计算机科学2000Vol. 27№ 5

可移动 Agent 系统 JMSAS 中对象迁移的研究与实现*)

The Study and Implementation of Object Migration in the Mobile Software Agent System JMSAS

毕 凯 陈 翀 麦中凡

(北京航空航天大学计算机科学与工程系 北京100083)

Abstract Mobile Software Agent(MSA) model is a special kind of distributed object model. JMSAS is a system that we implemented to support MSA model. In the system, one of the key techniques is object migration, which is the capacity that Agent must have. As a consequence, Agent languages should be more special than other languages in some ways. Java is a good language for Agent. However, it can not satisfy the requirement of MSA model in some aspects. This paper analyzes the core mechanisms concerning object migration in Java platform, points out the problems to be solved, and presents our implementation of object migration in JMSAS.

Keywords Mobile software agent model/system, Object migration, Agent language, Java language

可移动软件 Agent(Mobile Software Agent, MSA)是一种新的分布式计算模型,与传统的客户/服务器模型相比,在 MSA 计算环境中,计算实体不再是静止和被动的,而是能够自主迁移计算的 Agent[1]。MSA 的计算方式就是将计算任务封装在 Agent 内部,通过 Agent 在多个位置迁移并与其他 Agent 交互和协作来完成这些任务。JMSAS 系统是一个基于 Java 语言的供可移动 Agent 运行的平台。实现该系统的关键任务之一是实现 Agent 对象的迁移机制。

本文首先分析了 MSA 模型对 Agent 语言提出的要求,指出为了使 Java 语言成为可移动 Agent 语言而必须解决的问题。然后,针对这些问题,分析了 Java 平台与对象迁移相关的核心技术并且提出了解决方案。最后,给出了在 JMSAS 中的实现。

1 可移动的 Agent 和 Java 语言

在 MSA 模型中,Agent 是一种可自主迁移的对象。对象的迁移是指对象由一个进程移动到另一个进程(迁移的源进程和目的进程可能位于网络中不同的机器),对象保持其原有状态不变并且可以继续运行。对象的迁移包括两个方面的内容:状态迁移和代码迁移。对象的状态就是对象的各个成员变量的值;对象的代码是指完全恢复对象所需的所有类的代码。

MSA 模型是一种特殊的分布式对象模型,与一般的分布式对象模型中的对象传送相比较,其特点在于迁移的自主性和极强的动态性[3]。由于高度的自主性,Agent 对象迁移的路线是不可预定的,不可能在网络中预先安装 Agent 对象所属的类来支持 Agent 的行为;MSA 系统对对象和类的管理范围不但包括本地系统,而且包括异地系统。由于这些特点,可移动 Agent 语言必须满足一定的要求:它必须具备代码标识和迁移能力、支持异质平台、较高的安全性等基本特性,以及期望具备强类型、自动内存管理、远地资源访问等特性[5]。Java 语言是适合于网络环境的程序设计语言,它所固有的平台无关性、并发性、安全性、动态类装入以及对象序列化等特性使得它在实现对象迁移方面有着天然的优势,同时,Java 语言具备强有力的计算能力,它的运行环境提供了足够灵活的可扩展接口,能够进一步开发针对 Agent 语言的运行系统,并且适合于面向应用的研究。因此,目前大多数的 MSA 系统都使用 Java 作为 Agent 语言。但是,作为传统程序设计语言的延续,Java 语言并不能直接满足 MSA 模型的特性,这突出地表现在以下两个方面:

1)代码标识问题。在一般的分布式对象系统中,对象的代码在本地编译,在本地运行,因此,代码的标识只要能够在本地区别开即可。例如,Agent 语言通过本机的文件系统来标识代码。在 MSA 模型中,Agent 对象的运行范围是整个网络,所以代码的标识应能够将代码在整个网络范围内区别,使用文件系统来标识类代码很容易引起重名和版本问题[6]。因此,在这一点

