第 32 卷 第 3 期 计 算 机 工 程 2006 年 2 月
Vol.32 № 3 Computer Engineering February 2006
·软件技术与数据库· 文章编号：1000—3428(2006)03—0094—02 文献标识码：A 中图分类号：TP311.54

Perl 扩展到 JVM 的移植

张 磊，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：Perl 扩展作为 Perl 语言与其他语言的接口，使得 Perl 能够解决更加复杂的问题。该文从 Perl 扩展的创建入手，深入分析了 Perl 解释器的动态链接技术以及 Perl 扩展的工作原理。然后分析了 Perl 扩展到 Java 虚拟机移植的可行性和难点，最后给出了移植的几个实现方案及其评估。

关键词：Perl 扩展；移植；Java 虚拟机；动态链接

Porting of Perl Extension to JVM

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

【Abstract】 Perl extension is the interface between Perl and foreign languages. It can be used to handle more complicated problems. This paper introduces the creation of Perl extension at the beginning, and then it anatomizes the dynamic linking technology which Perl interpreter uses and explains how Perl extension works with it in detail. Subsequently, the feasibilities and difficulties of porting Perl to JVM are analyzed. In the end, this paper lays out several porting plans and gives the performance evaluation.

【Key words】 Perl extension; Porting; Java virtual machine (JVM); Dynamic linking

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 JVM 相关的硬件系统和软件系统展开。在 JVM 发展的同时，也需要考虑在此平台上软件的开发和移植问题。为了避免大量系统的重复开发，非 Java 语言到 JVM 的移植正在得到广泛的研究，这其中也包括 Perl 到 JVM 的移植研究[1]。作为 Perl 应用的强大后盾，Perl 扩展的移植将是 Perl 移植实用化中的一个重要内容。我们曾将 SPEC CINT2000[2] 中的 253.perlbmk 移植到 JVM，完成了 Perl 解释器核心部分的移植工作，本文将以 Win32 平台下 Perl5.005_03 对 C 语言的 XS 扩展为例，讨论在此基础上进行 Perl 扩展到 JVM 的移植方案。为方便起见，以下称 Perl 的 C 语言实现版本为 C_Perl，Java 语言实现版本为 J_Perl。

1 XS 扩展的实现机制

XS 语言是一种接口定义语言，用于创建 Perl 与 C 库之间的扩展接口。它可以包装一种叫做 XSUB（eXternal SUBroutine）的子例程，这些子例程可以封装 C 代码或外部的库函数。XS 语言的编译器是 xsubpp，它能够将 XS 源程序编译成 C 源程序。XS 扩展就是基于 XS 语言和 xsubpp 的，它至少需要解决以下几个关键问题：(1)提供可行的扩展装载方法，实现模块的引导和初始化；(2)能够进行数据类型的转换，实现 Perl 基本类型与其他语言基本类型的映射；(3)注意内存的管理方式，在数据跨越二者的接口时尤为重要；(4)兼顾对 Perl 脚本的影响，应当尽量地让 Perl 程序员感受到 Perl 的风格。

1.1 XS 扩展的创建

Perl 提供 h2xs 工具用于创建 XS 扩展框架[3]。首先编写 XS 文件，嵌入所需的 C 代码，通过 xsubpp 和 C 编译器最终会得到扩展的模块文件（.pm）和动态链接库文件（.dll）。当用户调用扩展中的 XSUB 时，模块文件就会自动搜索并加载对应的动态链接库文件，进而找到所需的函数。

1.2 XS 扩展的装载

当前很多平台都支持动态链接，采用动态链接技术可以避免每次更新 Perl 扩展时对 Perl 编译器的重新编译，从而方便高效地实现 Perl 扩展的装载和运行。

扩展的装载主要由 DynaLoader 模块实现，它是一个特殊的 XS 扩展模块，在编译 Perl 解释器时已经被静态加载，它负责在运行时动态加载所有其它的扩展模块，加载的一般过程是：经过一些必要的初始化和检验后，根据模块名找到待加载的动态链接库文件；然后在库文件中找到引导函数，完成对引导函数的合法性检查以后，执行它就可以把所有需要导出的子例程添加到 Perl 解释器的符号表中。扩展装载完毕后，用户就可以像访问一般模块中的子例程一样访问这些 XSUB。

1.3 XS 扩展的运行

Perl 解释器提供大量的 API 和全局变量用于操作变量、执行 Perl 代码和正则表达式、处理文件以及管理内存等。XS 扩展正是通过这些 API 和全局变量实现和解释器的交互。API 和全局变量由 perl.dll(Perl 解释器的核心部分)导出，Perl 扩展通过静态链接 perl.dll 获得对这些函数和变量的使用。

