基于 JMS 的分布式 ESB 的设计与实现*)

符 宁[1,2] 周兴社[1] 张海辉[1]

(西北工业大学计算机学院 西安 710072)[1] (解放军西安政治学院 西安 710068)[2]

摘 要 企业服务总线(ESB)是基于 SOA 思想的企业业务集成基础软件架构。本文在分析集中式 ESB 局限性的基础上,提出了基于 JMS 构建分布式企业服务总线的设计方案。分布式 ESB 实现了总线负载均衡,克服了集中式架构存在的单点故障和性能瓶颈问题,并且支持消息传输的持久性。原型系统的试验表明该设计方案有良好的应用效果。

关键词 企业服务总线,应用集成,SOA

Distributed Enterprise Service Bus: Design and Implementation

FU Ning[1,2] ZHOU Xing-She[1] ZHANG Hai-Hui[1]

(School of Computer, Northwestern Ploytechnical University, Xi'an 710072)[1] (PLA Xi'an Institute of Politics, Xi'an 710068)[2]

Abstract Enterprise Service Bus is a software infrastructure based on SOA for application integration. After the limitations of centralized ESB are analysed, a scheme design of distributed ESB base on JBI is proposed in this paper. Distributed ESB has the merits of load balancing, allowing single node failure, and supporting persistent messaging. A distributed ESB also avoids producing the performance bottleneck. Experiments prove that it is correct and effective.

Keywords Enterprise service bus, Application integration, SOA

1 引言

在传统的 EAI 和 B2B 应用中,各解决方案提供商采用不兼容的技术来设计各自的功能系统。这种技术状况满足不了企业对各种应用灵活地进行集成的需要,不利于系统的扩展,同时给企业带来了高昂的集成和维护成本。SOA 是一种构造分布式系统的方法,它将业务应用功能以松耦合的服务单元的形式提供给最终用户或其他服务。通过服务重用的方法 SOA 能够大幅提高软件资源的适应性和开发效率[1]。企业服务总线技术在这种背景下产生,其思想是提供一种标准的软件底层架构,第三方的程序组件能够以标准的方式"插入"到该平台上运行,并且组件之间能够以标准的消息通信方式来进行交互[2]。企业服务总线技术克服了传统 EAI 技术的缺陷,能够对各种技术和应用系统提供支持,具有很强的灵活性和可扩展性,是目前企业理想的应用系统集成平台。

2 集中式 ESB 与 JBI 规范

JBI 即 Java Business Integration,是 SUN 公司提出的、得到 Apache、Borland 等多个组织支持的企业服务总线的规范。JBI 的目标是通过定义一个标准的集成架构为企业级的业务集成提供解决方案。JBI 规范采用了面向服务的设计思想,解耦了组件之间的联系,并基于标准的消息交互定义了组件之间的交互语义。JBI 规范还定义了 JBI 容器和组件之间的交互接口和 JBI 环境的管理的标准接口。JBI 规范的最新版本是 SUN MICROSYSTEMS 公司于 2005 年 8 月 23 日发布的 JBI1.0 最终版,也称 JSR208[3]。

按照 JBI 规范,企业服务总线环境包含三方面的内容:标准消息路由器(Normalized Message Router)、JBI 管理部分和 JBI 组件框架。NMR 实现组件之间的消息路由。JBI 组件框架为 JBI 组件提供一个可插拔的组件容器。JBI 管理部分基于 JMX 实现 JBI 环境中系统和组件的管理。JBI 环境的组成如图 1 所示。

