# 基于Java的Mobile Agent原型系统的研究与开发

北京工业大学电子工程与自动化系　赵瑞彬　周　浩　秦世引　徐宁寿
中国科学院计算机网络信息中心数据库应用研究室　肖　云

Mobile Agent(MA)**具有可移动性、协作性、异步性等优点，其中可移动性为其他诸多特性提供底层通信支持，其成功实现是开发一个MA系统必须考虑的关键技术之一。文中以** Jama (Java mobile agent) **系统的开发为实践依据提出了一种基于Java语言实现MA原型系统的具体方案，重点介绍了可移动性和** MAF Finder (Mobile Agent Facility Finder) **的具体实现步骤。**

## 引　言

Agent 概念起源于70年代对分布式人工智能(DAI)的研究[1]。它代表独立的、交互式的、相互协作的对象实体，主要功能在于多个Agents之间的相互通信和协作以完成分布式问题求解(DPS)。而Mobile Agent (MA)不仅能具有自主性和协作性，它还能根据问题求解以及自身运行状态的要求在网络环境下在一台主机上中止运行，然后移动到另一台主机上继续运行。MA 这种基于 RP (Remote Programming) 的通信方式相对于传统的以RPC(Remote Procedure Calling)为基础的网络架构具有节省网络带宽、开放性和异步性等明显的优势[2]。因此，MA技术在智能信息获取、电子商务、工业过程控制等方面有广泛的应用前景。

虽然 MA 技术进入大规模商业应用的时机尚不成熟，但目前众多知名公司和研究机构在该领域的积极探索为这一天的尽早到来奠定了技术基础。国内对MA技术的研究也于近些年蓬勃开展了起来，但有关MA系统的具体技术实现的文献还不多见，作者结合自身研发体验写作此文，希望能对国内在此方面的研究有所推动。

## 开发 MA 系统所涉及的关键技术

随着用户数量呈指数级增长，目前Internet的架构以及Web技术无法满足两种相互矛盾的需求：日益复杂的通信类型和有限的带宽。虽然乐观地讲，MA可以在一定程度上解决Web应用中所出现的这类复杂问题，并且可以成为一种进一步发展Web应用行之有效的技术途径和有力的支持工具，甚至成为软件工业继面向对象技术之后下一个解决软件复杂性的技术生长点。但是目前在研究和开发 MA 的过程中也面临着许多技术难题，只有充分地加以分析解决才可能为MA进入实用阶段奠定基础。

(1) 自主移动。MA的自主移动包括三个层次的要求：首先要实现代码、数据和状态三者的传输，以及到达目标主机后的运行状态的恢复；其次，对某些异常的连接进行智能化协调处理，例如当MA在目标主机完成数据处理欲回到源主机时，源主机已经断开网络连接，此时 MA 进入等待状态直到连接恢复；最后是 MA 移动中的智能路由处理，即可根据实际任务需要和现行网络状况确定访问主机的列表和顺序。

(2) 通信模型[3]。在复杂、异构的网络环境下，对于一项任务通常必须由多个 Agents（既有 Stationary Agents，又有 Mobile Agents）相互协作共同完成，而多个Agents的协作依赖于完善的通信模型提供底层支持。在一个MA系统中，Agents之间的消息传递和同步处理是建立通信模型的核心要求，同时也是实现Agents知识层通信的基础。对于知识层上的通信，由于其实现的复杂性和效率等方面的原因，目前大多数MA系统尚未考虑该部分。知识层通信的基础是KQML（Knowledge Query Manipulation Language）和 FIPA（Foundation for

Intelligent Physical Agents）的 ACL（Agent Communication Language）。

(3) 管理和控制[3,4]。对于运行中的 MA 系统进行管理和控制就是利用现有的技术对Agents 实行监控和管理甚至修改 Agents 的任务指令。此外，对于 Agents之间的任务协调和管理也在该范围之内。