图 1 示意了通过动态链接实现 Perl 扩展的全部过程：主程序(perl.exe)在运行的初始阶段加载 perl.dll，perl.dll 中包括两个重要的部分：装载器和 API。装载器主要由 DynaLoader 模块实现，负责查找并动态装载所需的 Perl 扩展动态链接库；API 向 Perl 扩展提供函数接口，实现 Perl 解释器与 Perl 扩展的交互。另外，Perl 扩展还可以直接操作一些 Perl 解释器提

作者简介：张 磊(1981—)，男，硕士生，研究方向：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
收稿日期：2005-01-26 E-mail：yuzhang@ustc.edu.cn

供的全局变量。

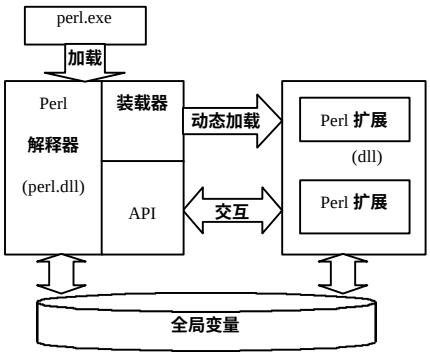

图 1 动态加载

2 Perl 扩展移植到 JVM 的关键问题

Perl 利用动态链接技术实现了 Perl 扩展的创建、装载和运行。在将 Perl 扩展移植到 JVM 时同样可参照这种机制。这时的主要问题是在 Java 环境中不能直接调用原有的扩展 dll，因为扩展 dll 的正常运行需要 C_Perl 提供的 Perl API 的支持，并且 J_Perl 和扩展 dll 中对应数据类型的存储结构显然是不一致的，扩展 dll 不能直接访问 Java 空间中的变量，反之亦然。所以在移植中有以下几个关键问题需要解决：

(1)如何动态加载 dll 文件。dll 文件的加载在 C_Perl 中是用 LoadLibraryEx()函数实现的，可以导入指定文件名的 dll 文件并返回该 dll 的 handler。但 Java 中没有类似的动态加载 dll 的方法，如何在 JVM 上实现动态链接库的加载有一定的难度。

(2)如何找到对应的过程入口。扩展被动态加载后，Perl 解释器会在符号表中保存扩展提供的函数名以及入口地址。从 dll 中找指定函数在 C_Perl 中是用 GetProcAddress()函数实现的，返回指定函数的入口地址。J_Perl 中同样需要考虑如何获取和保存 dll 文件中指定函数的入口。

(3)如何向扩展提供 Perl API。Perl API 是 Perl 扩展运行必不可少的一部分，它们在 C_Perl 中是由 perl.dll 提供的，J_Perl 也需要向扩展提供功能类似的接口，以便扩展 dll 在需要的时候调用。

(4)如何修改扩展文件。这些 XS 模块编译得到的 dll 文件需要 perl.dll 提供的 Perl API 和全局变量的支持，显然它们不能不加修改地被 JVM 载入，它们需要能够从 J_Perl 提供的接口中得到方法和变量。在不改变 XS 代码的前提下，唯一可行的方法是修改 XS 模块的编译器 xsubpp。这里需要有一个折中的考虑：既要顾及接口的方便高效，又要顾及编译器的实现难度。

3 移植方案

针对扩展移植的难点，可以把移植分为两部分考虑：(1)包括 Perl 解释器的 API 和 xsubpp 编译器，它们分别从 JVM 和动态链接库两个方面协调 Perl 解释器和 Perl 扩展的交互；(2)装载器，需要实现 Perl 解释器对动态链接库的查找及装载。

3.1 Perl API 和 xsubpp 编译器的移植方案

如图 2，在 J_Perl 中增加一个类 API，它封装一组方法，用于提供全部的 Perl API。另外，一个 XS 模块中可能会访问(读取或修改)很多 J_Perl 中定义的全局变量，Perl 扩展不能直接访问这些变量，这就需要 J_Perl 提供访问其全局变量的方法(get 和 set 方法)，这些方法也可以封装在类 API 中。另外在 J_Perl 和扩展 dll 之间增加一个 Glue 模块，Glue 模块负责提供全局变量的管理和所有 Perl API 的回调(callback)函数。

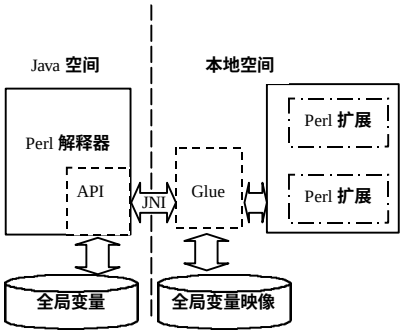

图 2 API 移植方案 2

它至少需要实现以下功能：

(1)为 J_Perl 提供一个调用扩展中 XSUB 的本地方法：由于所有的 XSUB 的输入参数和返回值均为空，因此本地方法只需要知道 XSUB 的入口地址就可以实现对 XSUB 的调用。