*)本文得到国家自然科学基金项目"可移动的软件 Agent 研究"资助,项目编号:69673003。毕 凯 硕士生,主要研究领域:可移动的软件 Agent,分布式对象系统。陈 翀 硕士生,主要研究领域:软件 Agent,可移动的软件 Agent。麦中凡 教授,主要研究领域:软件工程,程序设计语言,面向对象方法学,数据库,分布式智能软件系统等。

上，使用 Java 语言已有的机制不能满足要求，需要引入新的标识方法。

2)独立运行能力。移动式计算正在成为一种重要的计算模式。这种计算所采用的设备多为便携式计算设备，例如笔记本、掌上型电脑或个人数字助理(PDA)等，它们大多采用无线接入方式和固定网络连接，以支持移动用户在任何时候、任何地点访问网络中所需的资源。因此，移动式计算设备的网络线路具有带宽低、费用高、不可靠、有预见的断连性，以及计算依赖有限的能量源等问题。MSA 模型在移动式计算方面有着广阔的应用前景，通过可移动 Agent 在移动式设备和持久连接的网络之间迁移计算，可以免除维持线路持续的连接，所以它的对象迁移机制应该适应移动式计算的特点，要求 Agent 对象迁移到目的地后，可以在与原宿主机断连的情况下独自正确地运行[3-5]。是否具备独立运行能力与代码迁移机制密切相关。已有的代码迁移机制主要采用两种传送方式：全传送和按需传送。全传送方式是将对象恢复和运行所需的所有代码一次性地传送到目的地；按需传送方式则是根据对象创建、恢复或运行时的需要传送代码。Java 使用的是按需传送方式，在对象运行时可能需要从网络中传入所需的类，因此不具备单独运行能力。

因此，需要对 Java 平台进行改造。为了符合"100%纯 Java"标准，不应对 Java 语言本身进行修改，例如改动 Java 编译器或虚机。因此，我们完全利用 Java 语言已提供的机制和技术，通过对 Java 类库进行扩充来实现。

2 与对象迁移相关的 Java 平台技术

与对象迁移相关的 Java 技术包括：对象序列化技术(可以解决对象状态的传送问题)；类装入技术(可以解决代码传送问题)。

2.1 对象序列化(Serialization)技术

Java 的对象序列化技术实现了对象与字节流形式的相互转化。对象序列化包括两个相互反向的过程：序列化和反序列化[6]。序列化过程将对象转化成为字节流，它根据对象所属类的结构在字节流中记录对象的状态值(即成员变量的值)、对象所属类的名称、对象的成员变量所属类的名称以及其他相关信息。反序列化过程则是序列化的逆过程，它根据字节流中的信息恢复对象并还原其状态。实现序列化机制的类如表1所示。

2.2 类装入(Class Loading)技术

Java 虚机通过类装入器将类代码装入虚机[7,8]。类装入器按一定策略查找虚机所需的类代码，进行正确性和安全性检查，合格后装入虚机内部执行。每一个 Java 虚机都有一个缺省的系统类装入器，它只从本地装载 Java 的核心类和 CLASSPATH 环境变量所指定的路径下的类。一个虚机上可以有多个类装入器。应用程序可以创建自己的类装入器，实现自定义的类装入策略。例如，浏览器中的 Java 虚机使用的是 Java applet 的类装入器，它不但负责装入 Java 的核心类，也从相连的 WWW 服务器上下载类代码。

表1 与序列化机制相关的 Java 类

类名	功 能
java.io.ObjectOutputStream	对象序列化类。它的 writeObject 方法可以将对象写入某个字节流中。
java.io.ObjectInputStream	对象反序列化类。它的 readObject 方法从某个字节流中恢复对象。
java.io.ObjectStreamClass	提供字节流中各个对象所属类的信息，如使用它的 getName 方法可以得到流指针指向的对象所属类的名称。

类装入器由 java.lang.ClassLoader 类对象实现，其主要的方法如表2所示，其中 loadClass 方法是最核心的方法，由虚机直接调用，体现了该类装入器的类装入策略。继承 java.lang.ClassLoader 类，覆盖其 loadClass 方法，就可以实现不同的类查找和装入策略。

