.NET环境下的移动代理迁移机制*)

王 丹 于 戈 王国仁
(东北大学信息学院 沈阳110006)

Migration Mechanism for the Mobile Agent Based on .Net

WANG Dan YU Ge WANG Guo-Ren
(College of Information, Dongbei University, Shenyang 110006)

Abstract The migration mechanism is one of the most important techniques for the mobile agent techniques. In this paper, migration mechanism for the mobile agent based on Microsoft .NET developing environment is researched. First, the two serialized techniques provided by .NET platform are analyzed and compared, and Binary serialization techniques are adopted to implement the data state migration of the mobile agent. Then, the Web service approach is introduced to realize the code state migration of the mobile agent and the corresponding Web service is constructed. At last, the whole migration process of the mobile agent is described.

Keywords Mobile agent, Migration, .Net; Web service, Serialization

1 引言

移动代理(mobile agent)的迁移是指移动代理在运行过程中,从一个网络节点转移到另一个节点上继续执行的过程,是一种使代理当前的运行状态在分布式系统的另一个节点上恢复并持续运行的机制[1]。移动代理的状态由三部分组成:代码态、数据态和执行态[2]。其中代码态是指实现移动代理的程序任务代码,完成代理的功能并控制其行为;数据态指代理对象的全局变量和代理的属性(成员变量),它描述了代理本身的相关信息,如代理标识符、所有者、起始节点的地址以及迁移路径等,用户可以根据自己的需要进行定义;执行态则指代理执行过程中的上下文信息,如与调用顺序有关的堆栈、PC计数器等系统信息。很多文献根据代理的执行状态是否迁移,将移动代理的迁移分为强迁移和弱迁移两种,本文将考虑弱迁移,即如何实现移动代理在异构节点之间的迁移时的数据态和代码态的捕获和恢复问题[2]。

许多现有的系统都采用Java语言作为开发移动代理系统的语言[3,4]。采用Java的RMI机制或者CORBA的ORB实现分布式计算,由于Java语言的平台无关性,基于Java语言提供的序列化和类装载机制实现代理的迁移是一般系统常采用的方法。在这种机制下,要求通信的两端必须有同类结构,如同为Java代码或同一个公司的ORB产品。随着Internet的普及,分布于Web上的用户相互间的联系变得很松散,无法保证通信的另一端是否具备所需的基本结构,而且对象通常无法通过Web上的常用协议(如HTTP,SMTP)实现远程调用,难以在Web上提供完整的服务。

.NET是为简化在因特网的分布式环境下的应用程序的开发,基于开放互联网标准和协议之上,实现异质语言和平台高度交互性而构建的新一代计算和通信平台[5]。.NET平台支持标准的Internet协议和相关标准包括HTTP、HTML、XML、SOAP[6]以及其它网络标准,尤其支持了无状态、松散耦合的网络服务[7]。.NET主要基于包含实际语言和执行平台的.NET Framework,以及提供了丰富内建功能的基本类库,其中.NET Framework的核心思想就是通过XML和SOAP提供了一个新型的Internet软件集成开发环境。它包括了一个虚拟机CLR(公共语言运行时)。该虚拟机提出了一个直接进行多语言集成的新型体系结构,并提供了比以往任何语言类库都丰富的基础类库,是一个跨多种语言的编程环境,其中一个关键技术就是微软中间语言MSIL,所有语言经过编译后都成为MSIL形式,使得不同语言开发的应用能自动集成,从而实现跨语言调用。C#是开发.NET框架应用程序的最好语言,它是整个.NET平台的基础。采用C#将很容易开发出可以跨语言集成的网络应用。

Web服务是松散耦合的,可复用的软件模块,在Internet上发布后能够通过标准的Internet协议在程序中访问。Web服务是为了能够在Internet上不同的操作系统、硬件平台和编程语言之间集成应用软件,它的开发和使用是独立于Web上各种各样的操作系统、编程模型和语言[6]。

