计算机科学2000Vol.27№ 10

基于"事件-条件-动作" Agent模型的冗余服务管理框架*)

An Management Architecture for Redundant Servers Based on "Event-Condition-Action"Agent Model

钱 方 贾 焰 郑光辉 邹 鹏
(国防科技大学计算机学院 长沙410073)

Abstract In order to implement dynamic and adaptive management facilities, this paper presents a management architectrue based on "ECA(Event-Condition-Action)"Agent model for redundant servers in WAN environment.

Keywords System management, Redundant servers, Distributed objects, Agent, ECA rules, CORBA

在目前大规模事务处理等商业应用的驱动之下，越来越多的分布式应用采用冗余服务以提高系统的性能和可用性。针对现有的冗余服务管理系统[1~4]缺乏自动和自适应的管理功能，本文提出基于"事件-条件-动作 ECA(Event-Condition-Action)"规则的 Agent 模型，并基于该模型创建了针对冗余服务的管理框架——对象管理者 OM(Object Manager)。在 CORBA[5] 平台上的应用实践表明，它能够满足分布式系统管理的要求。

1 基于 ECA 规则的 Agent 模型

Agent，被定义为在分布计算环境中持续自主发挥作用的计算实体。为了使 Agent 的管理行为能够适应分布式应用的多样性和服务资源的动态变化，本文对文[6]中以事件处理为中心的 Agent 模型进行了改进。通过引入"事件-条件-动作"规则，定义了一个基于 ECA 规则集的 Agent 模型，以实现系统管理策略的定制化和可配置。

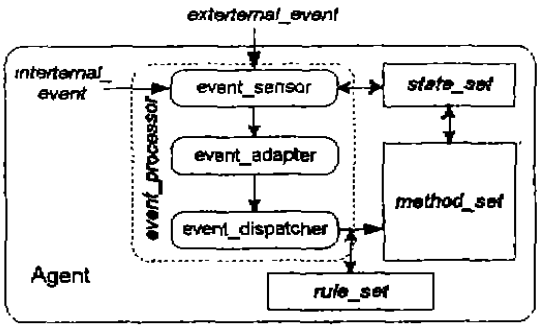

图1 基于 ECA 规则的 Agent 模型

基于 ECA 规则集的 Agent 模型由事件处理系统、方法集、内部状态集和规则集构成(图1)。其中，Agent 的状态集和方法集构成局部于 Agent 的知识表示系统；规则集与事件处理系统分离，它明确定义了事件集和方法集之间的映射关系，与事件处理系统一同构成局部于 Agent 的行为控制机制。事件处理系统涉及事件感知、事件适配和事件处理分发三个环节的活动，Agent 的活性就体现为事件处理系统在 Agent 生命期内始终持续自主地工作。当事件处理系统捕捉到所关注的事件时，它根据 Agent 内部定义的规则集引发相应方法的执行。具体的定义如下：

定义1 Agent 被定义为一个四元组：

$agent::=(event\_processor, method\_set, state\_set, rule\_set)$

其中，*event_processor* 是 *agent* 的事件处理系统、*method_set* 是方法集，*state_set* 是内部状态集，*rule_set* 是 *ECA* 规则集。

定义2 事件处理系统由一个四元组表示：

$event\_processor::=(event\_set, event\_sensor, event\_adapter, event\_dispatcher)$

其中，*event_set* 是事件集，*event_sensor*、*event_adapter* 和 *event_dispatcher* 分别为事件感知器、事件适配器和事件转发器。

定义3 事件集的定义为：

$event\_set::=(internal\_event, exteranl\_event)$；

其中，*exeteranl_event* 是外部事件；内部事件 *interanl_event* 包括时钟事件、内部状态被修改等事件。

规则集由各个 ECA 规则组成。

定义4 ECA 规则由一个三元组表示：

$ECA\_rule::=(event, condition, action)$；

*)国家863重点项目(项目号：863-306-ZD02-02-57)资助。

其中，*event* 是规则所关注的事件，*condition* 是引发动作执行的逻辑条件，*action* 被定义为方法集中的一个或一组顺序执行的方法，或者是一组在 *Agent* 内部顺序执行的操作。

2 冗余服务管理框架

2.1 管理模型

采用分布对象技术，本文对广域网环境中的冗余服务进行封装，并将冗余服务划分成多个管理域(domain)，在每个管理域内部，采用"一对多"的 Manager-Agent 集中管理模式，冗余服务的管理者——OM 介于客户和冗余服务之间，与驻留在被管理实例对象上的服务代理 SA(Server Agent)协同实现冗余服务的管理功能(图2)：OM 将管理命令发送给各个被管理实例的 SA，SA 通过向 OM 发送管理事件传递被管理实例的状态信息。

