利用 J2EE 平台实现 Agent

The Realization of Agent Based on J2EE Platform

赵 灿 王万森 （首都师范大学信息工程学院 100037）

摘要：本文针对目前 Agent 系统大多处于实验阶段这一现状，提出了利用先进的工程技术——J2EE 实现 Agent 系统的方案，并在该方案中采用了 Session Fa? ade 等多种设计模式，有效的改善了 Agent 系统的效率，提高了系统的可维护性和可重用性。

关键词：J2EE Agent 设计模式

1 概述

Agent 作为一项分布式人工智能技术，近年来得到飞速发展和广泛应用。Agent 是指具有自治性、社会性、反应性的智能主体，它能够决策自己的行为、并能够对环境做出相应的反应，以实现自己的目标。

多 Agent 系统（Multi – Agent：MAS）是由多个具有自治行为的 Agent 组成的，这些 Agent 之间相互协商、合作以解决单个 Agent 无法解决的复杂问题。以前对于 MAS 的研究主要是从面向合作的视点出发考虑，现在则趋向于从单个 Agent 的角度来考虑更一般的问题，即 Agent 应具有什么样的结构，才能在一个有时间约束的、开放的环境中自主行动、决策及与其他 Agent 交互。

J2EE 定义了开发和运行分布式企业级应用的标准，其构架为组件开发模型提供了广泛的支持，并且 J2EE 服务器以容器的形式提供了诸如多线程操作、资源共享等底层服务，从而使得编程人员只需关注业务逻辑的实现 ，就能获得很好的系统性能。同时，J2EE 平台是基于 Java 语言的，它继承了 Java 语言的平台无关性、网络计算等优点，能够很好的处理异构问题和遗留系统问题。

2 “插件式”构造 Agent

应用于不同领域的 Agent 在功能上虽然千差万别，但是就其本质而言，都是具有自主性、社会性、反应性和智能性的智能体。Agent 求解问题的能力可以分为两个层次：领域级求解问题能力和 Agent 级基本能力，其中，领域级求解能力指获取领域知识和解决特定的领域问题，Agent 级基本能力包括通信、合作、协调、任务执行和控制异常处理等。因此可以采用“插件式”的方法构造 Agent，即：把 Agent 的基本功能进行封装，定义成一个对于所有 Agent 都相同的内核，然后通过标准接口，把完成领域任务的功能模块插入到内核之上。采用“插件式”方法，可以在内核的基础上方便、快捷地构造不同的多 Agent 系统。

Agent 的结构如图 1 所示。消息处理器把接收到的消息提交给规划器，规划器查询黑板上的领域功能模块信息，规划 Agent 工作计划，并调度领域功能模块以完成任务求解。如果遇到 Agent 自身无法解决的问题，规划器将任务转交给协作引擎，协作引擎查询相识者模型，寻找可以协作求解的其他 Agent。

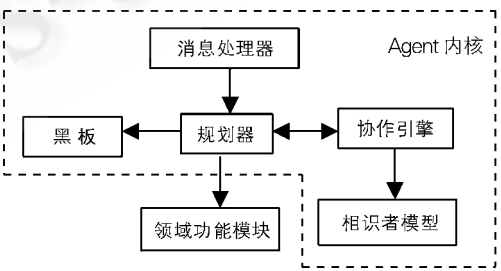

图 1 Agent 结构图

3 利用 EJB 实现 Agent

为了实现 Agent 的自治性、社会性、反应性，我们需要结合使用会话 Bean、实体 Bean 和消息驱动 Bean。首先介绍领域相关的功能模块的实现。由于功能模块代表系统的行为，我们采用会话 Bean 来实现它。这里利用了 J2EE 平台的 Session Fa? ade 模式，隐藏了功能模块的实现细节，通过对外提供统一的粗粒度接口，一方面简化了调用方的调用过程，另一方面减少了远程调用的次数，从而提高了系统效率。功能模块的统一对外接口如图 2 所示，其中方法 exec( ) 实现功能模块的功能，而方法 getMessage( ) 返回执行信息。

另外，J2EE 平台还提供了对 CORBA 以及 Web Service 的支持，这样我们可以很容易地访问已有的异构应用程序。我们只需要为原来编写的任何程序添加一个对外的远程接口，就可以将其作为 Agent 的一个功能模块，这样就很好地实现了软件的重用和移植。进一步说，如果积累了大量的功能模块，那么可以形成一种比“类库”更为高级的面向问题的“功能模块库”。这些模块可以随意组合到 Agent 内核上形成具有某种功能的 Agent。

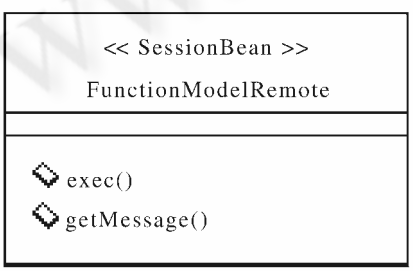

图 2　功能模块的远程接口

4 基本功能模块

以下介绍 Agent 的内核部分，也就是 Agent 的基本功能模块。内核是实现 Agent 整体功能的基础，它的性能以及结构直接决定了 Agent 功能的实现。在 Agent 内核中，主要有黑板、规划器、协作引擎等五个模块。