(4)安全性和可靠性。由于 MA是处于一种动态、复杂的网络环境下，其移动路线也是动态变化的，因此保证 MA 安全和可靠地运行是 MA系统得以大规模推广应用的必要条件。

在MA 系统中，安全性主要包括以下四个方面：

① MA 的安全传输；

②保护主机免受 MA 的滥用或破坏；

③保护 MA 免受其他 Agents 的攻击；

④保护 MA 免遭主机的破坏。

目前，大多数 MA 系统都实现了前三者，对于最后一项尚未有可行的方案。

MA 的可靠性主要包括：

① MA 传输中的可靠性；

②MA在宿主主机上的可靠性，即在宿主系统崩溃的情况下，能否恢复；

③ MA 对于动态网络环境的应变能力，如MA 的排队和转发。

在上述四种关键技术中，MA 的自主移动的成功实现是开发一个 MA 系统的基本要求。

## 基于 Java 实现 MA 原型系统的一种设计方案

Java语言的平台无关性、内置多线程、方便的TCP/IP Socket接口，以及对大多数数据库类型的支持，使得它一方面可以在网络环境下获得良好的运行支持，另一方面又具备了对主流应用的强大支持。Java 的这些优异特性使它成为开发一个 MA 系统的最佳选择。

Jama（Java mobile agent）是一个基于 Java 语言开发而成的 MA 原型系统，该系统具备了 MA 的可移动性、异步性并可以对 MA 进行监控管理。根据作者开发 Jama原型系统的体验，重点介绍在Jama系统中MA 的自主移动以及对 MA 进行监控管理的实现方案。

由于在 JDK1.2 中集成了 Swing、Java2D 等方便 GUI 编程的接口，尤其在安全性方面较 JDK1.1 有较大改进，因此 Jama 的开发工具选择了JDK1.2。在介绍具体实现方案之前，首先简要回顾所用到的 JDK1.2 中的接口。

(1) RMI (Remote Method Invocation)。RMI是指用Java 编写的程序中在不同主机上的对象可以相互调用对方的方法。一个 RMI 应用包含两部分，即 Client和 Server。在Server端提供一个远端对象（Remote Object，如果一个对象中包含有可以被远程调用的方法，那么这个对象即称为远端对象），这个远端对象的引用（reference）可以被 Client访问，Client 端通过它来完成对远端对象中方法的调用。

图 1 中描述了 Server 首先调用 registry 把远端对象与一个名字联系（bind）起来，然后 Client 端在 registry 中通过查找这个名字（look up）来获得远端对象的引用（reference）。成功完成上述两步之后，Client 端就可以通过远端对象的引用调用它的方法（invoke）。另外，图中的 Web Server 在需要时用来在 Server和 Client 之间进行动态类代码的下载。

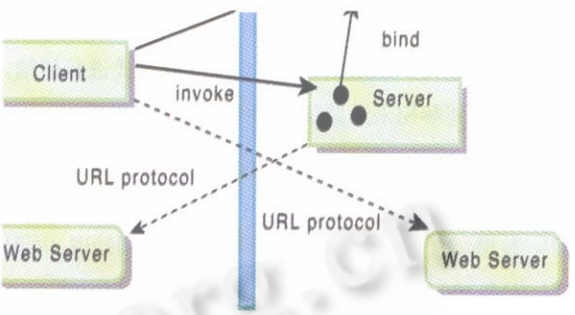


图 1 RMI 的执行过程和相互调用关系

(2) Classloader。在运行独立 Java 程序时要指定 main( ) 函数所在的类代码名称。但程序一旦运行起来，根据运行要求，装载的类代码是由 ClassLoader 动态完成的。图2描述了ClassLoader 在运行时导入字节码文件并交给执行引擎的流程关系。