表2 ClassLoader 类的主要方法

方法定义	功 能
Class loadClass(String name, boolean resolve)	虚机调用该函数装入由参数 name 指定的类。
Class defineClass(String name, byte data[], int offset, int length)	对类代码进行安全性检查，如合法，将类代码转换成为 Class 对象。
Class findSystemClass(String name)	判断该类是否是 Java 系统的核心类。
void resolveClass(Class c)	如果 loadClass 方法的 resolve 参数为 true，loadClass 将调用该方法递归载入被当前的类所引用的类。

2.3 对象序列化和类装入之间的联系

在对象的反序列化过程中需要使用类装入机制，过程如图1所示。ObjectInputStream.readObject 方法从字节流中恢复对象时，会调用 ObjectInputStream.resolveClass方法。该方法的作用是使用类装入器，得

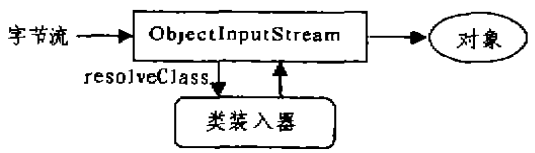

图1 对象序列化与类装入器之间的关系

到恢复对象所需的类。原有的 ObjectInputStream 类使用系统的类装入器。

3 MSA 系统对象迁移的实现

综上所述，对象序列化技术可以解决状态迁移，类装入技术可以解决代码迁移，实现对象迁移的思路就是将二者结合，同步完成对象的状态迁移和代码迁移这两个过程。但是在实现过程中，首先需要解决如下问题：

1)在序列化产生的字节流中只记录了对象及其直接或间接成员变量所属的类的名称。但是完全恢复对象，使之具备独立运行能力所需的类还包括对象方法的参数对象以及方法中的临时变量对象所直接或间接引用的类。因此，还应通过别的手段获得完整的类清单。

2)类装入器使用按需传送方式，这体现在 ClassLoader 类的 resolveClass 方法中，该方法只递归载入当时所需的类(按需传送)。

3)原有的类装入器只能装入 Java 的系统类和 CLASSPATH 环境变量所规定的路径下的类，没有从网络中查找并接收类的能力。

4)Java 平台通过文件名来标识类，这会引起类重名和版本冲突问题。因此，需要使用其它的标识方式。

其中前三个问题同独立运行能力有关，最后一个问题同代码标识问题有关。

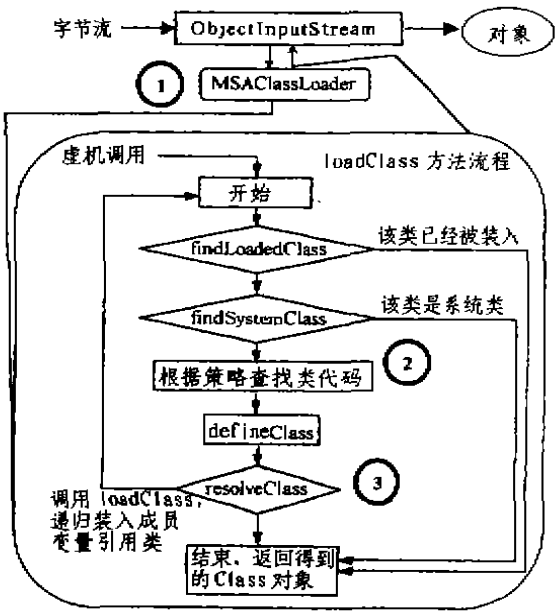

图2 独立运行能力的解决

3.1 独立运行能力的实现

利用对象序列化和类装入机制之间原有的联系，实现对象状态和代码的同步迁移，如图2所示。其中标号表示对原有 ObjectInputStream 和 ClassLoader 类所做的改动：

①MSAObjectInputStream 继承 java.io.ObjectInputStream 类。ObjectInputStream 类的 resolveClass 方法负责在对象恢复过程利用系统的类装入器得到所需的类代码，MSAObjectInputStream 类覆盖该方法，改为使用 MSAClassLoader