.NET环境下的代理迁移问题是开发基于.NET应用的重要内容,虽然Java语言是目前常用的移动代理系统的开发语言,而由于.NET目前并不支持Java语言,而.NET Framework提供的虚拟机CLR使得开发移动代理系统成为可能,并扩充了移动代理的应用范围。本文讨论了.NET平台下移动代理的迁移技术,分析了.NET提供的序列化机制,描述了基于.NET的二进制序列化机制实现代理数据态迁移的方法和过程,并利用Web服务的概念,构造了类代码的上传、下载等Web服务实现代理的代码态迁移,并给出了代理迁移的过程描述。

2 .NET的序列化机制

序列化是将对象的状态转化为可保持或传输的格式如字节流形式,并存储到存储媒介中。对象序列化一方面可以在以后重新创建对象的精确副本,另一方面通过值将对象从一个应用程序域发送到另一个应用程序域中。序列化机制分为序列化和反序列化两个过程:

(1)序列化:指根据对象所属类的结构在字节流中记录对象的成员变量的值、对象所属类的名称、对象的成员变量所属类的名称以及其它相关信息,从而使对象转化成字节流的形

*)基金项目:国家863/CIMS主题(2001AA415210)。王 丹 博士,主要研究方向为移动代理技术与分布式计算;于 戈 博士,教授,博士导师,主要研究方向为数据库,数据挖掘,工作流,Web Service。

式，以便于通过网络进行传输。

(2)反序列化：则是序列化的逆过程，它根据序列化后的字节流中的信息恢复对象并还原其状态，它是把流恢复成对象的过程。

序列化和反序列化提供了对象状态的存储和还原的能力。当需要跨越多个应用程序发送该类时，或者需要将该类在远程进行处理，序列化机制就非常有用。

2.1 序列化

支持对象序列化最简单的办法就是将整个类都标志为可序列化的，这要在类的定义上加上标志"Serializable"，并通过下面介绍的格式器 Formatter 将该类的实例的信息写入流中，也可以从流中构造该类的实例。

但如果类的某些数据成员包含敏感信息，如因为安全原因某些信息不能够通过网络传输，或者序列化之后的信息是无用的，如假设一个类将线程 ID 存储在成员变量中，反序列化该类时，在序列化该类时为其存储 ID 的线程可能不再运行，因此序列化该值没有意义，则可以将类本身标志为可序列化的，而将包含敏感性信息的成员变量单独标志为 NonSerialized，来防止这些成员变量被序列化。还有一种方法是实现 Iserializable 接口并仅序列化必需的字段。

另外，Serializable 属性不能被继承，如果类 A 从类 B 派生，B 为可序列化的，那么只有将 A 标志为 Serializable 才能序列化。下面是一个简单的例子。

```
[Serializable]
public class MyObject
{
public int n1;
[NonSerialized]public int n2;
public String str;
}
```

2.2 格式器与流

序列化对象时，可以控制对象的信息如何编码到流中，这可以通过.NET 的格式对象(Formatter)完成。格式器提供将序列化对象格式化的功能，.NET 提供了一个抽象的 Formatter 类，该类提供了将对象写到流和从流中读取对象的基本接口，同时还定义了该类的两个派生类 BinaryFormatter 类和 SoapFormatter 类，它们分别提供了以二进制和 XML 格式编码对象信息和读取对象的方法。

对象序列化后的信息是保存到流中的，流的本质是一系列的二进制数据。.NET 提供了一个抽象类 Stream，它定义了操作二进制数据的基本接口。.NET 基础类库提供了以下五个派生于 Stream 的类：BufferedStream、FileStream、MemoryStream、NetworkStream 和 CryptoStream。

格式器本质上是实现了 IFormatter 接口的类型的实例。此接口提供了以下两个公共方法：

(1)Serialize：将对象或具有给定根的对象图序列化为所提供的流，Serialize 方法会自动将所提供的对象以及与它连接的所有对象序列化为所提供的流，默认情况下，序列化过程将通过收集对象的所有字段的值来记录对象的状态，这些状态与有关对象的信息(如其类型的程序集)一起保存到流中；

