引擎对被嵌入 C++应用程序的支持,使得脚本能够透明地使用应用程序注册的类和函数,从而达到增强应用系统灵活性、可定制性和扩展性的目的。

参考文献

[1] JOHN K. Ousterhout scripting: higher-level programming for the 21st century[J]. IEEE Computer Magazine,1998,31(3).

[2] XIE Q, LIU J, CHOU P H. Tapper: a lightweight scripting engine for highly constrained wireless sensor nodes [C]. Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on, 2006:342-349.

[3] Alex Varanese.Game Scripting Mastery [M]. Premier Press, 2003.

[4] LINDHOLM T, YELLIN F. Java virtual machine specification [M].Boston, MA, USA:Addison-Wesley Longman Publishing Co., Inc,1999.

[5] ALFRED R S, AHO V, JEFFREY D. Ullman. compilers: principles, techniques, and tools [M]. 2rd. Boston: Pearson/Addison Wesley, 2007.

(收稿日期:2011-03-28)

作者简介:

张贝克,男,1976 年生,博士,教授,主要研究方向:系统仿真技术及安全系统工程。

符盛宝,男,1985 年生,硕士研究生,主要研究方向:计算机语言与编译技术。