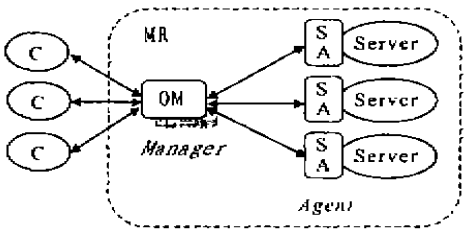

图2 OM 的管理模型

2.2 管理功能

针对冗余服务的容错和负载平衡问题，OM 为服务类的实例提供了配置管理、失效管理和性能管理：

(1)配置管理：为系统管理员提供接口创建、删除实例对象，接收实例对象的注册或注销，存储和维护实例对象的管理信息。

(2)失效管理：对实例对象进行实时监测，对失效的实例对象进行恢复。

(3)性能管理：以提高冗余服务的性能和可用性为目标，动态收集实例对象的管理信息；将容错和负载平衡结合在一起考虑，采用不同的管理策略，对发送给冗余服务的请求进行调度。

3 对象管理者 OM 的 Agent 模型

3.1 定义

为了实现自动、自适应的管理功能，本文将基于 ECA 规则的 Agent 模型应用到 OM 中(图3)。

定义5 OM 的状态集定义为：

$state\_set ::= (Object\_List, Request\_Queue)$

其中，*Object_List* 为 *OM* 所管理的实例对象的信息集合，它由各个实例信息项组成，包含了实例对象的地址

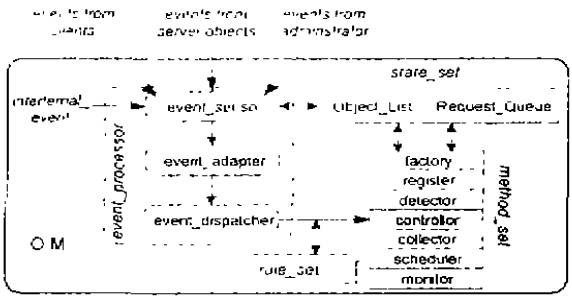

图3 OM 的 agent 模型

信息、当前负载值等管理信息；*OM* 将客户发送给冗余服务的请求放入队列 *Request Queue* 中，每个请求项包含请求的方法名、参数、优先级等管理信息。

定义6 OM 的方法集定义为：

$Method\_set ::= (factory, register, detector, controller, collector, scheduler, monitor)$

其中，*factory* 方法和 *register* 方法实现冗余服务的配置管理；*detector* 方法、*controller* 方法和 *monitor* 方法对冗余服务进行失效管理；*collector* 方法和 *scheduler* 方法对冗余服务进行性能管理。

定义7 OM 的内部事件集的定义为：

$internal\ event ::= (timer\_event, instance\_failure)$

其中，*timer_event* 是时钟事件，当 *OM* 的事件处理器感知到该事件，它引发 *detector* 方法，对实例对象进行逐一检测；若超过系统设定的超时时间(*timeout*)未收到应答，则 *OM* 断定实例失效，产生 *instance_failure* 事件。

OM 对外的接口由面向客户、冗余服务和系统管理员三部分组成，因此它的外部事件定义为：

定义8 OM 的外部事件集的定义为：

$internal\_event ::= (request\_arrive, result\_return, ack\_event, instance\_overload, instance\_register, instance\_unregister, create\_instance, destroy\_instance, recover\_instance, inquire\_instance, inquire\_request)$

其中，*request_arrive* 是客户向冗余服务发送请求时所产生的事件；*result_return*、*ack_event*、*instance_overload*、*instance_register*、*instance_unregister* 是实例对象产生的事件；*create_instance*、*destroy_instance*、*recover_instance*、*inquire_instance*、*inquire_request* 是系统管理员发出管理命令时所产生的外部事件。

3.2 自动和自适应管理功能

在分布式系统管理中，自动和自适应管理功能的实现分为三个层次：首先，能够针对被管理资源的状态(如实例的负载信息和运行状况)，采取相应的管理措施；其次，具有可扩展性，能适应被管理资源数目的动态改变(如实例的动态增加或减少)；第三，能够针对不

同的应用类型，采取不同的管理策略

在ECA规则中，一个事件可能引发多个方法的执行，尤其通过逻辑条件condition，可以更加灵活地实现事件和方法之间的映射关系。因此通过定义ECA规则，当冗余服务的状态和数目动态改变，即发生实例失效、重载，增加或减少等事件时，OM可以采取相应的管理措施，从而实现了第一、二层次的自适应管理功能。而且，由于ECA规则集与事件处理系统分离，因此可以针对不同的服务对象以及应用类型，通过定义不同的规则集实现管理策略的定制化和可配置。例如不同的应用对实时性的要求不同，针对实例对象的重载事件，OM可以采取不同的管理策略以维护冗余服务的性能。