(2)DeSerialize：反序列化所提供流中的数据并重新组成对象实例。

2.3 .NET 的序列化机制

.NET 提供了两种对象序列化机制[5]：

(1)Binary 序列化：它保持了对象类型的完整性，对于一个应用的不同存储方式之间保持对象的状态是非常有效的。

(2)XML 序列化：仅序列化公共的属性和字段，并不能保持类型的完整性。如果想提供和使用一些不受应用限制的数据时，它是非常有效的。

由于 XML 序列化不能转变方法、索引、私有字段和只读属性，因此为了序列化一个对象的所有字段和属性，包括公共的和私有的，只能采用 Binary 对象序列化机制。

下面以 Binary 序列化为例，给出序列化和反序列化的简单情况：

```
IFormatter formatter = new BinaryFormatter();
    //生成 BinaryFormatter 格式器
Stream stream = new FileStream("SrcMAfile", FileMode.Create,
                FileAccess.Write, FileShare.None);
                //生成流对象
formatter.Serialize(stream, obj); //序列化
//….
Stream stream = new FileStream("SrcMAfile", FileMode.Open,
    FileAccess.Read, FileShare.Read); //打开流
MyObject obj2=(MyObject)formatter.Deserialize(stream);
    //反序列化，从流中重构对象
//….
```

上面的代码使用了 Binary 格式器，同样也可以用 XML 格式器来实现序列化，这时只需要将上面代码中修改为：

```
//IFormatter formatter = new BinaryFormatter();
IFormatter formatter = new SoapFormatter();
```

.NET 还提供了一种自定义格式器。一般情况下，使用.NET 框架提供的格式器就够用了。如果想实现自定义的格式器，可以通过实现接口 IFormatter 或者从已有的格式器派生实现，如果只需要对现有格式器的某些行为进行小幅度的调整，就可以通过重载父的虚方法实现[5]。

3. Web 服务与代理迁移

Web 服务主要建立在三个角色的交互上，它们是：服务的提供者、服务的注册处和服务的请求者，而交互的内容包括发布、查找和绑定三个操作[7,8]。Web 服务提供者拥有可以通过网络访问的 Web 服务的实现模块，它为这个 Web 服务定义服务说明，并把它发布给服务的请求者或服务的注册处；服务的请求者使用查找操作从本地或服务的注册处得到服务说明，并使用服务说明中的信息与服务的提供者实现绑定，调用其中的操作。服务说明是一个使用 WSDL[10]表示的 XML 文档，定义了服务交互的接口和机制。UDDI[6]规范定义了一种发布和发现 Web 服务相关信息的标准方法。服务提供者将服务说明通过 UDDI 注册中心进行注册，需要查询需要此服务的用户可以检索 UDDI 注册中心，得到此服务的服务说明，并使用服务说明中的信息与服务的提供者实现绑定，与之交互并调用其提供的功能。服务的注册与发现过程如图1所示。

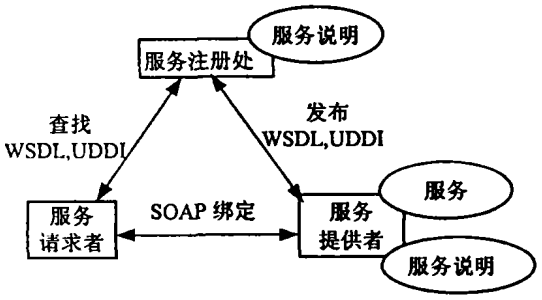


图1 Web 服务的角色和操作

在 Web 服务中，使用 XML 表达各种信息及这些信息的类型。由于各种系统或平台都有自己的类型系统，而 Web 服务的优势之一就是能够跨平台、跨系统，因此 Web 服务采用的是 XML 大纲规范的第二部分所定义的 XSD 类型系统。

使用 Web 服务的过程实际上是 Web 服务的使用者与 Web 服务实现绑定，并调用其方法的过程。绑定有两种方式，一种是静态的，静态是通过已经获得 WSDL 文档获得；另一种是动态的，通过 UDDI 实现的。服务请求者一方通过发 SOAP 请求，基于 UDDI 和 WSDL 定位到所需的服务，服务执行后将产生的结果返回给客户端。