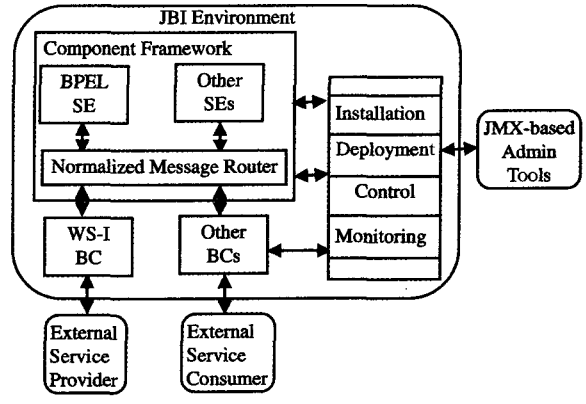

图 1 JBI 体系结构图

基于服务总线的应用由多个 JBI 组件组合而成,组件之间通过消息进行交互。组件接口和交互语义采用 WSDL 标准进行描述。与直接交互方式不同,JBI 环境中组件之间的交互过程是:首先由请求者产生一个服务请求消息,并将该消息按照 JBI 规范要求标准化。然后标准化的消息被发送给 NMR,NMR 根据请求消息中的服务名或者接口名进行目的组件查找,并最终将消息转发至目的组件。JBI 环境中存在两类组件:Service Engine (SE)组件和 Binding Component (BC)组件。Service Engine 类组件提供业务逻辑和数据转换类服务,例如 BPEL 引擎等。Binding Component 类组件用于

*)十一五国防预研项目(06004089)。符 宁 博士生,研究方向:分布计算、可信计算;周兴社 教授,博导;张海辉 博士生,研究方向:分布计算、网格计算。

连接 JBI 环境外部的服务，例如远程的 Web Service 等。连接在总线上的组件无需了解其他组件和应用系统的位置和交互协议，只需要向服务总线发出请求消息即可获得所需服务。服务总线事实上实现了组件和应用系统的位置透明和协议透明。技术人员可以通过开发符合 JBI 标准的组件(适配器)将外部应用连接至服务总线，实现与其他系统的互操作。

基于 JBI 规范的企业服务总线虽然具有很多优点，但是也具有一定局限性，主要表现在以下几点：

- 所有的 JBI 组件只能运行在一个 Java 虚拟机环境中，集成应用的负载集中在一个计算节点上，该计算节点是整个集成应用系统的性能瓶颈。
- 负责应用集成的服务会形成单点故障问题，当该节点出现故障，整个集成应用系统会处于瘫痪状态。
- JBI 规范目前不支持消息传输的持久性，一旦 JBI 节点出现故障，所有内存中的消息将被丢弃，JBI 环境重新启动后也无法重新获得丢失的消息。

针对 JBI 规范的局限性，我们提出基于 JBI 规范设计分布式企业服务总线的思想。分布式企业服务总线的设计目标包括：

- 实现服务总线的核心功能要求[2]，实现服务位置和访问协议的透明。
- 符合 JBI 规范，保持为应用程序提供一致的应用接口，符合 JBI 规范应用程序可以在不做改动的情况下在分布式 JBI 环境下运行。
- 负载均衡，将规范中的负载在多个计算节点上进行分摊，充分利用网络环境下节点的闲散的计算资源。
- 失效容忍，根据名字访问实现服务的位置透明，当一个计算节点失效时能够将服务请求消息自动路由至其他能够实现该服务的计算节点，提高系统的可用性。
- 支持消息传输的持久性，为提高系统可用性和故障处理提供支持。当计算节点或者服务实例故障时，处于传输状态的消息不会丢失。计算节点重新启动后能够获得故障期间其它组件发给自己的消息。

3 分布式企业服务总线设计方案

基于以上设计目标，我们提出如下分布式企业服务总线的设计方案。系统架构图如图 2 所示。

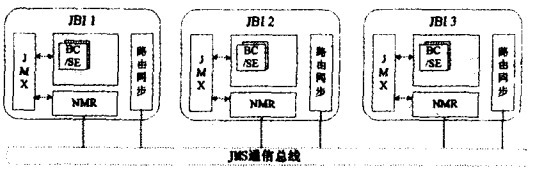

图 2 分布式企业服务总线架构图

分布式企业服务总线由通信总线和连接在通信总线上的多个对等的 JBI 节点构成。每个 JBI 节点都是符合 JBI 规范的独立 JBI 环境，支持符合规范的程序组件的运行。路由同步模块负责更新和同步各个节点上的全局消息路由表。通信总线负责不同 JBI 节点之间的消息和路由信息的传输。JMS 是标准的消息传输中间件，支持点对点和发布/订阅的消息传输模式，并具有支持消息传输的持久性的优点。因此，我们选择 JMS 作为 JBI 节点之间的消息通信总线。

