一种适用于分布对象环境的层次型故障检测方法的研究

李琪林[1] 甄 威[1] 周明天[2]

(四川电力试验研究院 成都 610072)[1] (电子科技大学计算机学院 成都 610054)[2]

摘 要 针对现有故障检测服务在灵活性、伸缩性和扩展性方面的不足,提出了一种适用于分布对象环境的层次型故障检测方法。研究工作以两种通用概念模型(即故障提供者-监控者-使用者模型和域模型)为基础,指出了相关基本问题的解决途径。该方法引入了混合型故障监控模式,弥补了现有监控模式的不足;使用异步的消息传递机制,实现了故障信息的及时传播;采用基于域控制节点的层次型故障检测方法,解决了域内不同粒度实体故障检测问题。与同类研究相比,该方法在灵活性、伸缩性和扩展性方面具有一定的优势。

关键词 故障检测,故障监控,层次型

中图法分类号 TN929.5 **文献标识码** A

Research on an Hierarchy Fault Detection Method for Distributed Object-oriented Environment

LI Qi-lin[1] ZHEN Wei[1] ZHOU Ming-tian[2]

(Sichuan Electric Power Test and Research Institute, Chengdu 610072, China)[1]

(Computer Science and Technologies School, University of Electronic Science and Technology, Chengdu 610054, China)[2]

Abstract Present fault detection services for distributed object-oriented environment have some deficiencies at flexibility, scalability and extensibility, thus a hierarchy fault detection method was presented. Two conceptual models, fault provider-monitor-user model and domain model, form the foundation of our method. Subsequently some solution ideas were proposed to resolve the related problems in fault detection. This method introduces hybrid monitoring mode to mend the deficiency of present mode, uses asynchronous message-notification mechanism to achieve the fault information propagation timely, and adopts a hierarchy method, which is based on domain control node, to solve the problem of fault detection of different granularity entities in domain. Compared with the related works, our approach has some advantages in flexibility, scalability and extensibility.

Keywords Fault detection, Fault monitoring, Hierarchy

1 引言

随着分布对象计算技术向关键业务应用方向转移,越来越多的部门,尤其是电力、电信、金融、宇航等关键业务部门,对分布对象系统呈现强烈的依赖性。而这些部门的应用往往必须连续、不间断地运行,一旦中断,将会带来无法估量的生命和财产损失。尽管复制技术可以满足分布对象系统的高可用性需求,但是如果冗余对象副本在产生故障后无法及时恢复,显然会导致可用的同类对象副本数目越来越少,最终造成该服务的终止。因此,仅有可用性支持是不够的,还必须提供可靠性支持,以便将故障服务恢复到正常状态。

故障检测[1]是对故障发生的一种认知过程,通常要求在故障恢复过程实现之前完成。系统只有检测到故障,才有可能进行故障的定位和恢复。相应地,故障检测是任何故障恢复措施所必需的前提,也就成为分布对象环境中可信系统必备的功能。

不幸的是,现有的故障检测服务[2-4]在支持故障检测功能的同时,往往也在灵活性、伸缩性、扩展性方面存在不足,从而导致它们大多只能适应小型的分布对象应用。一方面,它们往往集中考虑故障检测服务和服务对象之间的交互性,例如基于推和拉的故障监控模式,而对于应用特性和网络环境考虑不足;另一方面,这些故障检测服务大多只针对对象级的故障检测功能进行设计,而对分布对象环境中大量不同粒度的实体的故障检测功能考虑不够全面,这使得它们往往无法适应实际应用系统的需要。

本文的目的是设计一种适用于分布对象环境的故障检测服务,以支持大规模的分布对象系统。采用该方法很容易与现有的分布对象技术相适应,支持故障检测功能,以提高整个系统的可靠性。此外,该方法本身也是可伸缩的,在提供故障检测功能的同时,保证能够满足大规模分布对象系统的要求。