(2)为 Perl 扩展提供 J_Perl 中的全局变量：Glue 模块建立 J_Perl 全局变量在本地空间的映像，并提供对全局变量的管理函数，包括对全局变量的初始化（从 J_Perl 导入）和返回（更新回 J_Perl 中）。为了管理的方便，可以采用 struct 结构包含所有的全局变量。

(3)为 Perl 扩展提供 Perl API：这些 Perl API 实现部分由 J_Perl 的 API 类完成，Glue 模块只是回调 API 类中的方法而已。

Glue 模块可以用动态链接库实现，在解释器初始化时即被装载。它负责在每个 XSUB 开始之前将全部的全局变量从 J_Perl 载入，在 XSUB 结束后更新 J_Perl 中的全局变量。扩展 dll 只要通过 Glue 模块就可以得到所需的全局变量和接口函数，不需要做大的改动，只需在 C 代码开始重新定义一些宏即可。因为部分宏定义用于索引 C 结构体中的成员（如 #define SvIVX(sv) ((XPVIV*) SvANY(sv))->xiv_iv)，显然这些结构体在 Java 中的存储结构已经改变，所以这些宏必须被重新定义为函数(如#define SvIVX(sv) svivx(sv))，然后在函数中通过回调 Java 中的方法来得到对应的值。可以把这些宏定义集中在一个新的头文件中，再在每个扩展的开头添加这个头文件，这只需稍微修改一下编译器 xsubpp 即可。

3.2 动态加载的实现

既然装载器本身也是 XS 扩展，那么不妨将其当作一个普通的 XS 模块编译成 Perl 扩展 DynaLoader.dll，然后用 Glue 模块来加载 DynaLoader.dll。在 C_Perl 中 DynaLoader 模块是静态链入的，Glue 模块中不仅要负责 API 和全局变量的过渡工作，还需要一个单独的本地方法实现对 DynaLoader.dll 的调入过程，然后在每次 J_Perl 初始化时通过 Glue 模块就可以加载 DynaLoader.dll。而 Perl 解释器只需与 Glue 交互，由 Glue 模块实现对扩展 dll 的交互。

3.3 完整的移植方案

完整的移植方案如图 3 所示。

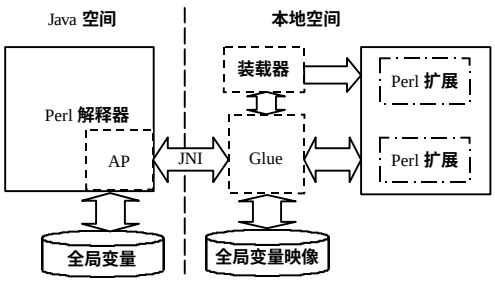

图 3 完整移植方案

(下转第 107 页)

是普通的 PC 机，使用 Windows 2000 操作系统；应用服务器使用的是 HP Unix 服务器，操作系统是 HP-UX 11.23；管理服务器集群是日本某大公司的大型机主机集群；存储设备是该日本公司的 raid1 磁盘阵列，光纤通道，光纤通道适配器（FCA Fibre Channel Adapter）是 HP Tachyon XL2 Fibre Channel Mass Storage Adapter。为了进行对比，在同样的环境下，我们对本地磁盘的文件存取也进行了性能测试。

由于进行 I/O 的单位是 Block，因此针对每个参数对不同的 block 大小进行了多次测试。对于共享磁盘打开文件、读写数据、关闭文件的操作速率均可达到 41MB/s，而且非常稳定。本地磁盘读速度较高，而写速度受到本地磁盘本身性能和 cache 的影响，值较小。结果如表 1 所示。

表 1　I/O 速度表

Blocksz	1024	2048	4096	16384	32739
本地磁盘 read 平均传输速率(MB/s)	62.69	62.58	62.63	62.86	62.55
本地磁盘 write 平均传输速率(MB/s)	6.60	6.54	6.59	6.56	6.53
共享磁盘 read 平均传输速率(MB/s)	41.90	42.23	42.06	42.33	42.45
共享磁盘 write 平均传输速率(MB/s)	41.60	41.49	41.66	41.63	41.36

