文章编号：1005-8451（2004）02-0004-03

基于Java的Mobile Agent结构研究

黄 健，罗四维，张 玮
（北京交通大学 计算机与信息技术学院，北京 100044）

摘 要：与传统的分布式计算相比较，Mobile Agent模型无疑具有更广泛的发展前景。它采用面向对象的思想，使Agent具有了类的一切特性。Java语言由于其对网络和多线程支持还有与平台无关的特性使得它是一种适合开发Mobile Agent的语言。以Java为基础，提出了一种开发Mobile Agent的方案，并对相应的关键技术问题给予说明，讨论了系统的可扩展性和安全性，从而为实现Mobile Agent的开发开辟了新的途径。

关键词：Mobile Agent；Java；可扩展性
中图法分类号：U29-39 文献标识码：A

Structure of Mobile Agent based on Java

HUANG Jian, LUO Si-wei, ZHANG Wei
(School of Computer Science &Information Technology of Beijing Jiaotong University, Beijing 100044, China)

Abstract: Mobile Agent was a more advanced model compared with the traditionally distributed computing because it used the object-oriented technique to enable the Agent to have all characteristics that were owned by a class. The Java, strongly supporting the programming of network and multi-threads, was a language independent of PlatForm and a proper tool to develop Mobile Agent. It built a new way for Mobile Agent's development and explained the key problems related to the scalability and security of the system, which could be some new ways for Mobile Agent's implementation.

Key words: Mobile Agent; Java; scalability

伴随着网络技术的发展，有价值的信息资源在不断的增长。对于如何捆绑分布异构环境中信息源的问题变得越来越突出。进入90年代以来，使这种现象得以改观的一种新技术—Mobile Agent正悄然兴起。Mobile Agent被认为是具有自主性、反映性、异步性和智能性的软件实体。它可以自主地在异构网络上按照一定的规程迁移，寻找合适的资源，完成用户给定的任务。作者主要介绍了一个基于Java的Mobile Agent结构及其关键技术，并说明了它的扩展性和安全性。

1 Java及其相关技术

1.1 Socket

Java中java.net包提供了与通讯和网络资源操作有关的类。Socket接口提供访问标准网络协议的方法。TCP/IP提供对Socket的支持：通过数据流传送在不同主机上的对象。Java通过提供一个简单化的，面向对象的接口给Socket，使得其应用更加简单，通过网络从一个Socket中读和写与标准的I/O操作一样简单。在Mobile Agent中的Agent间的消息发送和远程的命令调用都应用到了Socket技术。

1.2 多线程

多任务应用需要同时运行多个进程，因此可被划分为多个线程，相应的程序称为多线程程序。Java语言的重要特征之一就是在语言级支持多线程的程序设计。Mobile Agent中在启动Agent的时候，为了安全的因素，应该启用新的线程。

1.3 Java对象的动态迁移

Java程序是由若干类组成，运行时对类实例化，即构成对象。对象是一个动态的概念，有起点、终点与运行状态，对象中的成员变量即反映了对象的状态。如果把一个对象的运行状态记录下来，再传送到另一台机器上，恢复该对象的状态，则可让该对象接着前面的状态继续运行，这就实现了对象的动态迁移。在Mobile Agent中把Agent当作一个对象，对Agent的移动就变成了对象的移动。

收稿日期：2003-08-15
作者简介：黄 健，在读硕士研究生；罗四维，教授。

2　整体结构

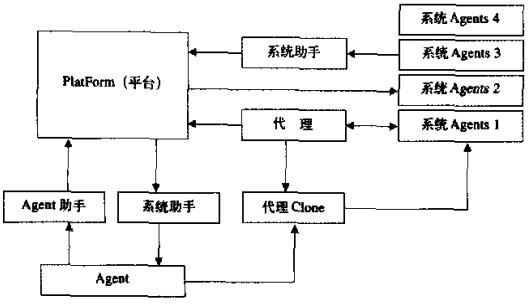

图1　Agent 系统的整体结构

系统的整体结构如图1所示。

系统主要由PlatForm、Agent和系统Agent（System Agent）组成的。PlatForm是Agent和System Agent的平台，它提供了一些使得这两部分可以对系统进行访问的方法，主要包括Agent和System Agent的启动控制，状态的获取，系统的输入输出，系统资源的维护，系统的关闭等。