本文第2节是两种通用概念模型的描述;第3节则在概念模型的基础上,设计了一种层次型的故障检测方法;第4节进行相关工作比较;最后是本文的结论。

到稿日期:2008-04-01 本文得到科技部十一五科技支撑项目-"现代服务业服务交互支撑平台"(项目编号:2006BAH02A0407)的资助。

李琪林(1973—),男,博士,主要研究方向为电力自动化、分布对象技术、可信计算技术等;甄 威(1956—),男,教授级高工,主要研究方向为电力系统分析、电力自动化、RTDS等;周明天(1939—),男,教授,博士生导师,主要研究方向为计算机网络、分布对象技术、并行分布处理等。

2 概念模型

2.1 故障提供者-监控者-使用者模型

按照功能进行角色划分，故障管理系统主要包含 3 种角色：故障提供者、故障使用者和故障监控者，其模型如图 1 所示。它们的定义如下。

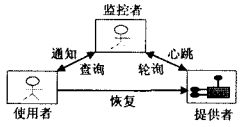

图 1 故障提供者-监控者-使用者概念模型

故障提供者：系统中被监控的实体，向外界以服务的形式提供一些可访问的接口，并具有清晰的访问语义，可以被客户应用访问。同时，为支持故障检测，故障提供者必须提供某种机制被监控者使用。

故障监控者：由系统提供的故障检测服务，用于收集故障实体的相关信息，使之对故障使用者可用。在实际应用中，故障提供者和使用者往往不存在固定的耦合关系，只是被监控实体状态发生改变时，故障使用者才会根据故障监控者提供的故障实体信息和故障提供者临时建立联系，执行必要的恢复动作。

故障使用者：使用故障信息的用户，通常即是系统中的基础设施成员或需要使用故障信息的客户应用。它通过故障监控者发现产生故障的服务实体，并接收监控者提供的故障通知消息，执行故障处理或恢复。

实现上，故障监控者通常表现为故障检测服务，故障提供者提供支持故障检测的相关接口，故障使用者则根据接收的故障通知信息完成故障恢复。

在实际系统中，模型通常会发生一定的演化。首先，故障监控者并非必需的，相应的故障检测功能可以集中在使用者那里，尽管从逻辑上看，这仍然符合基本的概念模型。其次，故障的通知功能可以从故障监控者中单独分离出来，相应增加一种新的角色：通知者。这时，监控者就单纯进行故障检测功能，而通知者就专注于将故障信息及时、有效地告知使用者。

上述各个角色之间的关系同时定义在图中，由角色的定义和分析不难理解其相互关系及涵义。

2.2 域模型

无论在伸缩性、扩展性还是灵活性方面，集中式体系结构都存在相当多的问题，对于拥有大量服务实体的分布对象环境并不适合。因此，本文引入第二个概念模型：域模型，如图 2 所示。域模型将整个分布对象环境划分为若干逻辑单元，同一域内可能存在大量的故障提供者和若干故障监控者，但都包含一个故障使用者。

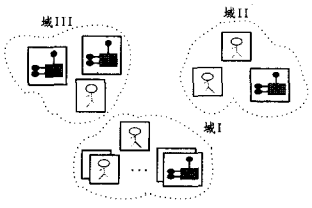

图 2 域模型

对于大规模的分布对象环境而言，域的划分是相当重要的。和大多数分布应用系统类似，域是由若干主机构成的逻辑上可管理的群体，在域内的系统行为、管理策略和容错方法大都是一致的。对于故障管理系统，域代表了集中式故障管理的最大范围和分布式故障管理的基本单位，体现了这两种机制的良好融合。

域将整个分布对象环境划分为若干逻辑范围，域模型的设计思想基于以下几点认识：

- 在多数情况下客户优先访问附近正常工作的服务实体；
- 集中式故障管理方法在小范围区域中具有明显的优势；
- 域的分割使大规模分布系统中的故障检测和恢复更加容易。

为便于讨论，本文以下部分假设文中针对的故障类型为崩溃故障，包括主机崩溃、进程崩溃和对象崩溃，但忽略对网络分区和 Byzantine failure[5] 的处理。

