Parrot 动态编译及其到 Java 的移植

戴莉莉　张　昱　张　磊
（中国科学技术大学计算机科学技术系　安徽 合肥 230027）

摘　要　动态编译是 Perl 的主要特征之一，Perl6 是 Perl 的下一代版本，而 Parrot 作为 Perl6 解释器的实现，提供了更强大的动态编译技术。在深入分析 Parrot 动态编译技术的基础上，总结其到 Java 移植的关键点，重点讨论了 PMC 的移植方法，以及用于动态编译的编译器的获得和保存、注册、运行等的移植方法。阐述了利用 Java 的 Reflection 技术访问动态装载的类的方法，从而实现动态链接编译器的移植。

关键词　Parrot　移植　动态编译　反射

PARROT RUNTIME COMPILATION AND ITS PORTING TO JAVA

Dai Lili　Zhang Yu　Zhang Lei
(*Department of Computer Science, University of Science & Technology of China, Hefei 230027, Anhui, China*)

Abstract　Runtime compilation technique is one of Perl main features. Perl6 is the next vision of Perl, and Parrot is an implementation of Perl6 interpreter and provides stronger runtime compilation technique. Based on anatomizing Parrot runtime compilation technique, the keys of porting it to Java are analyzed and the porting plan of PMC and the manipulations on the compilers for runtime compilation are emphasized on, such as getting and saving, registering, running and so on. The way to implement and access dynamic linked compiler with the help of Java Reflection technique in order to realize the porting of the compilers for runtime compilation is expained.

Keywords　Parrot　Porting　Runtime compilation　Reflection

0　引　言

Perl 语言广泛应用于系统管理、Web 开发、网络编程等领域，目前发行的主版本为 Perl5。Perl6 作为它的下一代版本，着眼于解决 Perl5 解释器难以维护的问题，并增加了许多新特性[1]。Perl6 的编译运行体系被划分为四个模块：解析器、编译器、优化器和解释器。解析器将 Perl 源程序解析成语法树，编译器接受语法树生成字节码（Parrot ByteCode，PBC）文件，经过优化器优化，交给解释器执行。Parrot 是 Perl6 解释器的一个实现[2]。动态编译功能是 Perl 被广泛应用的原因之一，它是指在 Perl 程序运行时通过某些指令将字符串或文件作为 Perl 程序编译执行的功能。Parrot 实现的动态编译功能更为强大，它不仅可以在运行时编译执行 Perl 程序，也可以通过指定编译器编译执行其它程序。

随着 Java 跨平台优势的日益体现，目前已有很多研究工作围绕着 Java 虚拟机（JVM）相关的软硬件系统展开。在发展 JVM 的同时，也需要考虑在此平台上应用软件的开发和移植。为了避免重复开发已有系统，非 Java 语言到 JVM 的移植正得到广泛的研究，其中也包括 Perl 到 JVM 的移植[3]。在 Perl 移植中，动态编译的实现始终是难点之一。本文将以 Win32 平台下 Parrot v0.1.0[4] 为例，讨论 Perl6 到 JVM 移植中动态编译的实现方案。为简便起见，本文称 Parrot 的 C 语言实现版本为 CParrot，Java 版本为 JParrot。

1　CParrot 的运行机制

CParrot 负责装载并运行 PBC 文件，运行时需要维护各种信息，包括全局符号表、堆栈及当前运行的 PBC 文件等。PBC 文件由目录段、修正段、常量段、代码段等构成。CParrot 装载 PCB 文件时须将其解析成内部结构 PackFile，并重定位修正段中的地址信息，而后执行代码段中的指令流。CParrot 是基于寄存器的，它为每种内部的基本数据类型提供一组寄存器，当前有整型、浮点型、字符串和 PMC（Parrot Magic Cookie）四种基本数据类型。

```
struct PMC {
    pobj_t obj;   // 各类数据值的共用结构
    VTABLE *vtable; // 方法表指针
    DPOINTER *data;
    PMC *metadata;
    SYNC *synchronize;
    PMC *next_for_GC;
};
```

图 1　PMC 类型定义

图 1 是 PMC 的数据类型定义，它由数据区 obj 和方法表指针 vtable 等构成。obj 由一个共用体类型实现，保存各种数值；vtable 指向一组可以动态绑定的、处理 PMC 数据的函数入口。CParrot 定义有 50 种不同类型的 PMC，除用于表示 Perl6 中的所