②java.lang ClassLoader 原有的类代码查找范围是系统类和由 CLASSPATH 环境变量指定的本地文件系统。在此处，MSAClassLoader 则根据一定的协议从网络中的代码源中查找所需的类代码。

③由对象序列化产生的字节流中只记录了完全恢复对象所需的部分类的名称，同时，Java 的类装入方式是按需装入，这体现在 ClassLoader 的 resolveClass 方法，它只递归装入当时运行所需的类。根据 Java 虚机代码规范[12]，类代码的常数池域(constant_pool)中存储了该类所引用的完整的类名清单，MSAClassLoader 的 loadClass 方法在此处对已得到的类代码进行分析，从而得到完整的类代码。

在 Java RMI 中有一种动态类装入技术(Dynamic Class Loading，RMI DCL)，也可以实现类代码的迁移，有的可移动 Agent 系统就是使用这种方式实现代码迁移的[10]。但是，我们在此没有选择这种方式。一方面，RMI DCL 只能基于 Internet 的标准协议，如 http 和 ftp，代码只能从 Web 服务器或 ftp 服务器中载入，而 MSA 系统有自己的传输协议和代码源；另一方面，RMI DCL 的类装入过程对程序员是透明的，我们不能控制类的装入方式(RMI 只使用按需传送，不能进行全传送)，并且不能对装入的类进行管理，因而不能满足系统的要求，其实，我们使用的方法和 RMI DCL 在思路上是一致的。

3.2 代码标识问题的解决

在代码标识问题上，有两种选择：以类为单位和以 Agent 为单位。因为 Java 的类装入器通过类的名称寻找类的代码，所以在这两种选择中，代码的标识都应以类的名称为基础，我们在此使用“代码初始注册地地址＋代码名＋版本号”作为标识，以类似于 ActiveX/DCOM 中的 GUID 来标识类代码是不可行的。

以类为单位，优点是能够区分分布在网络中的所有类，在 Agent 的恢复过程中，可以就近使用所需的类，不必全部从固定地点传送类代码，因而灵活性和效率较高。但是，在 Java 语言中，代码之间的引用并不携带版本信息，而这种版本标识方式可能会使引用代码

的版本号出现差异，当 Agent 运行需要某一代码时，由于缺少版本信息，类装入器将无法进行装入操作。

如果以 Agent 为单位，Agent 以"注册地 IP 地址＋类名－版本号"为标识，将 Agent 使用的所有类为整体进行注册。当本地有同名的类时，代码管理器为其分配一个新的版本号，被注册的所有代码都共享这个版本号和 IP 地址。由于在一次注册的所有代码中，代码名是唯一的，因此这种方式可以实现代码标识的唯一性。实现这种方式的关键在于 MSAClassLoader 对象的配合，每一个 Agent 对象都对应一个 MSAClassLoader 类装入器，MSAClassLoader 对象中保存有该 Agent 对象的"注册地 IP 地址"和"版本号"信息。当系统运行 Agent 需要装入某个类时，还是以该类的类名为标识向对应的 MSAClassLoader 提出请求；而 MSAClassLoader 在本地及网络中查找该类时，则用自己保存的"注册地 IP 地址＋版本号"信息连同该类的类名组成标识，从而得到所需的类，实现正确装入。这样，既维持了 Java 语言原有的类请求方式（以类名作为请求），又实现了在整个网络环境中对类的正确定位。

3.3 JMSAS 中对象迁移部分的系统结构及工作流程

JMSAS 系统中的对象迁移子系统结构如图3所示。在网络通信部件中实现了一套通信协议，Agent 对象及其代码的传送都由该部件完成。代码迁移部件包括代码提供部件、代码服务器和对象恢复部件。