3 故障检测

根据前一节中的概念模型，分布在每个域中的故障监控者和使用者构成了故障管理系统的核心设施。它们相互协作，完成故障的及时检测与恢复。

为使故障实体的信息能被域中的使用者利用，系统必须能及时地获取其状态信息。该过程称为故障检测，包括故障信息的收集和传播。相应地，故障检测可进一步划分为两阶段：故障的监控和故障信息的传播。按照前述的概念模型，故障的监控过程就是故障监控者定期监控服务实体状态是否发生改变的过程；故障信息的传播则是指故障监控者在发现产生故障的服务实体时将故障信息发送到故障使用者的过程。

鉴于分布对象环境中存在大量不同粒度的实体，例如主机、进程和对象，因此故障检测服务必须针对不同粒度的实体，提供具有良好伸缩性、扩展性和灵活性的故障检测方法。

3.1 故障监控

在域范围内，故障监控方法可分为两大类：主动监控和被动监控，它们的根本区别在于故障信息提供者与故障监控者之间的交互关系。如果故障实体的信息是由监控者通过某种手段主动得到，那么就是主动监控，反之则是被动监控。有时，主动监控方法也称为拉模式（Pull Mode），被动监控方法则称之为推模式（Push Mode）。

在主动监控过程中，故障监控者通过定期地发送探询消息询问被监控实体是否存活。若被监控实体在规定的超时间隔到达前返回应答，则表明它工作正常，否则监控者认为该实体已发生故障。主动监控模式的 IDL 语言描述如图 3 所示，它描述了被监控实体和监控者间的相应接口。

```
//Active Monitoring IDL Description
Enum Status {Suspected, Normal};
Interface ActiveMonitorable
{
  Boolean Is_Alive();
};
Interface ActiveMonitor
{
  Void Start_Monitoring(in ActiveMonitorable obj);
```

```
  Void Stop_Monitoring(in ActiveMonitorable obj);
}
```

图 3　主动监控模式的 IDL 语言描述

在被动监控过程中，被监控实体主动实施行为。它们定期地发送心跳消息至监控者，表明它们依然存活。若被监控实体未在规定的超时间隔内送达要求的心跳信息，则监控者将引发一次故障怀疑。被动监控模式的 IDL 语言描述如图 4 所示，它定义了支持被动监控模式监控者和被监控实体的相关接口。

```
//Passive Monitoring IDL Description
Enum Status {Suspected, Normal};
//Forward Interface Declaration
Interface PassiveMonitor;
Interface PassiveMonitorable
{
  Void Send_Hearbeats(in PassiveMonitor m, in long mFrequency);
};
Interface PassiveMonitor
{
  Void Start_Monitoring(in PassiveMonitorable obj);
  Void Stop_Monitoring(in PassiveMonitorable obj);
}
```

图 4　被动监控模式的 IDL 语言描述

在具体实现中，故障监控方法大都采用超时机制。相应地，超时值的选择对于保证监控方法的正确性和完整性就显得至关重要。一方面，若超时值设置过短，尽管监控者可以迅速发现故障实体，但无疑也增加了监控者误判的概率，影响监控方法的正确性。另一方面，若超时值设置过长，相应地，故障监控者错判的概率将减少，但无法适应期望快速获得故障实体信息的故障使用者的需求。因此，故障监控方法必须根据应用的需求在超时值和正确性间进行适当的折衷。

在实际应用中，监控者和被监控实体之间具体采用哪种交互方式往往取决于监控者容许的故障信息延迟时间：若监控者期望在被监控实体发生故障时立即获得相关信息，则采用被动方法；若监控者只是打算在特定的时间点探询被监控实体的状态信息，则显然主动方法更适合。

相较之下，被动监控模式采用更加高效的通信方式，但故障监控参数，例如超时值，必须分布在所有服务实体上。同时，对于服务实体数量巨大的系统，该方法也会产生大量的心跳消息，易导致应答风暴。与被动方法不同，在主动方法中，超时值仅需位于监控者一方，但主动方法采用双向消息流轮询被监控实体，因此会引起极大的开销，从而导致性能无法接受。