收稿日期：2005－03－07。**戴莉莉**，硕士生，主研领域：程序设计语言理论与实现技术。

有数据类型(包括标量、数组,散列等)外,还用于表示内部的特殊模块,如支持动态编译的 Compiler 和 Eval 等类型。

Compiler 结构如图 2,其数据区保存编译器主函数的入口地址及其调用接口的入口地址;方法表则保存 invoke( )等函数入口。调用接口是 CParrot 提供的在运行时调用动态绑定的本地 C 函数的一种有效手段。根据函数原型不同,CParrot 提供不同的调用接口。调用接口从寄存器中取出参数,调用 C 函数,而后将返回值放回寄存器。在 Compiler 中,invoke( )就是通过调用接口调用编译器的。

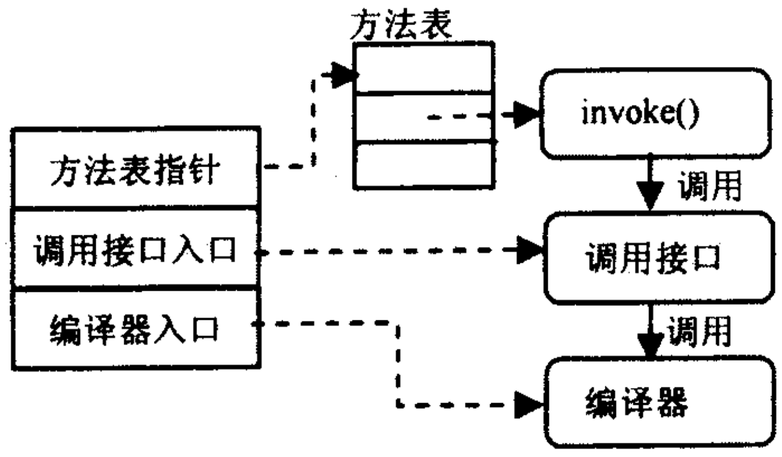

图 2　Compiler 的结构

Eval 的数据区保存动态编译得到 PackFile,方法表保存了 invoke( )等函数入口。此时的 invoke( )负责从当前指令跳转执行 Eval 保存的 PackFile。跳转前,invoke( )保存 CParrot 中当前的 PackFile、代码段和指令等信息,以便在 Eval 保存的 PackFile 执行完毕后恢复。

2　CParrot 中动态编译的实现[4]

2.1　编译器的获得和保存

CParrot 中,用于动态编译的编译器可以通过三种途径获得:其一,动态装载 PBC 格式的编译器,在运行时将其解析到 PackFile 结构并链接至当前运行的 PackFile。其二,静态链接的编译器,CParrot 在初始化时使用 Compiler 类型 PMC 保存了几个内置编译器及其调用接口的入口地址。其三,编译器被编译成动态链接库,CParrot 在运行时按名查找并装载库文件,然后将其保存在 Compiler 类型的 PMC 中。

2.2　编译器注册

CParrot 维护一个特殊的 PMC 结构以保存所有注册的编译器。3.1 提及的三种编译器注册方法各不相同。动态装载 PBC 格式的编译器需要在装入编译器后,通过 Parrot 汇编指令 compreg 指定编译器名称,显式地注册编译器。静态链接的编译器需要在 CParrot 初始化时,对每个编译器调用 CParrot 提供的注册函数。动态链接的编译器需要在装入时,查找库文件并执行其中的初始化函数(它将调用注册函数)完成注册。

2.3　编译器的运行

完成注册后即可通过 Parrot 汇编指令 compile 运行编译器。compiler 指令的格式为:

compile DEST,COMPILER,SOURCE

其中,DEST 和 COMPILER 都是 PMC 类型,前者用于保存编译的结果,后者是所选的编译器。SOURCE 保存待编译的字符串。对于保存在 Compiler 类型 PMC 中的编译器,即静态和动态链接的编译器,compile 指令通过其中的 invoke 函数完成对编译器的调用;编译得到的 PackFile 保存在 PMC 中;随后将该 PMC 定型为 Eval 类型以便调用。对于动态装载的 PBC 格式编译器,compile 指令跳转到所装入的 PackFile 中执行。

3　动态编译的 Java 实现

JParrot 需要实现 CParrot 中三种编译器的获得和保存、注册及运行。由于动态编译需要大量使用各类 PMC 保存编译器、参数及结果等,因此 PMC 的移植是 JParrot 实现的基础。

3.1　PMC 的实现

由图 1 知,实现 PMC 的关键是实现 pobj_t 和方法表。在 JParrot 中,与 pobj_t 对应的类为 PObj,其中定义一个 Object 类型的成员实现对任意类型数值的存储,并定义类 IntVal、NumVal 和 CharArrayBuf 等分别包装 int、double 和 char[ ]等数值。

不同类型 PMC 的数据区中,值的类型和含义也不同,需要有相应的一套方法支持对数据区的操作。CParrot 通过函数指针来实现方法的动态设置,JParrot 则先定义基类 VTable 实现 PMC 默认的方法表,其它特定的方法表都由 VTable 派生,如 EvalVTalbe 和 CompilerVTable。图 3 即 JParrot 中部分 PMC 实现的类图。