4.1 黑板

为了便于 Agent 内部模块之间传递消息以协调它们之间的行为，我们设置公共数据区——黑板。黑板的主要功能有：

(1) 存放功能模块的信息，包括功能模块的名称、能完成的任务等；

(2) 存放计算的中间结果，这样一方面可以避免重复计算，另一方面能尽早发现冲突，这对于冲突的消解是非常有利的；

(3) 存放模块间的请求，实现模块间的合作。可以采用 Java 类实现黑板：

```
public class BlackBoard{
    private int stub_count;//记录当前的存根号,以便生成新的唯一存根
    private boolean lock;//控制对黑板的并发操作
    private Vector itemList;//存放黑板条目
    public synchronized boolean addItem(BlackBoardItem item);//增加黑板条目
    public synchronized boolean modifyItem(BlackBoardItem oldItem, BlackBoardItem newItem);//修改黑板条目
    public synchronized boolean deleteItem(BlackBoardItem item);
    public synchronized BlackBoardItem getItem(int stub_id);//获取黑板条目
    public synchronized int getNextStub();//获取新的唯一存根号,用于标识黑板条目
    public synchronized String getModelName(String task); //获取有能力完成任务的功能模块名称
}
public class BlackBoardItem{
    public String fromModelName;//在 Agent 中,每一个模块都有一个 name
    public String toModelName;
    public int messageType;
    public int stub_id;//黑板条目的存根号
    public String content;//黑板条目的内容
}
```

4.2 Service Locator

Agent 在完成任务过程中往往需要访问数据库、消息队列等系统资源，有时还需要与其他 Agent 合作，这些都离不开资源的查找。J2EE 平台提供的命名和目录服务(JNDI)，使得应用程序可以与命名和目录服务器无缝连接，实现资源的获取。同时 JNDI 还抽象了资源的实际物理地址，有效的简化了资源查找的复杂性。

但是利用 JNDI 查找资源的成本很高，并且查找方法会因为使用不同厂家的命名和目录服务器而改变，这会影响代码的可移植性，所以我们采用 Service Locator 模式抽象所有的 JNDI 查找。利用该模式，可以缓存所有的 JNDI 查找结果，这将有效的减少反复查找的费用。

```
public class ServiceLocator{
    Hashtable modelCache = new Hashtable();//modelCache 负责缓存已查询的功能模块
    public static FunctionModelRemote getModel(String modelName){
        if(modelCache.containsKey(modelName))//判断
```

```
此功能模块是否已缓存
        return (FunctionModelRemote) modelCache.get(modelName);
      else{
       //如果功能模块没被缓存,则从命名上下文中获取,并将其缓存
        EJBHome home = context.lookup(modelName);
        FunctionModelRemote remote = home.create();
        modelCache.put(modelName,remote);
        return remote;
       }
      }
      public static Agent getAgent(String agentName){...}//获取 Agent
      public static Connection getDBConn(String database){...}//获取数据库连接
      public static QueueConnection getQueueConn(String queue){...}//获取消息队列连接
    }
```

4.3 规划器

规划器负责规划、调度、协调各个功能模块。它首先分析 Agent 的目标(goal),并将目标分解为一系列的任务(task),然后再查看黑板上的功能模块信息,将任务交给有能力完成的功能模块,如果没有合适的功能模块,它将把任务交给协作引擎,请求其他的 Agent 协作完成。

规划器采用基于知识的方法处理自己的目标,即为每一目标建立一个模板存放于知识库中,在模板中存放实现目标所要执行的任务。表 1 是知识库的一个示例片断,可以看出"查询故障"这个目标被分为了五个任务。

表 1 知识库示例

Goal	taskSeq	task
查询故障	1	判断是否存在故障
查询故障	2	根据专家知识来诊断故障
查询故障	3	利用 BP 神经网络开发自学习能力,进一步诊断
查询故障	4	进一步询问用户信息
查询故障	5	最终分析可能出现的故障以及处理方法。

我们可以利用 J2EE 平台提供的 EntityBean 代表知识库的数据,这将简化数据库的访问过程,另外,EntityBean 对数据的并发访问和数据关联都有很好的支持. 规划器的功能通过 SessionBean 来实现,具体代码如下:

```
public void plan(Goal g){
  ArrayList list = KnowledgeDB.lookup(g);//? 查询知识库,获取目标所对应的任务队列
  BlackBoard blackboard = new BlackBoard();
  for(int i=0;i<list.size();i++){
   String task = (String)list.get(i);//取得任务内容
   String modelName = blackboard.lookup(task);
   //查询黑板,寻找能完成任务的功能模块名称
   if(modelName!=null){
    FunctionModelRemote remote = ServiceLocator.getModel(modelName); //生成功能模块接口
    remote.exec();//将任务交给功能模块完成
   }
   else
    CooperationEngine.exec(task);//如果没有合适的功能模块,将任务交给协作引擎
  }
 }
```