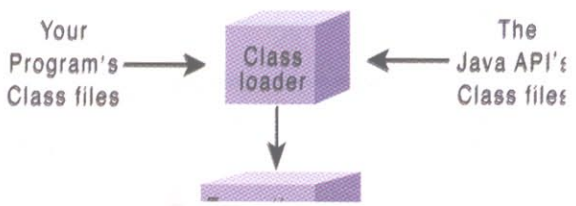


图 2 Java 虚拟机(JVM)的 ClassLoader 机制

在 Java 程序中，有两种 ClassLoader：primordial

class loader和class loader objects。Primordial class loader是JVM的一部分，负责从本地文件系统装载Java API的类文件，每个JVM中只有一个primordial class loader；class loader objects象普通Java程序一样在运行时被装载、初始化，因此从更大程度上它是Java应用程序的一部分，每个Java应用可以有多个class loader objects。利用class loader objects，Java应用可以根据需要从网络环境下载类代码。

由于ClassLoader具有类代码动态下载的强大功能，使用不当有可能引发安全性问题。ClassLoader本身从两方面提供了安全保护措施：不同的类用不同的命名空间下载；保护Java核心API不受非法访问。

(3)Object Serialization。Object Serialization即对象序列化是指系统能将对象解码为一个字节流，同时也能从字节流中将对象图（Object Graph）重构。序列化可用于对象持续化（Object Persistence），还可使对象通过Sockets进行传输。在缺省情况下，对象中static和transient型的属性是不被序列化的。这一特性可以用来实现对象序列化过程中对重要信息的保密，比如可以把需保密的信息限定为transient，而把它的加密形式进行传输，在接收端对其解秘来还原信息。

Jama系统中利用RMI、ClassLoader和Object Serialization实现了agent的自主移动的前两个层次，即代码、数据、状态的传输和对异常连接的协调处理。同时利用RMI初步实现了对Agents的监控和管理。

(1) 系统结构

如图3所示，Jama系统主要包括agentpool package、launcher package、maf(mobile agent facility) package和openwin package。

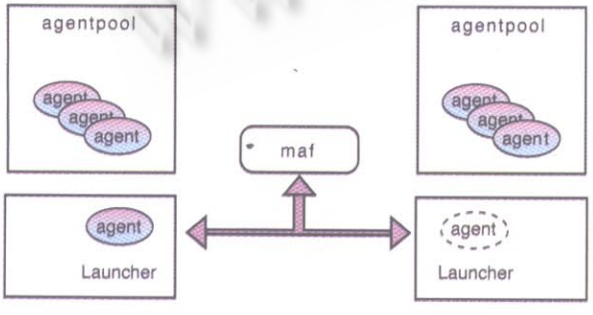


图 3 Jama原型系统的结构

Agentpool package中存放各种Agents的类定义，用户也可以根据实际应用的要求通过执行Jama提供的接口来定制Agents。Launcher package中的Launcher class实际上是以RMI Server和RMI Client双重身份的形式出现。首先作为RMI Server，它提供了receiveAgent( )方法被RMI Client调用以接收Agents；其次作为RMI Client，它可以调用RMI Server中的receiveAgent( )方法以发送Agent。Launcher class通过对象序列化与Agent通信以便接收Agent的状态。Maf package提供了对整个系统中各Agents进行追踪监控的Finder class。Finder通过作为RMI的Server端提供了可以被远程调用的registerAgent( )方法和unregisterAgent()方法来实现对各Agents的注册和注销。Openwin package提供了便于人机交互的GUI。

在Jama中的每个Agent是一个单独的类，具有独立的执行线程。它所要完成的任务由自身的方法和属性决定，在任何时刻某个Agent的状态由它的类实例唯一决定。

(2) 系统的工作流程。如图4所示，系统的初始化是构造Launcher和Finder的类实例。在此基础上系统的一个完整的工作周期分为三个步骤：

① Creation。Creation的完成首先由Launcher从agentpool中初始化一个Agent的对象traveller，而后Launcher通过调用Finder的registerAgent( )方法（如图4中的registerAgent( )）把Agent的Name、Location等基本信息注册到Finder的对象。