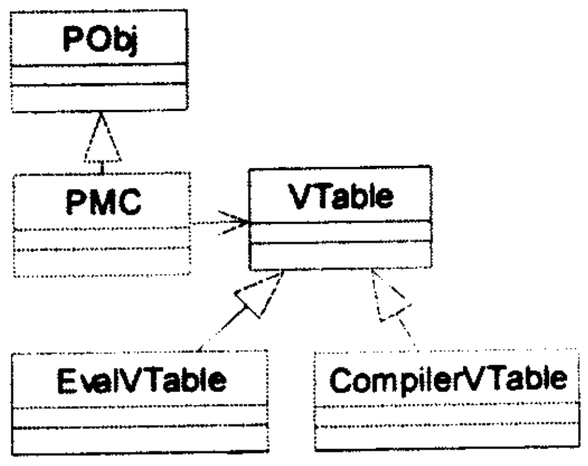

图 3　PMC 的类图

3.2　编译器的获得和保存

在 JParrot 中,实现动态装载 PBC 格式的编译器需要解决被装载文件的重定位、多 PBC 文件的组织、被装载文件中例程的保存和查找这些问题。实现静态链接和动态链接的编译器共同的关键是编译器的保存方法。而动态链接编译器的装载问题也是一个关键。以下分别说明这些问题的解决办法。

a) 动态装载 PBC 格式的编译器

JParrot 读入指定的 PBC 文件,将其解析到内部数据结构 PackFile 中;而后处理修正段中的重定位信息,包括指令标号和子例程。在 CParrot 中重定位的目的是用绝对地址代替相对地址,而 Java 中不可能取得绝对地址,因而 JParrot 仅在重定位的对象中保存相应代码段的引用。当运行 PBC 时,根据代码段和偏移确定指令标号和子例程的位置。

装入和解析 PBC 文件后,需要建立被装入文件与装入者的关联。方法是分别在被装入者 PackFile 的目录段和装入者 PackFile 的目录段中,添加对对方的引用,从而形成树形结构。

PBC 格式编译器的主函数是具有全局作用域的例程,可以被其他 PBC 文件调用。PBC 文件在其常数段中为它定义的每个全局作用域的例程创建一个 PMC,保存相应代码段、起始位置等信息。当 PBC 文件被解析时,JParrot 将这些 PMC 都注册到当前的全局符号表中,该表可根据名称查找例程。

b) 静态和动态链接编译器的保存方法

静态和动态链接的编译器都保存在 Compiler 类型的 PMC 中。JParrot 中有两种可选的保存方法:一种是保存编译器的调用标识(String),另一种是保存编译器的调用接口;两种方法都需要保存相应的编译器类(Class)和编译器主方法名(String)。编译器的调用接口可根据调用标识获取。

Parrot 设有许多调用标识,每一种调用标识对应一种调用

(下转第 74 页)

分析的正确性。同时由于面向对象确保了从分析到设计的完全映射,简单易行的系统建模也提高了成功开发的可能性,并有助于实现系统的增量实现,值得推广。

参 考 文 献

[1] 刘玉林,赵玉兰. 一种基于 UML 的面向对象需求分析方法[J]. 内蒙古大学学报,2005(7):449-452.

[2] 孙昌爱,金茂忠,等. 一种基于 UML 的面向对象需求分析方法[J]. 航空学报,2003(1):75-78.

[3] 孔军,孙怡宁,等. 基于 UML 的系统需求分析[J]. 计算机工程与应用,2003(15):217-219.

[4] Scott W Ambler. The Application Developer's Guide to Object Orientation and the UML[M]. 机械工业出版,2003.

[5] Cay Horstmann. Object-Oriented Design & Patterns[M]. 电子工业出版社,2004.

(上接第 71 页)

接口实现。CParrot 中通过函数指针可以灵活地处理调用接口的动态设置和访问。JParrot 中我们设计统一的调用接口 interface,其中声明了如下方法:

public void parrotCall(Class cls,String name);

该方法由各调用接口实现类实现,其职能是从特定寄存器中取出值,传递给由类 cls 和方法名 name 确定的方法,再调用该方法并将返回值放回寄存器。

c) 动态链接编译器的装载

动态链接编译器的装载有两种不同的实现方法。一种方法是利用 JNI(Java Native Interface)技术实现态链接编译器的装载。这种方法的优点是可以继续利用原有的动态编译器,其缺点是:第一,在 JParrot 和编译器之间,一些数据结构(如 PMC)的传递十分复杂;第二,编译器的执行需要 JParrot 解释器核心类的支持,它们之间的交互也十分复杂。另一种方法是用 Java 重写编译器,利用 Java 的类装载器机制实现动态装载,这样就不需要涉及 JVM 和本地动态链接库之间的交互。