4. 代理的数据态和代码态迁移

4.1 基于 Binary 序列化机制的数据态迁移

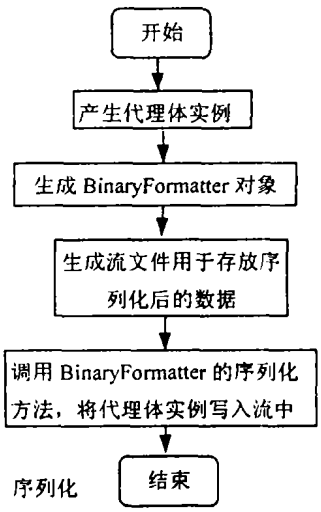
由上面的介绍可知，.NET 的 Binary 序列化可以保持对象类型的完整性，对于移动代理这种需要跨越不同网络节点执行而且需要保持对象的状态的应用来讲是非常有效的。基于 Binary 序列化机制的移动代理的数据态迁移过程如图2所示。其中 BinaryFormatter 对象和流文件用于完成数据态的迁移；并使用 C# 说明代理从节点 S1迁移到节点 S2，并在 S2恢复执行的过程。

·代理对象在节点 S1被序列化的过程：

```
myMobileAgent myagent = new myMobileAgent()
    ;//产生代理体实例，myMobileAgent 是代理类，用户自己定义；
Formatter formatter = new BinaryFormatter ( );//生成 BinaryFormatter 对象
Stream temp = new FileStream ( " SrcMAfile", FileMode. Create, FileAccess. Write, FileShare. None);//在节点 S1产生 temp 文件流
formatter. Serialize(temp, myagent);//调用序列化方法，将代理数据态序列化到 temp 流中。
temp. Close();
```

·代理对象在节点 S2被反序列化：

```
IFormatter formatter = new BinaryFormatter ( );//生成 BinaryFormatter 对象
Uri WebSite = new Uri ( " http://SrcSiteIP/CodePath/SrcMAfile");//定位源节点 S1上的文件流
HttpWebRequest myRequest = ( HttpWebRequest ) WebRequest. Create(WebSite);
HttpWebResponse myResponse = (HttpWebResponse) myRequest. GetResponse();
Stream temp = myResponse. GetResponseStream();//定位网络数据流
myMobileAgent myagent = (myMobileAgent) formatter. Deserialize (temp);
//反序列化方法，用于从 temp 流中恢复代理对象的实例。
temp. Close();
```



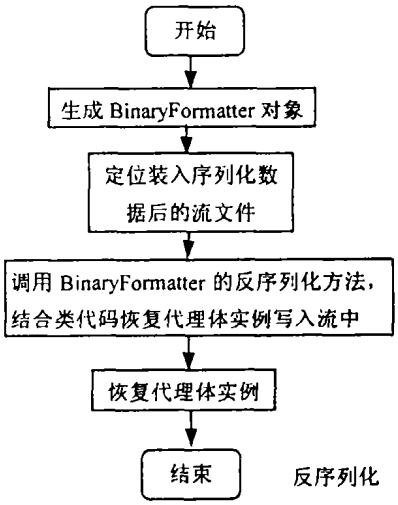


图2 基于.Net 的 Binary 序列化的代理数据态迁移

4.2 基于 Web 服务的代码态迁移

只有目的节点存在恢复代理所需的代码，才能恢复代理实例。我们采用特定的网络类代码存储服务器来存储代理的类代码，并为每一个代理体分配全局唯一的标识，当代理到达的新节点上没有恢复代理所用的类代码时，可以通过网络，从指定的网络类代码存储服务器上实现“按需下载”。由于类代码的上传与下载的过程都是可复用的软件模块，因此，我们将代理的类代码的上传和下载等操作构造为 Web 服务，包括：