对于不同大小的 Block，共享磁盘文件存取 CPU 的使用率（%SWCPU 和%SCPU）能够基本稳定地保持在 10%以下，运行 3h 以后会有所增加，但不会超过 16%，而且随着 Block size 的增大，内存使用率显著减小。对于本地磁盘文件存取，CPU 的使用率（%WCPU 和%CPU）能够稳定的保持在 5%以下。图 4 是 Block size 为 4kB 的 CPU 使用率对比曲线图。长时间、高负荷运行时，对于共享磁盘来说内存的使用量值会基本稳定在 150kB，随着时间的增长内存使用量会有所下降。

测试结果表明，本地磁盘的文件存取仍然具有一定的优势，但该 SAN 存储共享系统的性能已经接近于本地存储系统的存取性能。综合看来该系统具有良好的性能，其在日本已经得到肯定，正获得越来越多的用户的青睐。

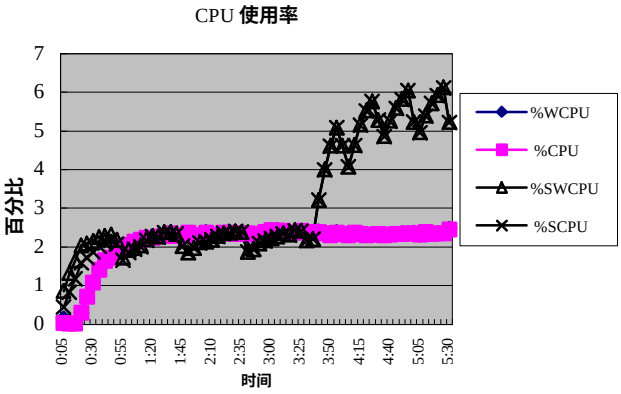

图 4　CPU 使用率曲线图

5 结束语

SAN 技术带来了全新的存储结构与存储理念，使存储技术跃上了新台阶。目前，国内外对 SAN 的研究十分活跃，许多专业从事数据存储解决方案开发的企业，如 IBM,EMC,HP 等，都把 SAN 作为未来网络存储的发展方向，已经开发出了数个成熟的企业存储管理系统。此 SAN 存储共享系统无需改变开放系统内核，可利用应用层的特性来优化磁盘 I/O 性能，易于实现、维护和移植，可实现异构数据共享、数据存储透明和高性能的数据读写等功能，具有广泛的应用前景，但它也存在着 SAN 系统的安全等问题，有待进一步的改善。

参考文献

1 IBM Corporation. IBM TotalStorage SAN File System Draft Protocol Specification[Z]. http://www-5.ibm.com/Storage/europe/uk/software/virtualization/sfs/protocol.pdf, 2003-04.

2 Molero X. A Tool for the Design and Evaluation of Fibre Channel Storage Area Networks[C]. 34th Annual Simulation Symposium (SS01), 2001.

3 Milanovic S, Petrovic Z. Building the Enterprise-wide Storage Networks[C]. Eurocon' 2001,Trends in Communications, International Conference, 2001.

4 杨　进, 魏轶伟. 存储区域网的性能测试[J]. 计算机工程, 2003, 29 (16): 43-44.

（上接第 95 页）

Glue 模块负责协调 Perl 解释器、装载器和 Perl 扩展。Glue 模块和装载器在 J_Perl 初始化阶段即被加载。当需要动态加载 Perl 扩展时，Perl 解释器通过 Glue 模块调用装载器的 XSUB；查找并安装指定的扩展 dll；执行其引导函数，将 Perl 扩展提供的 XSUB 信息导入到 Perl 解释器的符号表中。在调用 Perl 扩展中的 XSUB 时，Perl 解释器通过本地方法向 Glue 模块提供 XSUB 的入口地址。Glue 模块首先获取 Perl 解释器的全局变量，然后执行对应的 XSUB；执行过程中 Glue 模块需要协调 Perl 解释器和扩展之间的交互；XSUB 执行完毕后将全局变量更新回 Perl 解释器。

4 结束语

按照文中总结的扩展方案，我们已经实现了部分 Perl 扩展的移植，如 socket，SDBM_File 等。但要实现扩展的完全移植，还需要对 API 和全局变量进行扩充；并对 Glue 模块进行必要的优化，降低对内存的消耗，提高执行的效率。在移植中我们也注意到，Java 通过 JNI 提供对本地方法的处理接口非常丰富，包括数据类型的相互转换，本地方法对 Java 类中成员的访问以及对 Java 方法的回调等；但在对本地库的装载以及库函数的调用方面远不如 C 语言灵活方便，这主要是由于 Java 对运行时的安全性有更多的考虑。

参考文献

1 Kuhn B M. Perljvm: Using B to Facilitate a Perl Port to the Java Virtual Machine[C]. The Perl Conference 5.0. San Diego, CA, USA, 2001-07:68-72

2 Wall L, Christiansen T, Orwant J. Programming Perl[M]. O'Reilly & Associates, 2000-07