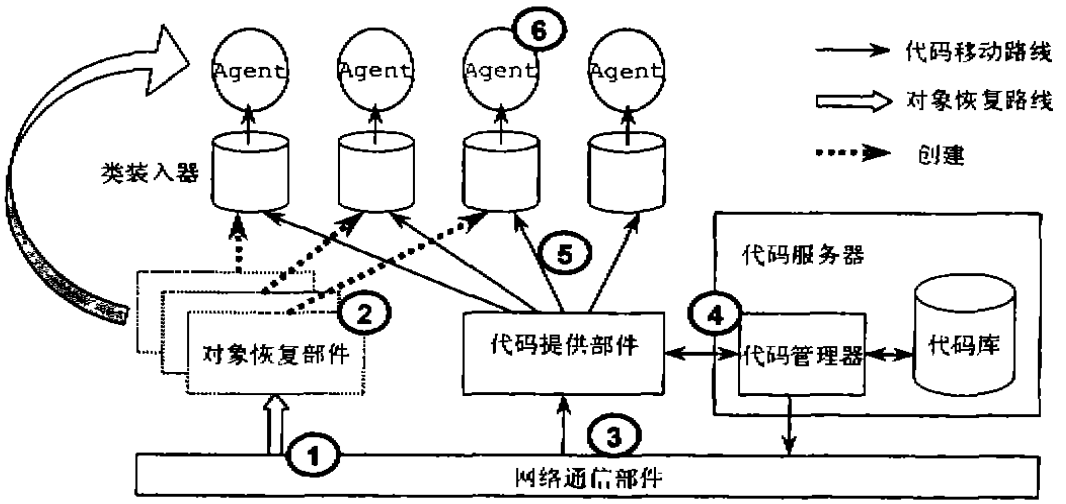

图3 JMSA 系统代码迁移部件结构

代码提供部件负责向本系统提供 Agent 创建、恢复和运行所需的代码，以及根据系统的代码迁移策略在网络中确定代码源。代码提供部件可以根据需要实现全传送或者按需传送方式。

代码服务器由代码管理器和代码库两部分组成，它可以作为一个独立的部件存在。代码库既作为本地注册的代码的存储地，同时也作为系统的代码缓存存储外部代码；代码管理器提供了访问以及维护代码库的接口，包括 Agent 类代码的注册，查询并获取某个代码，删除代码以及代码缓存的置换等。除了 Java 语言和 MSAS 系统的核心类，MSAS 系统不能使用任何文件系统中的类，而只能使用代码库中的类。为了实现类代码的正确标识，Agent 使用的类都必须在代码库中进行注册。

对象恢复部件就是 MSAObjectInputStream 类对象，当网络通信部件接收到对象向本地迁移的请求后，就创建一个 MSAObjectInputStream 类对象，负责恢复整个对象。

在 Agent 对象的迁移过程中，首先由对象的发送方向接收方发送对象迁移请求。接收方的网络通信部件得到请求后，接收该对象的状态信息（即对象序列化所产生的字节流）（标号①）。之后网络通信部件创建一个对象恢复部件，来完成后续步骤（标号②）。对象恢复部件根据该 Agent 对象的注册地和版本号，创建一个 MSAClassLoader 类装入器，进行反序列化，在反序列化过程中，类装入器向代码提供部件提出类请求，代码提供部件首先在本地的代码服务器中查找，如果本地已存在该类，则返回类代码，如果没有，代码提供部件则使用通信协议向代码源申请所需的类代码（标号③）。

（下转第49页）

分:聚焦维度(如客户);为聚焦而计算的维度(如利润);内部属性(如客户年龄等);外部属性(如某客户最喜爱的产品颜色);非聚焦维度(如某客户平均一次购买的商品的数目)。在执行影响性分析时,焦点沿着不同的基本维度(或类)旋转。

影响域模型的旋转模式不能用于数据仓库,它仅仅用于数据挖掘,因为旋转模式适合于影响空间,但不适合数据空间。可对数据仓库采用稍稍改动的星型模式,对数据挖掘采用旋转模式。目前,基于影响域的模型还没有相应的产品出现。