·类代码的上传：代理迁移之前将代理的类代码上传到指定的网络类代码存储服务器上，用以在其它节点准备恢复代理时下载使用；

·类代码的下载：当代理迁移到另一个节点中，其节点上的移动代理服务器为恢复代理的运行，可以调用这个 Web 服务，从指定的网络类代码存储服务器上来下载代码，实现代理体的恢复。(由于篇幅所限，我们不详细描述系统的实现，移动代理服务器指支持代理迁移、通信等一个执行环境，位于代理运行的各个节点上[3,4])。

在实现时，我们采用如图3所示的模型，实现 Web 服务的调用：

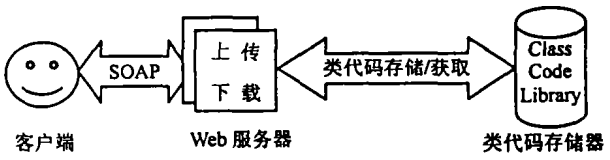


图3 Web 服务应用模型

(1)客户端，这里的客户端有两层含义：当客户端是提供代理体的类代码者，如代理体的所有者，当第一次将代理体发送网络中的第一个节点，通过调用 Web 服务的上传方法向网络类代码存储服务器上传代理类代码；当客户端是移动代理服务器，它需要使用代理体的代码恢复代理的运行时，则需要调用代码下载方法从网络类代码存储服务器上下载类代码。

(2)Web 服务器，是提供相关 Web 服务的 Web 服务器，负责服务的事务处理逻辑。它的任务是接收客户端的请求，如果客户端需要代码上传操作，将用户的代理类代码上传到与 Web 服务器相连接的类代码存储服务器上，并分配给类代码的物理存储路径相对应的代理标识，将此标识返回给用户；如果客户端需要代码下载操作，则 Web 服务将从类代码存储服务器上根据所要下载的类代码标识查找所需的代理类代码，

并传输给调用下载服务的客户端。

(3)类代码的存储器,提供物理存储空间用来存储代理的类代码,为 Web 服务使用。

基于这种 Web 服务的思想实现代码的上传和下载,可以节省网络带宽,使系统更能适应用户多变的服务要求,更具有灵活性和可扩展性。

4.3 Web 服务的实现及服务调用

在实现服务绑定时,我们采用静态绑定方式:通过静态的事先已经获得的 WSDL 文件来实现。在.NET 中提供了一个用于创建 Web 服务代理类的命令行工具 wsdl.exe[10,11],其中 Web 服务代理类指的是根据 Web 服务的 WSDL 文件产生的本地类,它包括类和方法的声明。因此,客户端使用代理类中的信息是通过底层 SOAP 协议来访问 Web 服务,实现方法的调用。

类代码迁移的 Web 服务中提供了两种方法:PutCode 和 GetCode,完成代码的上传和下载。

(1)上传操作 当客户端准备将一个特定的代理体发送到网络中时,须先绑定此 Web 服务,调用类代码上传方法 PutCode,将类代码上传到特定的网络类代码存储服务器上,其输入参数为用户存放类代码的本地虚拟路径。PutCode 方法根据此路径下载类代码并存放到网络类代码存储服务器上,并返回给用户一个与类代码相对应的全局唯一的代理标识,此标识将对应于存放类代码的网络类代码存储服务器路径,类代码的代理标识与存放类代码的类代码存储服务器路径的对应关系将在数据库中存储。主要代码描述如下:

```
[WebMethod] public int PutCode(String VirtualPath){
    System.NET.WebClient Client=new WebClient();
    Client.DownloadFile ( " http://" + VirtualPath ,
        ServerAgentClassPath); //上传操作
    int myAgentID =AllocateAgentID(ServerAgentClassPath);
            //分配代理标识 myAgentID 与代理类路径相对应
    return myAgentID;//返回代理标识}
```