Java 体系允许动态扩展 Java 程序,这个过程包括在运行时决定所使用的类型,装载它们,使用它们。通过传递类名到 java. lang. Class 的 forName()方法,或者用户自定义的类装载器的 loadClass()方法,可以动态扩展 Java 程序。JParrot 提供自定义的类装载器 NJIClassLoader,它从 java. lang. ClassLoader 继承,其覆写(Override)的 loadClass()方法能够在运行时根据文件名装入并解析指定的 Java 类文件。

3.3 编译器注册

JParrot 维护着一个以编译器注册名为 key 的哈希表,用以保存所有注册的编译器,称为编译器表。三种编译器中,动态链接的编译器在装入后通过调用其中的初始化方法完成注册。由于编译器的结构在 JParrot 运行时才能确定,因而如何访问该编译器是注册中的难点。这可以利用 Java 的 Reflection 技术实现。

Reflection 技术是 Java 语言的特征之一。它允许运行中的 Java 程序对自身进行检查,允许一个类使用当它被编译时甚至还不存在的类。Reflection 技术提供 Field、Method 和 Constructor 三个类及一系列方法。通过它们,Java 程序能发现被装入类的域、方法和构造方法等信息,并在安全限定(即对被装入类中公有、私有及受保护组件访问的限定)下使用获得的域、方法和构造方法,以实现对其实例的操作。

调用初始化方法的代码片断如图 4,Class 对象 clasz 中保存着利用 NJIClassLoader 在运行时装入的编译器类信息,编译器类在 JParrot 被编译时不可知。利用 Reflection 技术提供的 getMethod()方法,JParrot 可以根据类名和参数在 clasz 描述的类中找到初始化方法,并为其创建相应的 Method 对象;接着根据 clasz 中的类信息,创建编译器的对象实例;而后利用 Method 类中的 invoke()方法,在编译器实例中执行初始化方法,并通过对象数组 arglist 提供初始化方法的参数。

```
ClassLoader myClassLoader = new NJIClassLoader();
Class clasz = myClassLoader.loadClass(name.toCstring());
Class partypes[] = new Class[1];
partypes[0] = (new PMC()).getClass();
//查找名为"ParrotLib"+classname+"Init", 参数为一个 PMC 的方法
Method meth = clasz.getMethod("ParrotLib"+classname+"Init", partypes);
Object arglist[] = new Object[1];
arglist[0] = null;
//在 calsz 的实例上, 以参数为空, 调用该方法
if (meth!=null) meth.invoke(clasz.newInstance(), arglist);
```

图 4 调用初始化方法

3.4 编译器的运行

由于编译器的保存方法不同,compile 指令对三种编译器需要做不同的处理。

compile 指令调用动态装载的 PBC 格式编译器时,需要根据 Parrot 函数调用约定将编译器所需参数(待编译字符串)保存到指定寄存器中,编译器则在编译完成时将编译结果(返回 PMC)放在指定寄存器中。

对于静态和动态链接的编译器,compile 指令需要利用 Compile 的 invoke()方法调用编译器,而后将返回的 PMC 定型为 Eval 类型。对于 4.1(c)中提到的两种不同保存方法,调用编译器的方法也不相同。保存调用标识时,invoke()方法中有大量的判断分支,它根据不同的调用标识对参数做相应的处理,利用 Reflection 技术查找调用所需方法,而后处理返回值。而保存调用接口的方法时,则只需要取出并调用这一调用接口即可,由调用接口处理参数并调用编译器。可见,前者每次调用相同的编译器时都需要重复选择接口的步骤,而后者在保存编译器时即选择好了调用接口,此后只需直接调用。这是后者对前者的一个改进。

4 结束语

目前,我们已经建立 JParrot 框架,实现了它的核心部分,可以运行部分 PBC 文件,并实现了 Parrot 的动态编译功能。其中利用 Reflection 技术实现动态链接编译器,它允许在运行时加载、探知、使用编译期间完全未知的类文件。因此,较之 CParrot,我们的实现使得动态编译器更易于生成。但由于 Reflection 技术对安全性的考虑,其运行效率要低于通过 JNI 调用本地的编译器动态链接库的实现方案。

参 考 文 献

[1] http://dev. perl. org/perl6/.

[2] Allison Randal, Dan Sugalski, Leopold Tötsch. Perl 6 and Parrot Essentials, O'Reilly & Associates, June 27, 2004.

[3] Bradley M Kuhn. perljvm: Using B to Facilitate a Perl Port To the Java Virtual Machine, Department of Electrical and Computer Engineering and Computer Science University of Cincinnati, Cincinnati, 2002.4.

[4] http://www. parrotcode. org/.