考虑到主动方法和被动方法各有优缺点，我们采用混合型故障监控技术[3]来弥补二者的不足。混合型故障监控方法正是设计用于解决分布式故障信息收集问题的，其思想是将主动监控方法和被动监控方法相互融合，但同时又避免了二者的缺点，相应地，它采用两类消息对被监控实体的状态实施监控。正常情况下，被监控实体定期发送心跳消息到监控者，表明自己仍然存活。一旦被监控实体在特定的时间间隔内未发送存活信息至监控者，则监控者将发送探询消息，询问该实体是否工作正常。若该实体仍未送回相应的应答消息，则监控者对该实体触发故障怀疑。

混合型故障监控方法实质上是主动方法和被动方法的有机结合。它首先采用被动方法，等待来自被监控实体的心跳信息。若在特定时间间隔内未收到来自被监控实体的相关消息，随后它将采用主动方法探询相关实体是否存活，从而达到故障检测的目的。图 5 例示了混合型监控模式中监控者和被监控实体间消息交换的过程[3]。其中，被监控实体 M1 定期地发送心跳消息到监控者，而 M2 则在监控者主动轮询时才回送相应的存活消息。监控者采用两个超时间隔 T1 和 T2。在 T1 时间间隔内，监控者等待来自被监控实体发送来的心跳信息。若经过 T1 时间间隔后，监控者未收到相关存活消息，则它将启动超时间隔 T2 并发送探求消息，询问被监控实体是否活动。若在 T2 时间间隔内，监控者仍旧没有收到该实体发回的应答消息，则它将引发对该实体的故障怀疑。本例中，被监控实体 M1 在时间间隔 T1 内发送存活消息，随后它崩溃。接下来，由于监控者在 T1 时间段内未收到 M1 送来的心跳消息，因此它启动 T2 超时间隔并发送探询消息至 M1。若在 T2 间隔到达前，它还是未收到 M1 返回的应答消息，则监控者怀疑 M1 已崩溃。

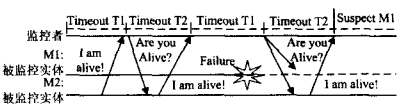

图 5　混合模式中的监控消息

混合型监控模式的 IDL 语言描述如图 6 所示，它具体描述了被监控实体和监控者间的交互接口。

```
//Hybrid Monitoring IDL Description
Enum Status {Suspected, Normal};
Interface Monitorable {};
Interface Monitor
{
  Void Start_Monitoring(in Monitorable obj);
  Void Stop_Monitoring(in Monitorable obj);
};
//Active Monitoring Mode
Interface AcitveMonitorable: Monitorable
{
  Oneway void Is_Alive();
};
Interface ActiveMonitor: Monitor
//Passive Monitoring Mode
Interface PassiveMonitor: Monitor {};
Interface PassiveMonitorable: Monitorable
{
  Void Send_Hearbeats(in PassiveMonitor m, in long mFrequency);
};
//Hybrid Monitoring Mode
Interface HybridMonitor: ActiveMonitor, PassiveMonitor {};
```

图 6　混合型监控模式的 IDL 语言描述

3.2　故障信息的传播

上一节解决了故障的监控问题，即监控者如何及时有效地收集状态不断变化的故障提供者的相关信息。对于故障信息的使用者来说，在执行恢复服务之前，必须首先从监控者处获取故障实体的信息，然后才能按照某种恢复策略对这些故

障实体执行相应的恢复，我们称该过程为故障信息的传播。本节说明如何将故障信息从监控者传递到使用者的相关机制。

具体而言，有两种基本的方式可以使故障信息的使用者从监控者那里获得故障实体的相关信息：同步和异步。前一种方式，使用者可以在特定的时间点同步地调用故障监控者提供的服务接口，获取相关的信息。异步方式中，使用者注册到监控者提供消息通知机制中，一旦监控者发现被监控实体的状态发生改变，它将该实体的故障信息立即传递给使用者。