(2)下载操作 当在迁移后的节点上准备恢复代理时,移动代理服务器调用相应的 Web 服务 GetCode 方法,在指定的网络类代码存储服务器上下载相应的代理类代码,用于实现代理对象的恢复。其输入参数的第一个为所需类代码的代理标识 AgentID,第二个为下载类代码所要存放的本地路径,结合(1),下载操作描述如下:

```
[WebMethod] public void GetCode(int myAgentID,string desPath){
        string ServerAgentClassPath = FindAgentPath
        (myAgentID)
        //查找存放标识为 myAgentID 的类代码路径
        System.NET.WebClient Client=new WebClient();
        Client.UploadFile ( " http://" + desPath ,
            ServerAgentClassPath);}
```

4.4 基于 Web 服务的代码迁移交互过程

(1)首先,Client 将准备迁移的移动代理类代码编译成动态链接库(.dll)的形式(C# 中源文件为.cs,编译后为.dll);

(2)Client 调用代码上传 Web 服务的方法 PutCode(),将代理体上传到指定的网络类代码存储服务器上;

(3)当在新节点上准备恢复代理的代码时,通知该节点的移动代理服务器调用代码下载的 Web 服务的方法 GetCode(),从指定的网络类代码存储服务器上下载类代码,以用于代理对象的恢复。

结合代理的数据态和代码态的迁移,代理的整个过程如图4所示。其中,在反序列化过程中,启用代码下载的 Web 服务,实现代码和数据态的迁移,并恢复代理的执行。

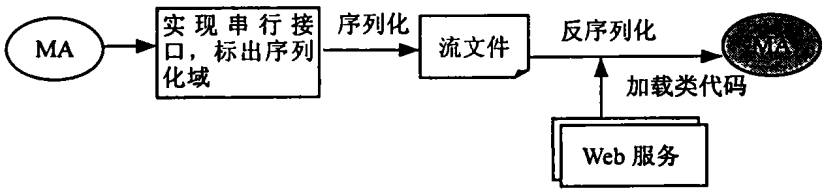


图4 代理迁移过程

结束语 本文对代理的迁移原理和代理三种状态的迁移问题进行了分析,研究了基于序列化技术、Web 服务实现代理迁移方法。描述了.NET 的 Binary 序列化和反序列化技术实现代理数据态迁移的方法,及基于 Web 服务调用的代码态迁移的方法和服务的构造及调用过程。目前我们仅实现了.NET 环境下的弱迁移技术,有关代理执行态的迁移,我们正在研究之中。随着.NET 的应用越来越多,对基于.NET 的移动代理系统的需求也会越来越多,我们在这方面进行研究,希望能够为实现基于.NET 的应用程序的开发提供支持。

参考文献

1 Lange D B, Oshima M. Seven good reasons for mobile Agents. Communications of ACM, 1999, 42(3):88~89

2 Torsten I, Frank K. Migration of mobile Agents in Java: Problems, Classification and Solutions, MAMA'2000. Austrilia: 1574-362

3 Alberto R S,Entl A R. Towards a Reference Model for Surveying Mobile Agent Systems. Autonomous Agents and multi-agent systems,2001,4:187~231

4 Lange D B. Java Aglet application programming interface. IBM Tokyo Research Lab. http://www.trl.ibm.co.jp/aglets, 1997

5 刘晓华. 精通.NET 核心技术. 电子工业出版社,2002

6 World Wide Web Consortium. SOAP 1.1. http://www.w3.org/TR/#Notes.2001.1

7 郑小平..NET 精髓-Web 服务原理与开发[M]. 北京:人民邮电出版社,2002

8 IBM,Ariba,Microsoft. UDDI data structures reference[EB/OL]. http://www.uddi.org/pubs/Iru UDDI Tech. White-Paper, 2000

9 Tidwell D. Web service architecture. http://www6.software.ibm.com/developerworks/education/wsbasics/, 2001

10 Meredith G, Curbera F. WSDL specification, W3 C. http://www.w3c.org/TR/wsdl, 2001

11 Microsoft.NET Framework SDK Documentation: Serializing Objects. http://www.microsoft.com.2002.3

12 麦中凡,陆永定. C# 编程语言[M]. 北京:航天航空大学出版社,2001