第31卷 第8期 计 算 机 工 程 2005年4月
Vol.31 №8 Computer Engineering April 2005
·软件技术与数据库· 文章编号：1000—3428(2005)08—0082—02 文献标识码：A 中图分类号：TP311.54;TP314

Perl后端到JVM的移植

徐 丽，张 昱，陈意云
(中国科学技术大学计算机科学技术系，合肥 230027)

摘 要：描述了Perl解释器的框架，介绍其后端虚拟机(Perl VM)的工作机理，给出了PVM所用到的栈、数据类型以及操作码函数(PP Code)在Java中的设计和实现，并陈述了在此移植中的几个关键问题及其解决方法。

关键词：Perl；Perl虚拟机；Java虚拟机

Backend of Perl and Its Porting to JVM

XU Li, ZHANG Yu, CHEN Yiyun
(Department of Computer Science, University of Science & Technology of China, Hefei 230027)

【Abstract】 In this paper, the framework of Perl interpreter is described, which is concentrated on the working mechanism of its backend(PVM). Then the design and implementation of PVM porting to JVM are specified , which involves the stacks, data types and operating functions (PP Code),etc. Moreover, some key problems of this porting and their solutions are stated.

【Key words】 Perl ; Perl virtual machine(PVM); Java virtual machine(JVM)

1 Perl解释器工作原理

Perl虽然是解释执行的语言，但是它的解释器perl并不是一个纯粹的解释器。它拥有编译器的前端以及后端：前者包括词法分析器和语法分析器，负责解析输入的Perl程序，产生IR——OP树，随后对OP树进行后序线索化以及窥孔优化；后者负责实现Perl的本地数据类型以及PVM，通过遍历OP树的执行路径，依次调用相应的操作代码完成对Perl程序的解释执行。

Perl的内建数据类型主要有3种：标量(SV)、数组(AV)、散列(HV)。此外，还引入CV、GV和IO分别表示代码值、符号表的Glob值以及IO值。这些数据类型是同构的，都有3个数据成员：sv_any, sv_refcnt(引用计数), sv_flags。它们的不同之处在于sv_any所指的数据结构不同。如若变量为字符串类型，则sv_any引用XPV结构来存储字符串；若变量表示一个IO句柄，就需要sv_any引用XPVIO结构,如图1所示。

```
Struct XPV
{char *    xpv_pv;
 STRLEN xpv_len;
 STRLEN xpv_cur;
}
```

(a)XPV数据结构定义

```
Struct XPVIO
{...
 PerlIO* xio_ifp;
 PerlIO* xio_ofp;
 ...}
```

(b) XPVIO数据结构

图1 XPV数据结构示例**

PVM是基于栈的，它使用7个栈来管理程序的执行，即保存栈、临时变量栈、作用域栈、参数栈、标记栈、上下文栈、返回栈。前3个用来实现变量和值的作用域，后3个管理子例程的调用、计算和循环。

OP树中操作码种类繁多且抽象级别较高，为此PVM用一组称为“PP(Push/Pop) Code”的函数实现这些操作码。

2 Perl后端的Java实现

在了解Perl的实现机制后，我们认为Perl后端的Java实现依赖于以下几点：(1) 内建数据类型及其操作的类设计；(2) 栈的设计；(3) PP Code的实现与整体管理架构。

2.1 内建数据类型的类设计

根据文献[4]对Perl内建数据类型的描述以及对Perl源代码的理解，目前设计并实现的内建数据类型结构如图2。类SV包含构成SV的3个数据域的定义，并且收集了Perl数据类型共有的方法及SV特有的方法。AV等类从SV继承(图2(a))，增加了一些自己特有的方法，如存储数组元素的函数av_store等。sv_any引用的数据结构按其数据域构成组织成相应的类体系(图2(b))。

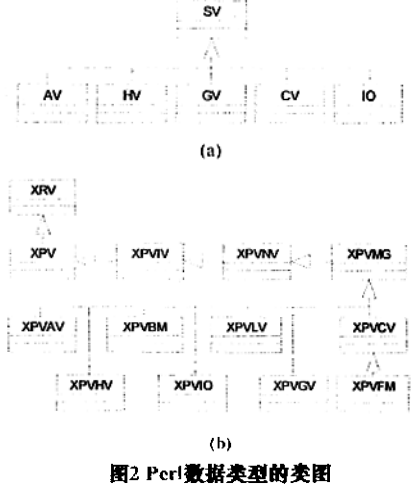

图2 Perl数据类型的类图

但是在测试中发现这种结构并不完全合理，这主要体现在SV、AV等的关系上。在C版本中，SV、AV等是结构等价的，C的强制类型转换机制使得一个变量无论最初是作为何种类型分配的，都可以随意地被强制转换成任何其它类型，完成一些相关的操作。而在Java中，上溯造型是允许的，但是downcasting则会导致异常。对于Perl来说，一个变量的类型在它的生命期是可以动态变化的。当新建一个变量时，一般按SV分配，它利用sv_flags保存当前实际存储的类型信息；当变量类型发生变化时，就会调用sv_upgrade( )进行相