从实际应用的角度考察不难发现，故障监控者和使用者之间的交互模式更多是由故障信息驱动的，较少存在显式的交互过程。因此，故障监控者和使用者的交互应当是松耦合的，二者间不宜采用同步方式传递故障信息。

相应地，我们采用异步的消息通知机制来解决这一问题。消息通知机制正是设计用于解决松耦合实体间的消息通信问题，其核心思想是获取故障信息的职责从利用信息的故障使用者一方转移到传播信息的监控者一方，从而将二者进行了有效的分离。一旦监控者发现被监控实体的状态发生改变，它负责将相关信息及时通知使用者。引入消息通知机制后，监控者与被监控实体间的故障监控模式也必须进行相应的扩展，以提供必要的支持。图 7 例示了采用异步消息通知机制后被动监控模式的 IDL 语言描述。

```
//Passive Monitoring with Asynchronous Message Notification IDL Description
Enum Status {Suspected, Normal};
//Forward Interface Declaration
Interface PassiveMonitor;
Interface PassiveMonitorable
{
  Void Send_Hearbeats(in PassiveMonitor m, in long mFrequency);
};
//Notification Service
Interface Notifiable
{
  Void Notify (in PassiveMonitorable obj, in Boolean suspected);
};
Interface PassiveMonitor
{
  Void Start_Monitoring(in PassiveMonitorable obj, in Notifiable n);
  Void Stop_Monitoring(in PassiveMonitorable obj, in Notifiable n);
}
```

图 7　采用异步消息通知机制后被动监控模式的 IDL 语言描述

4　基于域控制节点的层次型故障检测方法

第 3 节主要侧重于故障信息的收集和传播问题的解决。在此基础上，本节具体提出了分布对象环境中如何对服务对象进行故障检测的方法。

在分布对象环境中，服务对象可能频繁地加入或离开，其负载在不同时间差别可能很大。而服务对象数量的巨大导致出现故障的概率也随之增长，服务对象的状态总是处于不断变化中。根据前面定义的域模型，分布对象环境中的服务对象分布在各个域中。相应地，同一域内部可采用统一的故障检测方法，例如主动监控或被动监控。然而，若域内采用集中式的故障检测方法，即在整个域中只部署一个故障检测器，那么它很容易成为整个域的性能瓶颈并导致单点故障，从而使得故障检测器在可伸缩性、可靠性和灵活性方面都存在一定的问题，对于大规模的分布对象系统并不适合。换句话说，由单一的故障检测器完成一致、可靠、及时的故障收集功能是不可行的。

解决此问题的思路很自然会想到在域内引入分布式的故障检测体系结构。相应地，有两种模型可供选择：平板型和层次型。平板型指各个故障检测器完全平等，相互可直接通信，不需要中介；层次型是指故障检测器分布在各个层级中，每个检测器只能与上一层级或下一层级的检测器直接通信。

相比之下，我们认为，采用层次型模型是更好的选择。这是因为，分布对象环境中故障检测服务往往涉及不同的实体（如主机、进程和对象）。按照域的划分，域内的同一主机可能运行大量的服务进程，每个进程也会包含大量的对象。同时，不仅对象故障会导致它提供服务的终止，而且主机和进程的崩溃也会造成同样的后果，因此，故障检测服务应能提供不同级别实体的监控机制。而层次性模型非常适合这类针对大量不同粒度实体的故障检测服务。

采用层次型模型后，我们将整个域内的故障检测服务分为两级：第一级是分布在不同主机上的故障检测器，我们称其为局部故障检测器 LFD(Local Fault Detector)，它负责完成对本地所有对象的监控；第二级则是位于域控制节点上的一个特殊故障检测器，称之为全局故障检测器 GFD(Global Fault Detector)。域控制节点是每个域都需要维护的一个特殊节点，域内故障使用者 FIU(Fault Information User)也运行在这个节点上。由于本地进程或主机的崩溃会导致 LFD 失效，因此 GFD 的职责主要是对域内不同主机上的 LFD 进行监测，根据它们的状态改变识别是否产生主机级的故障信息。图 8 表示了域内故障检测服务部署的层次型结构。