结论 通过以上分析,可以得出下面的结论:OLAM是OLAP与数据挖掘相结合的产物,它兼有OLAP多维分析的在线性、灵活性和数据挖掘对数据处理的深入性,是数据库(数据仓库)应用工具未来发展的方向。目前,这个领域中的研究工作尚处于起步阶段,还有很多问题需要得到解决,包括技术问题和非技术问题,这不仅给广大研究工作者带来挑战,同时也带来了机遇。

主要参考文献

1 Codd E F, et al. Beyond decision support. Computerworld, 1993, 27(30)

2 Fayyad U M, et al. Advances in knowledge discovery and data mining. California: AAAI/MIT Press, 1996

3 Han J W. Towards on-line analytical mining in large databases. ACM SIGMOD Record, 1998, 27: 97～107

4 Han J W, et al. Issues for On-Line Analytical Mining of Data Warehouses. In: Proc. of 1998 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98), Seattle, Washington, June 1998

5 Han J W. OLAP Mining: An Integration of OLAP with Data Mining. In: Proc. 1997 IFIP Conf. on Data Semantics (DS-7), Leysin, Switzerland, Oct. 1997. 1～11

6 Han J W. Conference Tutorial Notes: Integration of Data Mining and Data Warehousing Technologies. In: 1997 Int'l Conf. on Data Engineering (ICDE'97), Birmingham, England, April 1997

7 Han J W, et al. DBMiner: A System for Data Mining in Relational Databases and Data Warehouses. In: Proc. CASCON'97: Meeting of Minds, Toronto, Canada, Nov. 1997

8 Parsaye K. Surveying decision supprot. Database Programming and Design, 1996, 9(4): 27～33

9 Parsaye K. OLAP & Data Mining: Bridging the Gap. Database Programming and Design, 1997, 10(2): 30～37

10 Parsaye K. Data Mines for Data Warehouses. Database Programming and Design, Sept. 1996

(上接第38页)

代码提供部件得到类代码后,首先将其存储在本地的代码服务器中,以备再次使用,然后将类代码返回给类装入器(标号⑤)。在得到所需的类代码后,对象恢复部件就可以恢复对象,提交给系统(标号⑥),之后对象恢复部件被撤消。

总结 实现Agent对象迁移的总的思路是使Java成为适合于可移动软件Agent的编程语言,主要是解决独立运行能力和类在全局范围内的标识问题。在对Java语言的序列化、类装入、RMI技术,以及类代码格式进行深入的研究后,我们将序列化和类装入技术结合,实现了Agent对象的状态和类代码的同步传送,从而在JMSA系统中使Java编写的Agent对象具备了迁移能力。

参考文献

1 陈 羽中,麦中凡. 可移动的软件Agent研究. 计算机科学, 1998(5)

2 陈 羽中,毕 凯,麦中凡. 可移动软件Agent系统代码迁移机制的研究. 1998

3 陈 羽中. 可移动软件Agent系统的研究、设计与实现:[北京航空航天大学硕士生毕业论文]. 1999

4 Gray R, Kotz D. Mobile agents for mobile computing. [Technical Report]. Department of Computer Science Dartmouth College, 1996

5 Knabe F C. Language Support for Mobile Agnets: [PhD thesis]. School of Computer Science, Carnegie Mellon University, 1995

6 Picco G P, et al. Expressing Code Mobility in Mobile UNITY: [Technical Report]. Washington University, 1997

7 McManus C. The basic of Java class loaders. Available at: http// www javaworld. com, October 1996

8 Krumel A. Revolutionary RMI: Dynamic class loading and behavior objects. Available at: http: // www. javaworld. com, December 1998

9 Sun Microsystems Inc. Object Serialization Specification. 1997

10 Sun Microsystems Inc. Java Remote Method Invocation Specification. Revision 1. 50, JDK 1. 2, October 1998

11 Sun Microsystems Inc. Java Platform 1. 2 API Specification, Version 1. 2. 1998

12 Lindholm T, Yellin F. The Java Virtual Machine Specification, Version 2. Sun Microsystems Inc. 1999