System Agent主要目的是为了实现Mobile Agent的定位，移动和发送消息等功能，它被分成几个部分，每个部分实现部分相对独立的功能。它通过注册到自身的系统助手（SysHelper）和平台（PlatForm）进行通讯，调用PlatForm上的函数，同时PlatForm可以直接对System Agent进行访问。

Agent是用户编程的主要对象，用户在里面加入想要实现的方法和功能即可，Agent对PlatForm的访问是通过AgentHelper来进行，相应的PlatForm通过Agent装载器（AgentLoader）来对它进行控制，Agent为了实现移动，发送消息等通讯功能需要调用一些System Agent的功能，这种调用是通过对System Agent代理的Clone上的方法调用来实现，这样的调用出于安全性考虑。

3　Mobile Agent 主要功能和实现

3.1　MAFFinder

MAFFinder是Mobile Agent的主要部分，它主要实现了对于注册到PlatForm的Agent进行定位，以方便用户对Agent位置查询和在移动的时候选择路径。它还包括对于System Agent注册的检查，来确保每个System Agent只能运行一个实例。有这样一个System Agent，是考虑到和OMG组织已经开展有关Mobile Agent的标准化工作，并提出的Mobile Agent互操作机制标准MASIF[1]。MASIF标准的所有通讯是用CORBA实现的,这里的MAFFinder的设计基本上是与通讯协议无关的。

3.2　通讯服务

通讯服务为处理所有的通讯设计，需要通过网络通讯的有Agents和System Agents，System Agents中真正需要通讯的是Communication Services，它们使用自身的程序和MAFFinder完成通讯。系统总的通讯结构见图2。

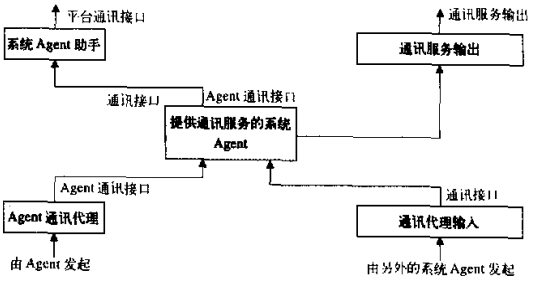

图2　Agent 通讯的结构

Agent通过得到一个代理（AgentCommProxy）返回一个可以应用于通讯的接口（AgentCommIFace），通过此接口可以和作为系统Agent的Communication Service（提供通讯服务）进行通讯，并调用上面的方法。在AgentCommIFace接口中定义了Agent和Communication Service通讯的方法，而且这个接口由Communication Service来实现；同样的，CommunicationIFace中定义了PlatForm和Communication Service通讯的方法，并在Communication Service中实现。这样Communication Service会继续把Agent的请求传递到具体的PlatForm上。所以，系统中主要的类是Communication Service本身，它维持着所有其它的部分并且根据协议使用的不同，在同一个类或不同的两个类之间提供通过代理来调用方法，用于输入和输出。

Agent移动是Mobile Agent的主要特征，根据是否迁移执行状态，Agent迁移可分强迁移(Strong Migration)和弱迁移(Weak Migration)。强迁移又称透明迁移(Transparent Migration)，当Agent申请迁移时，Agent当前的执行状态和数据状态被封装并连同代码传送到目标位置，一旦到达目的地，Agent恢复并从移动语句后的指令开始执行。弱迁移又称非透明迁移(non-Ransparent Migration)，它仅迁移数据状态和代码，当Agent迁移到目的地后，并不是从迁移点之后