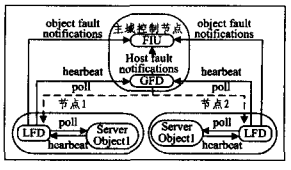

图 8　域内故障检测服务部署的层次结构

根据前面几节的讨论，LFD 和本地对象间以及 GFD 和 LFD 间均采用混合型故障监控方法，收集故障实体的相关信息。同时，为了将故障信息及时传递到故障使用者，LFD 和 GFD 都采用异步消息服务将检测到的故障实体相关信息通知故障信息使用者，从而引发故障恢复。按照前面定义的概念模型，实质上 LFD 是位于本地的故障监控者，而 GFD 则是位于域控制节点的整个域的全局故障监控者。它们和使用者彼此协作，完成故障的检测和恢复。图 8 也清晰示出了 LFD，GFD 和使用者间的消息交换过程。

5　相关工作及其比较

与本文提出的方法不同，已有的研究[2,4,6]往往采用推和拉监控模式，而忽视了应用特性和网络环境对故障检测服务

（下转第 295 页）

表 16-b 系统层实验结果

	$System\_Effort_{real}$	System_Effort = 1.814 * System_Size + 1006.179		
		System_Size	$System\_Effort_{pred}$	MRE
1	3268	1649.07	3997.59	0.22
2	3752	1338.37	3433.98	0.08
3	2106	807.02	2470.11	0.17
4	5352	1975.03	4588.88	0.14
5	2345	862.35	2570.48	0.10
6	1668	698.58	2273.40	0.36
7	2092	947.57	2725.07	0.30
8	4311	1685.08	4062.91	0.06
9	2392	1416.85	3576.34	0.22
10	2543	1382.55	3514.12	0.38
		MMRE		0.23
		PRED(0.25)		0.70

由实验的结果得出以下结论：(1) 模块层工作量的估算值平均相对误差为 0.13，被估算的 20 个模块中相对误差在 0.25 以内的有 17 个，占 85%；(2) 系统层工作量的估算值平均相对误差为 0.23，被估算的 10 个系统中相对误差在 0.25 以内的有 7 个，占 70%。

结束语 面向对象估算方法的研究从 20 世纪 80 年代开始，经过多年的研究与发展，理论上取得了比较大的进步，但是也存在一些比较大的缺点，例如统一的估算体系研究不足以及估算误差较大等。本文在传统面向对象软件估算方法的基础上，提出了一种多粒度的面向对象软件估算模型并对模型的 4 个层次作了详细的描述，然后利用最小二乘法回归分析探讨了规模与工作量的关系。最后通过实验数据的验证，证明了此估算模型是有效的。

参 考 文 献

[1] Pressman R S. Software Engineering : A Practitioner ' s Approach. Fifth Edition. McGraw-Hill, 2001

[2] Glass R L. Facts and Fallacies of Software Engineering. Boston: Addison-Wesley, 2003

[3] LI Ming-shu, HE Mei, YANG Da, et al. Software Cost Estimation Method and Application. Journal of Software, 2007, 18(4): 775-795

[4] Fetcke T, Abran A, Nguyen T-H. Mapping the OO-Jacobson approach into function point analysis // Proceedings of IFPUG 1997 Spring Conference. 1997: 134-142

[5] Uemura T, Kusumoto S, Inoue K. Function-point analysis using design specifications based on the unified modelling language. Journal of Software Maintenance Evolution-Research Practice, 2001, 13: 223-243

[6] Antoniol G , Fiutem R , Lokan C . Object - oriented function - points: an empirical validation. Empirical Software Engineering, 2003, 8: 225-254

[7] Karner G. Resource estimation for objectory projects. Objective Systems SF AB, 1993