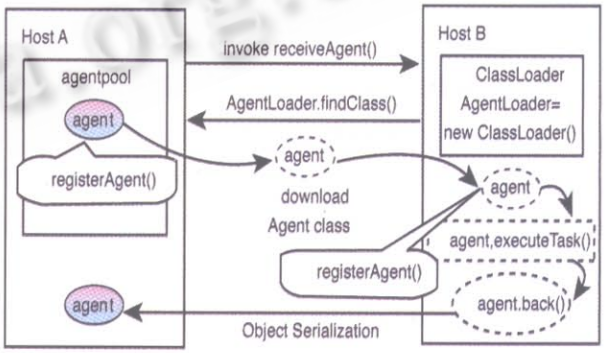


图 4 Jama原型系统的工作流程图

② Dispatch。Dispatch事件的产生可以是应使用者的要求，也可以是Agent本身运行需要或对其他Agent的消息响应。

首先，由Launcher作为RMI Client调远端对象的receiveAgent( )方法（invoke receiveAgent( )）。在

receiveAgent( )方法定义中，创建 ClassLoader 的对象 AgentLoader（ClassLoader AgentLoader=new ClassLoader），由 AgentLoader 执行 findClass( ) 方法（AgentLoader.findClass( )）从指定的 URL 搜索路径下载 Agent 类的代码（download Agent class）。当代码被成功下载后，由相应的接口把代码恢复为类定义并初始化为类实例。当 Agent 到达远端主机后，再次调用 Finder 的registerAgent( )方法以更新注册信息(registerAgent)。在此之后 Agent 对象就可以根据所携带的状态（State）自主决定所进行的操作（agent. executeTask( )），比如说启动本身的执行线程完成数据库的智能查询等。

③ Back。Back 事件的产生也可以是对多种消息的响应，它由 Agent的对象调用自身方法（agent.back( )）来完成。

具体的实现是：Agent 对象与源主机上 Launcher 的对象进行通信，通过对象序列化（Object Serializa-tion）传输本身的状态，当源主机完全接收了Agent的状态并重构了对象图之后，Agent 就成功返回了源主机。如果 Agent 与源主机的通信过程中出现网络连接异常，Agent 即转入等待状态同时监听网络连接，一旦连接恢复，Agent 重新调用 back()方法以实现返回。

以上只说明了 Agent 从源主机出发访问一台目标主机的过程，实际上在Jama系统中Agent可以根据需要在方法 executeTask()的定义中安排多台主机的访问路由。

(3) Jama 的安全策略。在涉及访问系统资源的敏感操作时 Java API 采用 Security Manager 对其进行安全检查。Jama 完全采用了 JDK1.2 API，因此具备了灵活的基于 policy 的安全体系。 在本例中直接采用了 RMISecurityManager，由于RMISecuriyManager 提供了较严格的类似于对 Applet 的安全限制，在应用中可以根据实际需要在 policy 文件中增加安全许可。例如在 Jama 原型系统中涉及到了构造 ClassLoader 对象、更改线程组、建立套接字（Socket）的操作，于是在 poilcy 文件中增加了如下许可:

```
grant
permission java.lang.RuntimePermission "create ClassLoader";
permission java.lang.RuntimePermission "modify - Thread";
permission java.lang.RuntimePermission "modify-Thread-Group";
permission java.net.SocketPermission "*", "connect, accept, resolve";
};
```

对涉及数据库查询的 MA 应用，还应增加:

permission java.util. PropertyPermission "file. encoding", "read"; ■

**参考文献**

(1) Software agents: an overview, HYACINTH S. NWANA, The Knowledge Engineering Review, Vol.11:3, 1996, 205-244

(2) Mobile Agents, William R.Cockayne and Michael Zyda, Manning Publications Co., 1997

(3) Joachim Baumann, Fritz Hohl, Nikolaos Radouniklis, 'Communication Concepts for Mobile Agent System', First International Workshop MA'97

(4) Mole3.0, www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html