4.5 协作引擎

协作引擎负责查询相识者模型,寻找可以协作求解的 Agent 并请求其他 Agent 协作完成任务。相识者模型中存放其他 Agent 的模型——包括职责、技能、资源、目标等。协作引擎完成任务的代码如下:

```
public class CooperationEngine{
 ArrayList recongizance;//相识者模型队列
 public void exec(Task task){
  for(int i=0;i<recongizance.size();i++){
   AgentModel model = (AgentModel)recongizance.get(i);//获取其他 Agent 的模型
   if(model.canExec(task))//根据 Agent 模型,判断 Agent 是否能够完成某项任务
    Agent agent = ServiceLocator.getAgent(model.getName());//查找 Agent
    Goal goal = new Goal();
    Goal.add(task);
    SendMsg(agent,goal);//发送消息给 Agent,请求其完成目标
  }
 }
}
```

4.5 通信器

为了实现 Agent 之间的合作,Agent 必须包含通信模块。J2EE 平台提供了远程方法调用(RMI)和 Java 消息服务(Java Message Service:JMS)两种交互方式。前者采用同步的方式及时执行请求,但客户端必须阻塞于请求等待请求结果;后者采用异步的方式执行请求,这样客户端不必阻塞于请求,系统的效率能得到很大改善。另外,面向消息中间件

(MOM)为 JMS 提供了存储转发、担保发送等服务。

Agent 最显著的特点之一就是自主性,即 Agent 有能力决定它所要追求的目标以及如何实现这些目标。Agent 接受外界的请求,但是否执行请求取决于其自身,并非取决于请求方。由此看出,RMI 并不适合 Agent 之间的通信,因而在 RMI 方式中决定权在于调用者。所以我们采用异步方式进行通信,这在一定程度上改善了 Agent 之间的耦合程度,提高 Agent 的自主性。另外,采用异步方式进行通信,使得多个 Agent 可以并行地处理事务,从而系统工作效率得到显著提高,同时处理事务的能力得到了增强,能够应付非常复杂的环境变化。

J2EE 平台提供的 Java 消息服务(JMS)支持异步处理,同时 J2EE1.3 版本还提供了消息驱动 Bean,这种 Bean 能够很方便的访问消息队列或消息主题。我们利用消息驱动 Bean 编写 Agent 之间交互的接口,通过该接口可以向 Agent 发出指令以期望 Agent 实现特定的目标,或者访问、修改 Agent 的属性,或者影响 Agent 的行为模式。以下是消息驱动 Bean 处理消息的方法代码:

```
public void onMessage(Message message){
 Object obj = ((ObjectMessage) message). getObject
();
 //外界请求 Agent 实现目标
 if(obj instanceof GoalMessage){
  Goal g = ((GoalMessage) obj). getGoal();
  new Program(). plan(g);//将目标交给规划器以实
现
 }
 //外界访问或者修改 Agent 的属性
 else if(obj instanceof StateMessage){
  StateMessage stateMsg = (StateMessage) obj;
  String action = stateMsg. getAction();
  if(action. equals("SET")){//如果外界期望设置 A-
gent 属性
   Goal g = new Goal("SetAttribute", stateMsg.
getAttributeName(),
    stateMsg. getAttributeValue());
  new Program(). plan(g);//将属性设置工作交给
规划器完成
 }
 else{
  Goal g = new Goal("GetAttribute", stateMsg.
getAttributeName());
  String attributeValue = new Program(). plan(g);
  ResultMessage resultMsg = new ResultMessage
(stateMsg. getAttributeName(), attributeValue);
  sendMsg("resultQueue", resultMsg);//将属性值作
为消息发送到结果队列中
 }
}
```

5 小结

Agent 技术的出现,尤其是 Agent 之间自主协调合作这一特点给软件开发带来了革命性的变化。但是这一协调合作是以 Agent 自身的结构作为基础的。本文利用了 Java 语言的跨平台特性以及 J2EE 分布式平台特性,提出了 Agent 的实现方案,由于该 Agent 具有了通信模块,能够很好的与外界交互,实现可扩展的多 Agent 系统。本方案为 Agent 技术应用于实践开拓了新的思路。

参考文献

1 Ed Roman. Mastering Enterprise JavaBeans[M],电子工业出版社,2002。

2 Graig A. Berry,John Carnell,Matijaz B. Juric 等著,邱仲潘等译,实用 J2EE 设计模式编程指南[M],电子工业出版社,2003。

3 姚莉,智能协作信息技术[M],电子工业出版社,2002 -03。

4 刘芳、姚莉、张维明、武强,主体接口技术研究及其 Java 实现[J],计算机工程,2003,4:p51 -p52,p81。

5 甘雯、李陶深,基于 Multi - Agent 的农业专家系统在 Internet 上的系统设计[J],计算机工程与应用,2000,10:p164 -p165,p173。