规则1 *ECA_rule1 ::= (instance_overload, there are lightload objects in Object_List, scheduler);*

规则2 *ECA_rule2 ::= (instance_overload, -, factory)*

规则1定义了一种被动的性能管理策略：当*Object_List*中存在轻载实例时，通过引发*scheduler*方法，请求被转发给轻载实例执行；但当*OM*所管理的实例都重载时，客户的请求有可能得不到及时的响应。规则2定义了一种主动的性能管理策略：当实例都重载时通过引发*factory*方法，动态创建新的实例对象，以均衡系统的负载。它提高了请求的响应速度，但占用的服务资源较多。因此对于实时性要求严格的应用，可以采取规则2的管理策略，而其它应用则采取规则1的管理策略。

4 实现

本文采用对象管理组OMG的通用对象请求代理框架CORBA[5]作为实现平台。采用的具体平台是我们开发的StarBus分布计算环境，它遵循CORBA2.0标准。

在StarBus系统中，SA驻留在对象实现的skeleton中，由IDL编译器自动生成；OM介于客户和服务类的实例对象之间，作为CORBA的系统服务实现。按照基于ECA规则的Agent模型，OM在CORBA平台上的实现分为三层：底层的数据库实现了OM的状态集；利用面向对象技术，OM将方法集中的方法封装在各个CORBA对象中；同时OM实现了面向客户、系统管理员和冗余服务的外部策略对象，以及面向OM内部事件的内部策略对象。

通过将OM的ECA规则封装在策略对象的方法中，将OM所关注的事件定义为方法的入口，可以灵活地实现OM的管理策略。而且针对不同的应用类型，只需替换OM相应的策略对象，就可实现自适应的管理功能。

5 相关工作比较

文[1]通过将冗余服务的管理者SM分别驻留在客户方和服务方，实现对冗余服务的分散管理。它的缺点是没有引入分布对象的观点，因而对异构平台上的服务不能进行有效的管理；同时由于采用分散管理方式，各个管理者之间需要协调，造成管理冗余服务的难度和复杂度较大。ROMANCE[2]和Arjuna[3]系统通过引入对象的观点对异构服务进行封装，能够管理系统中异构的服务资源，但它们只能基于特定平台，没有与其它管理系统互操作的开放接口；同时缺乏对自动和自适应管理功能的支持。IBM的Component Broker[4]基于CORBA平台实现了对冗余服务的工作量(workload)管理，它能够对异构环境中的冗余服务进行管理，但它采取分散管理模式，将冗余服务的管理者驻留在客户方的扩展ORB中，同时缺乏全局的系统管理视图。

综上所述，与其它管理框架相比，对象管理者OM满足了分布式系统管理的要求：1)开放：通过采用分布对象的观点，将冗余服务封装成对象，OM可以基于广域网环境对异构服务进行管理，而且通过采用CORBA标准作为实现平台，可以提供开放接口，实现OM与其它管理系统之间的互操作；2)自动：基于Agent模型，OM成为分布计算环境中独立、自主的计算实体，因此能对冗余服务进行自动管理；3)自适应：基于ECA规则，OM不但可以适应服务资源状态和数目的动态改变，还可以根据不同的应用类型，实现相应的管理策略；4)OM基于CORBA平台为客户实现了透明的管理机制，同时为系统管理员提供了全局的管理视图，除此之外，OM采取manager-agent的集中管理模式，不但减少了系统的管理开销，还降低了管理的复杂度。

参考文献

1 Nehmer J, Mattern F. Framework for the organization of cooperative service in distributed client-server systems. Computer Communication, 1992, 15(4): 261～269

2 Rodrigues L, Verissimo P. The ROMANCE approach to replicated object management. In: Proc of the 4thWorkshop on Future Trends of Distributed Computing System. Lisboa Patugal, 1993

3 Little M, et al. Object replication in Arjuna. Computing Laboratory, University of Newcastle upon Tyne. [Tech Rep: TR94-50], 1993

4 Mcfall C. An object infrastructure for Internet middleware-IBM on component broker. IEEE Internet Computing, 1998(March/April): 46～51

5 Object Management Group. The Common Object Request Broker: Architecture and Specification Revision 2.2. July 1998

6 王怀民，吴泉源，高洪奎，邹鹏. 基于agent的分布计算环境. 计算机学报(863专辑)，1996，19(3)