开始执行，而是从可执行代码的入口开始执行。由于Java语言是解释性语言，自身的JVM不能自动完成强迁移，但是可以通过修改JVM[2]和在Java代码中做标识来实现强迁移[3]。本系统中的Agent并没有采用完全强迁移的方法，它采用了Java中流技术对要传送的对象进行操作，将得到的当前状态的对象通过网络直接发送到目的地。

在move命令被调用后，对Agent的状态进行一些设定，并停止这个Agent的工作，返回这个Agent对象，用于在网络上传递。然后通过通讯服务实现对远程系统接受Agent（receiveAgent）的调用，远程的系统完成对这个Agent对象的接收，启动和注册。最后是调用本地的传送命令（transferOK），删除这个Agent在这个系统中的所有注册信息，从而终止了Agent。

3.3 Admin（管理服务）

Admin提供了用户和系统的交互的手段，用户可以通过Admin来启动、停止，终止一个Agent或System Agent，查看所有的Agent和System Agent的状态，查看系统中所有的运行线程和系统中提供服务的代理情况等。

4 系统的可扩展性

用户需要对Agent进行二次开发来完成所需功能，良好的可扩展性为用户提供快捷的开发模式。

用户可以通过对Agent类的继承来实现Agent的基础部分。如：public class AgentEx1 extends Agent 。而且这个类还提供了一些方法用来被用户覆盖，这样用户可以主要去关心自己程序实现的功能。如initUser() 这个方法在初始化Agent的时候调用。这是在Agent生命最开始做的事情。这里应该放入新的Agent的初始化的代码。类似的方法还有：startUser(), suspendUser(), terminateUser(), 它们分别在启动、停止和终止一个Agent时被调用。用户自己的方法同样写在类中，如果使用变量，要使用全局变量，否则，变量信息在Agent移动过程中不会被保存到对象中。

5 系统安全性考虑

5.1 代理的使用

系统中广泛定义了接口（interfaces），并在接口中说明对于系统中提供的公有化资源访问的定义。对每一个Agent都提供一个对象（Object），在这个对象中，接口中定义了什么，它就只能包含什么。这个对象仅能实现在PlatForm中登记的接口的方法，它就像在用户（Agent）和提供者（System Agent）之间的一个代理(Proxy)。System Agent仍将执行所有的关于资源的访问的方法，但是会产生并注册一个Proxy而不由PlatForm自身来完成。这个代理实现和System Agent有一样的接口，而且内含System Agent提供的服务参数。所有的对Proxy的方法调用包括了对于System Agent的调用。但是一些方法调用需要了解是哪个Agent调用了它，这需要对给定的Agent的proxy个性化，因此对于每个Agent的调用，Proxy都需要产生一个具有Agent身份标识的克隆（clone）。

5.2 安全性管理

本系统中提供了AgentSecurityManager，它是由Java中的安全性管理SecurityManager继承来的，在Java中这个类定义管理Socket可否打开，文件可否访问，线程可否创建等[4]。这样，用户完全可以根据自己的需要和所运行环境的安全性的要求来自己改写这个类。

6 结束语

本文对一个简单的Mobile Agent的总体结构，内部的实现机制，可扩展性和安全性作了介绍，用户很容易在这个结构的基础上构建自己的Mobile Agent框架，或者直接对本系统进行扩展来实现自己的功能。用户通过对本系统的扩展，如消息的群发机制，网络上MAFFinder的具体管理，将它用于实现网格的计算和应用。

参考文献：

[1] Dale, J. A mobile agent architecture for distributed information management[D]. Southampton: University of Southampton.1977.

[2] 田敬军. 移动Agent系统-IBM Aglets初探 [J]. 唐山师范学院学报，2002，(3).

[3] 王 红，曹广周，林守勋. 基于Java的Agent强迁移 [J]. 小型微型计算机系统，2002，(3).

[4] Ken Arnold, James Gosling. Java程序设计语言 [M]. 杨承高. 北京：北京大学出版社,1998.