基金项目：Intel中国研究中心基金资助项目
作者简介：徐 丽(1977－)，女，硕士生，主研方向为软件体系结构、程序设计语言理论和实现技术；张 昱，副教授；陈意云，博导、教授
定稿日期：2004-03-08 **E-mail**：lixu@mail.ustc.edu.cn

应的转换处理，这种转换不会改变变量的SV空间，但是可能会改变其sv_any指向的结点空间。尽管可以确定一些变量在初始时所具有的类型，对它们按实际的类型(如AV,HV)进行创建，但是由于Perl本身的类型体系，无法确定全部。这样，就导致了现有的数据类型的类结构在测试某些例子时，会出现ClassCastException。针对这一问题，目前尚没有一种更合理的解决方案。只能采取对象重构或设标志等一些临时性的解决方法。

2.2 栈的设计

PVM使用的栈虽称为栈，且其多数情况下的行为也符合栈的特性，但却并不能简单地使用JDK中提供的Stack类来实现它们。首先，每个栈都有自己的空间增长方式。其次，一些栈并不是只有pop、push操作就够了，还需要访问栈中的特定元素。再者，栈中元素的类型可能是不统一的，如保存栈中保存的可以是存储类型的整型编码、各种数据的值、存放数据的位置信息等。最后，需要从栈的语义出发提供一些抽象程度更高的方法，如保存栈可以提供savel32( )方法，支持对32位整数的现场保护等；还需要考虑各个栈之间的协作关系。

为此，设计一套Perl专用的栈类：PerlStack、PerlIntStack为除参数栈外其余6个栈的基类，这两个类的区别在于栈中的元素类型分别为Object和int；参数栈在C中是用AV来实现的，因为在某些时候需要将其看成一个AV，从而调用某些AV的函数，为此在参数栈类ArgumentStack中也封装了一个AV对象；类StackCotainer为各种栈实例的容器类，它负责管理这些栈。

需要特别说明的是保存栈的设计。为使得在退出某一作用域时，某些全局变量的值能得到正确地恢复，保存栈需要保存这些变量的地址。这在C中很容易用指针来实现，而在Java中却不行。为解决此问题，将全局变量分类保存在多个全局数组中，并给它们分配全局唯一的索引号以及通用的按全局索引存取的方法，这样用全局变量对应的全局索引替代地址进行现场保护和恢复。

2.3 PP Code 的设计

在C版本中，用全局的函数指针数组ppaddr保存各种操作码对应的函数指针；利用宏简化PP Code的书写，并保证代码执行的效率。这些函数的管理简单，但是可读性很差。

在Java实现中，定义PPFunc接口类来替代C中的pp函数指针类型，它包含一个抽象的方法pp( )。每个PPFunc实现类负责实现特定操作码的pp( )。另外，用类PPFuncMgr容纳各种操作码接口的实例，近500种的操作码接口保存在该类的一个PPFunc[]类型的数据成员中；PPFuncMgr负责初始化该数组，并提供特定操作码函数的访问。

在PP Code整体架构搭出来后，主要的困难就是这些函数的实现。由于Perl是由C发展而来的，其内置的、与系统操作有关的函数与C极其类似，因此用C来实现这些功能非常简单直接，但是用Java来实现这些功能却困难得多，如：

(1)I/O操作的实现

Perl内部用一个称作PerlIO的对象来处理所有的I/O请求。这一抽象层实质上是建立在两个功能库(即stdio和另一个速度要快得多的sfio)上的可移植层。Java的I/O类库虽然提供了非常丰富的类用于处理各种各样的I/O操作，但用其来实现Perl这种C风格化的IO操作时却很难。

1)Perl的多数I/O操作函数都是与C库函数相似的，因此在用C来实现时只需直接调用相应的C库函数即可。而Java对许多函数并不能直接支持，比如dup、fileno、flock、fstat等。

2)不管是文件，还是socket或者管道，Perl的C实现统一采用在符号表中保存相应的I/O *类型指针，I/O结构中的sv_any引用图1(b)中的XPVIO结构，xio_ifp指向所打开的文件或者socket连接的文件结构缓冲区，当需要调用某个I/O操作时，只需取得相应的文件指针即可。但是JDK中对各种I/O功能进行了细分，仅文件操作就提供了缓冲读取类BufferInputStream、管道类PipedInputStream、随机存储文件类RandomAccessFile等多种，并且JDK没有提供一个统一的纽带将它们联系起来。因此，如果要完全仿效C版本的这种实现方式，显然不行。