3.1 分布式消息路由器

分布式企业服务总线设计的核心问题是如何实现不同 JBI 节点上的组件之间的消息交互。在保持消息路由器对上层程序组件提供的接口不变的情况下，我们设计了分布式的消息路由器(Distributed Normalized Message Router)，其架构图如图 3。DNMR 由物理上独立的多个 NMR 通过底层通信总线连接而成，在逻辑上可以将其看成跨节点的虚拟消息路由器。组件要与其他节点上的组件进行交互时，不考虑其他组件的位置，只需将服务请求消息发给本地的 NMR。DNMR 根据消息包中的服务名查找目的 JBI 节点，并进行跨节点的消息转发。

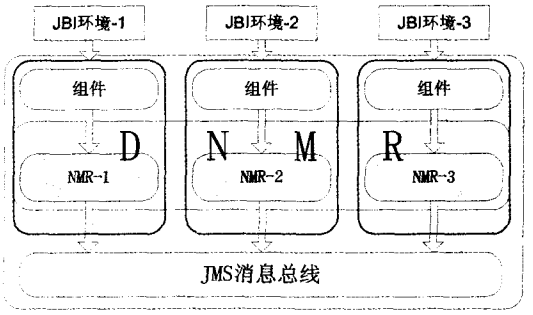

图 3 分布式消息路由器

3.2 消息传输策略

在分布式企业服务总线中消息传输需要借助于 JMS 消息通信总线。考虑到 JMS 是松散耦合环境下实现消息持久传输的标准机制，我们采用 JMS 作为 JBI 节点之间消息传输的通信总线，并采用如下的消息传输策略：

- 当组件 A 的交互对象组件 B 同处一个 JBI 环境中时，由该节点上的消息路由器直接进行消息转发。当组件 A 和组件 B 不在同一个 JBI 环境时，由组件 A 所在节点的 NMR 将消息通过 JMS 消息中间件发送至组件 B 所在节点的 NMR，再由该 NMR 转发给组件 B。
- 每个 JBI 节点中的 NMR 在系统 JMS 消息服务器上都拥有属于自己 JBI 环境中的 NMR 消息队列。该消息队列在 JBI 节点启动时进行创建。每个 NMR 监听自己的消息队列，当发现有发给自己的消息时进行处理，将该消息转发至目的组件。当 NMR 要将消息发送给其他节点的组件时，只需将该消息发送给该 NMR 对应的消息队列即可。

JMS 具有持久传输消息的能力，如果某个 JBI 节点出现故障，该节点重新启动后仍然能够收到发给该节点的消息。这种消息持久传输能力为提高系统的可用性和错误恢复机制提供了良好的支持。

3.3 消息路由表生成和维护策略

NMR 是组件之间消息交互的中介，为了能将消息正确的发送至目的组件，NMR 必须具有消息路由的能力。在单 JBI 环境中，组件加载的时候会向 JBI 容器注册该组件提供的服务和接口信息，并注册服务和接口的 endpoint 信息。endpoint 是与具体组件对应的消息传输的具体目的地址[3]，每个组件对象唯一对应一个 endpoint 对象。这些注册信息形成消息转发路由表。NMR 根据服务请求消息中的接口名或者服务名查找消息转发路由表，找到目的组件对应的 endpoint，并进行消息转发。分布式企业服务总线需要解决在全局环境下消息转发的路由问题，因此需要建立全局的消息转发路由表和实现全局范围内的路由发现。

我们采用如下的全局消息路由表生成和维护策略：

- 每个 JBI 节点的消息路由表分为内部 endpoint 与外部

endpoint 两部分。内部 endpoint 信息区保存本地的服务、接口和 endpoint 的对应关系。外部 endpoint 信息区保存其他 JBI 节点上的服务、接口、endpoint 到 JBI 节点的对应关系。这样事实上在每个 JBI 节点上都拥有全局的消息路由信息。

• 系统初始化的时候在 JMS 服务器上创建名为 Broadcast 的消息 Topic。所有 JBI 节点都通过 Broadcast Topic 发布路由变更及节点变更的信息。每个 JBI 节点启动后都会监听 Broadcast topic。

• JBI 节点启动时通过 JMS 服务器向其他 JBI 节点发广播消息，通知其他节点有新的 JBI 节点加入 ESB。该广播消息中包含该 JBI 节点的 ID 以及相关的通信地址(JMS 通信地址)。然后，该节点通过 JMS 服务器广播其上所有处于活动状态的 endpoint 的信息。