[8] Braz M R, Vergilio S R. Software effort estimation based on use cases // Proceedings of the 30th Annual International Computer Software and Applications Conference

[9] Costagliola G, Tortora G. Class Point: An Approach for the Size Estimation of Object-Oriented Systems. IEEE Transactions on Software Engineering, 2005, 31(1)

[10] Banker RD, Kauffman RJ, Kumar R. An empirical test of object-based output measurement metrics in a computer aided software engineering environment. Journal of Management Information Systems, 1991-1992, 8(3): 127-150

[11] Whitmire S A. 3D Function Points: Applications for Object-Oriented Software // Proc. ASM'96 Conf. 1996

[12] Minkiewicz A F. Measuring Object Oriented Software with Predictive Object Points // Proc. Conf. Applications in Software Measurements (ASM'97). 1997

[13] Zivkovic A, Rozman I, Hericko M. Automated software size estimation based on function points using UML models. Information and Software Technology, 2005, 47: 881-890

(上接第 281 页)

的影响。此外，这些研究往往依赖于特定的分布对象平台，缺乏对于分布对象应用的通用性支持，因而限制了这类研究在遗留系统中的应用。

我们的方法采用的混合型监控模式与 OGS[3] 类似。本文的方法与 OGS 关键的区别之一在于我们按照域模型对分布对象环境进行了划分，并通过部署层次型的故障检测服务，实现了对域内不同粒度实体的故障检测，而 OGS 仅能提供对象级的故障检测功能。与 OGS 的另一区别是，本文按照功能角色的划分，明确提出了一种新的故障提供者-监控者-使用者概念模型。

FTBUS[4] 的设计思想在采用层次型故障检测机制与我们的方法类似，但它未采用混合型故障监控模式，同时引入 CORBA 事件服务来传播故障信息，相应地，对 CORBA 平台的依赖过强，可能导致单点故障。

结束语 故障检测对于分布对象环境中的容错系统具有非常重要的作用。然而，已有的系统和研究工作大多集中于对故障恢复问题的研究，较少考虑故障检测问题，而且多数系统采用集中式的故障检测方法，机制上普遍比较单一，因此故障检测服务在伸缩性、扩展性和灵活性方面往往存在问题。本文在基本的故障提供者-监控者-使用者概念模型基础上，引入了域的概念，将整个分布对象环境划分为若干域。在域内，是集中式的层次型故障检测方法，采用了混合型的故障监控模式，弥补主动和被动故障监控方法的不足；引入异步消息通知机制实现了故障信息的及时传播；层次型故障检测部署方式则能对不同粒度实体执行故障检测。

目前，我们提出的故障检测服务是适用于同一域内不同粒度实体的层次型故障检测方法，并未考虑域之间的全局范围内的故障信息传播问题。因此，下一步，我们打算引入更加灵活的移动 Agent 模型，实现全局范围内的故障信息传播。

参 考 文 献

[1] Tanenbaum A S. Distributed Operating System. Prentice Hall Inc., 1995

[2] Natarajan B, Gokhale A, Yajnik S, et al. DOORS: Towards High-performance Fault Tolerance CORBA // Proceedings of the 2nd Distributed Applications and Objects (DOA) Conference. Antwerp, Belgium, Sep. 2000: 21-23

[3] Felber P. The CORBA Object Service : a Service Approach to Object Groups in CORBA. PhD thesis. Switzerland: Swiss Federal Institute of Technology at Lausanne, 1998

[4] 周明辉. 面向对象的容错中间件的研究与实现. 博士学位论文. 国防科技大学, 2002

[5] Alvisi L, Malkhi D, Pierce E, et al. Dynamic Byzantine Quorum Failure Systems // Proceedings of the International Conference on Dependable Systems and Networks. June 2000: 283-292

[6] Schonwalder J, Garg S, Huang Y, et al. A Management Interface for Distributed Fault Tolerance CORBA Services // Proceedings of the IEEE 3rd International Workshop on System Management. Newport, RI, Apr. 1998