鉴于这些难点，采用Java本地方法(JNI)封装相应的C函数来实现Perl的I/O操作。其类结构如图3所示。

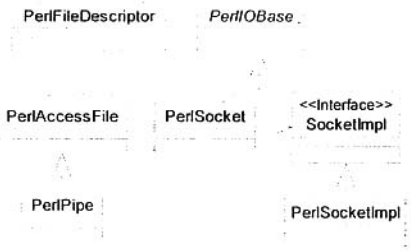

图3 I/O类的层次结构

PerlFileDescriptor是文件描述符类，该类有一个域存储文件描述符，由其标志一个打开的文件和socket连接，并提供针对文件描述符的相关操作。引入该类的目的在于Perl同时提供缓冲和非缓冲两种文件读写方式，非缓冲文件的读写方式以文件描述符为参数，尽管在符号表中只保存了文件结构缓冲区的指针，但是C中提供了fileno和fdopen函数在文件标号和文件结构缓冲区指针之间作转换。另一方面Perl中允许获取文件描述符的当前值，比如它提供了fileno标准函数。而在JDK中未提供这些功能。

1)PerlIOBase为抽象基类，它声明一些I/O公有操作的接口，如close等，并封装PerlFileDescriptor实例。

2)PerlAccessFile类封装了文件的各类操作，PerlPipe从它继承并实现了管道特有的一些操作。

3)PerlSocket、SocketImpl、PerlSocketImpl 3类实现了socket的相关操作，这里为了增强可扩展性，引入桥梁模式，socketImpl接口抽象了socket的操作，可以有不同的实现类。我们给出的实现类PerlSocketImpl是基于JNI的，完全封装了C的socket操作函数。

4)PerlAccessFile、PerlSocketImpl、PerlPipe 3类实现的方法均为native方法，与C的I/O库函数相似，由C实现。

这样用PerlIOBase可以取代C版本中的所有PerlIO类型声明，而具体的操作则由PerlIOBase的实现类完成。这简化了移植的工作量，也弥补了JDK功能上的不足。

(2)访问Unix shell

Perl最初是为Unix系统设计的，因此它具有很深的Unix烙印，如它提供了系统命令system、exec可以直接访问Unix shell。而Java设计之初是将其定位于可移植的、跨平台的语言，因此，它并没有提供强大的工具用于访问Unix shell。虽然利用Java提供的Runtime和Process对象，可以在Java应用程序中执行非Java程序，但是相对于C来实现要麻烦得多。

(3)goto语句的转换

PP code实现中的另一个困难是在C版本中频繁出现的各种宏、goto等如何在Java版本中转换。为了增强代码的可读性和可维护性，相当一部分的宏映射到Java中的方法。一些宏会使用其上下文中出现的全局或局部变量，将这些宏映射

(下转第150页)

种集成的并行软件开发环境，使得一群联网的异构计算机在用户程序看来是一台虚拟并行计算机，它集成了一组软件工具和库函数，能在由不同系统结构的计算机组成的网络上，模拟通用的可缩扩的并发计算机系统[5]。药品的销售预测作为医药物流系统的一个重要组成部分，对于影响销售的各种因素进行非线形分析，及时预知药品销售的趋势和走向，对于指导进货、促销策划、地域分布分析和客户分析等决策提供必要的支持，是BP网络在数据挖掘中的一个有实际价值和广阔前景的应用。网络输入层、隐含层、输出层的神经元个数分别为6-10-1。

本次训练所用样本为医药公司1998年9月到2003年6月间总共1 380条销售和进货数据。所有数据在训练前经过归一化函数 $Pn=(P-meanp)/stdp$ 的预处理。其中P和Pn分别为归一化前、后的数据，meanp为原始数据均值，stdp为方差。测得所用时间和加速比见表1。

表1 二次并行搜索BP算法在不同计算机数目中运行的加速比

epoch / time (s)	计算机结点数				
加速比	1	2	3	4	5
串行算法	5 789 / 2 011				
一般并行算法	5 789/2 011	5 789/1 059	5 789 / 755	5 789 / 598	5 789 / 488
	1	1.899	2.664	3.363	4.121
二次并行搜索算法	5 789/2 011	5 775/1 046	5 621/ 703	4 632 / 438	3 895 / 305
	1	1.923	2.861	4.591	6.593

可以看出，一般并行算法跟串行算法比起来，epoch数并没有减少，只是运算任务平均分配到了各从处理器，所以训练时间随着机器数的增加而降低。粗略看起来，随着机器节点数的增加，训练时间在不断减少，但是并不是呈线性下降的趋势，降低的比率在下降，按此趋势当机器节点数增加到一定数量的时候，训练时间便不会再下降，反而有可能上升，这是因为随着节点数增加，每个节点可以分配到的训练模式的比值就会降低，导致提高的计算时间的优势下降，通信开销反而增大到不容忽视。而采用二次搜索并行策略的算法在epoch和训练时间上都有所提高，说明其加速了网络的收敛。同时，随着机器节点数的增加，时间降低的比值并没有减少，说明了二次并行搜索算法除了减少运算时间外，有效地接近了最小极值区域，加快了收敛速度。

3 结语

采用BP网络进行销售趋势预测是人工神经网络技术在数据挖掘中的重要应用之一，但是由于用于此类应用的训练数据集一般比较庞大，传统的BP训练算法在训练速度和精度上无法满足实时要求。本文针对BP神经网络学习算法收敛速度慢、易陷入局部极值的缺陷，提出了一种新的BP算法的并行策略，详细给出了算法描述，并将其应用于一个大型的商业数据集，得到了令人满意的结果。此并行算法实现简单有效，能更好地应用于现实问题。

参考文献

1 袁曾任.人工神经元网络及其应用[M].北京:清华大学出版社,1999

2 Li Jun, Li Yuanxiang, Xu Jingwen, et al. Parallel Training Algorithm of BP Neural Networks[C].Proceedings of the 3rd World Congress on Intelligent Control and Automation, 2000,(2): 872 -876

3 Riedmiller M. Advanced Supervised Learning in Multi-layer Perceptrons-from Backpropagation to Adaptive Learning Algorithms [C] . Comput. Standards Interfaces 16, 1994,(5): 265-278

4 SchiJmann W, Joost M, Werner R. Comparison of Optimized Backpropagation Algorithms[C].Proceedings of the European Symposium on Arti-cial Neural Networks, ESANN '93, 1993: 97-104

5 高 曙.基于机群的并行BP算法的设计与实现[J].武汉理工大学学报(交通科学与工程版), 2002,26(5):589-591

6 刘 皓,魏 平,肖先赐.面向特定结构的几种BP并行算法及比较[J].系统工程与电子技术,2000,22(1):70-76

7 章锦文,马远良.神经网络算法的并行实现[J].计算机工程与设计,1994,16(4):16-21

(上接第83页)

成方法时，就需要将所引用的上下文变量作为方法的参数传递进去；如果宏操作改变了这些变量的值，就需要考虑如何将改变后的结果带回。有两种做法：一是通过返回值；二是将上下文变量封装到数组中，通过数组间接传递。

出于效率的考虑，Perl中的很多函数都很长，其中不乏使用goto进行流程的控制，Java中不支持任意的goto，这给代码的改写增加了困难。我们采取的做法是将代码按goto跳转进行程序段分块(类似于编译中的基本块的划分)，将每一块对应一个私有的方法，然后再写一个上层的方法调用这些私有的小方法。

3 测试结果与总结

在Perl的Java实现中，我们深切体会到C与Java两种语言各自的优劣，C的指针、宏、goto的合理使用会使得程序灵活、高效，但是也带来了程序难读性；Java的面向对象机制会便于代码的复用，它的强类型体制在一定程度上会保证代码的质量和安全性，不过Java程序的调试和字节码的执行的确难以让人恭维。我们同样也体会到Perl实用性的含义所在——完全、易用、高效。

我们在2003年初完成了Perl解释器的大部分实现，到目前为止测试和代码的修正工作仍在进行中，从已测试的情况看，一个短小的Perl脚本在C Perl上的编译、运行与它在Perl JVM上的编译、运行相比，后者要比前者多出百倍的时间开销。显然，要想真正使得Perl程序在JVM上运行，必须进一步优化Perl的Java实现，提高它的执行效率。

参考文献

1 Wall L, Christiansen T, Orwant J. Programming Perl(3rd Edition). O'Reilly & Associates, 2000-07

2 Kuhn B M. Considerations on Porting Perl to Java Virtual Machine [Masters Thesis]. Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, http://www.ebb.org/bkuhn/writings/technical/thesis.pdf,2001-01

3 Cozens S. Perl5 Internals. http://www.netthink.co.uk/downloads/internals.pdf, 2001

4 Aas G. PerlGuts Illustrated(Version 0.10). http://gisle.aas.no/perl/illguts/, 2001-06

5 Srinivasan S. Perlish译. 高级Perl编程. 北京:中国电力出版社, 2000