• 当节点 n 发现新节点 x 加入总线时，节点 n 首先将节点 x 的系统信息加入本地 JBI 节点信息列表。然后节点 n 将广播自身的 JBI 系统信息，以及在节点 n 上的活跃组件的 endpoint 信息。节点 x 根据其他节点的广播信息形成全局的消息路由表。

• JBI 节点每激活或卸载一个组件的时候，除了要向本地 JBI 容器注册以外，还要将本节点上的路由变动信息通过 JMS 服务器进行广播，通知其他的 JBI 节点更新路由表。

我们对在每个 JBI 节点保存全局路由信息基于如下考虑：首先，基于企业服务总线应用的路由信息相对稳定，不存在海量的 JBI 节点的节点同时参与应用集成的需求。每个 JBI 节点是自治的，可以动态地加入或者离开服务总线，但这种动态变化的频率是比较低的。因此用于维护消息路由的内存、CPU 和网络资源的开销不大。其次，在每个 JBI 节点保存全局信息的路由，加快了路由的查找速度和系统的运行效率。

3.4 节点失效处理策略

在分布式 ESB 环境中计算节点存在故障的可能。为了提高系统的可用性，我们采用了如下的节点失效处理策略：

• 每个 JBI 节点定时向 Broadcast Topic 发送 alive 消息，通知其他节点自身处于活动状态。

• 每个 JBI 节点保存 ESB 中所有其他 JBI 节点的信息列表。该列表同时记录最后一次收到各个节点 alive 消息的时间。

• 当节点 N 长时间收不到节点 I 的 alive 消息，则认为节点 I 失效，节点 N 删除本地全局路由表中有关该节点的路由信息，并向 Broadcast Topic 发送节点 I 失效消息。

• 当节点 N 收到节点 I 失效的消息，节点 I 删除本地路由表中所有有关该节点的路由信息。

• 当某个 JBI 节点上的 NMR 发现消息目的地(组件、服务、接口或 endpoint)不存在的时候，除了要向消息发送者返回 fault 消息外，还要向 Broadcast Topic 发送路由失败消息。其他 JBI 节点根据该消息传输失败信息调整或者更新路由表。

4 实现关键技术

4.1 基于 jms 的消息发送和路由发现机制

分布式企业服务总线基于 JMS 实现跨节点的消息转发和全局路由表的维护。我们将 jms 上的消息队列分为路由同步队列和消息传输队列两部分。图 4 是服务器上队列结构的示意图。

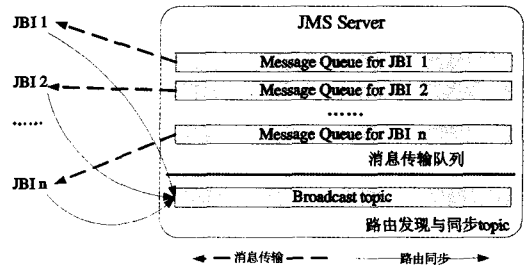

图 4 JMS 通信队列示意图

系统初始化的时候在 Activate MQ 服务器上创建名为 Broadcast 的消息 Topic。每个 JBI 启动后会在 JMS 服务器上创建属于自己 NMR 的消息队列用于消息传输。Broadcast Topic 用于全局消息路由表的形成和维护。所有 JBI 节点都通过 Broadcast Topic 发布路由变更及节点变更的信息。每个 JBI 节点都会监听 Broadcast topic 并根据该 topic 中的信息有路由变更的信息时更新本地路由表。在实现上我们选择开源 JMS 产品 Active MQ 作为通信总线。

4.2 基于事件驱动的路由信息同步机制

我们基于事件驱动的机制来实现各个节点上的消息路由更新和同步。在每个节点内部，事件以对象的形式存在。所有的事件类都继承于 Event 对象，并且按照各自的类别进行派生。关心某类事件的对象在容器中注册自己所关心的事件类。当某事件产生时，描述该事件的对象首先被产生，然后触发该事件的对象调用系统中的 FireEvent 函数。事件处理函数根据系统中监听该事件的对象列表依次调过这些对象的事件处理函数来响应这个事件。

当某个节点消息路由信息发生变更时，首先在该节点内部产生一个 endpoint 变更事件对象，并触发一个路由变更事件。该节点上的系统事件处理函数依次调用监听该事件的对象的事件处理函数进行响应。然后，该事件对象会被序列化，并发送至 JMS 服务器的 Broadcast Topic。其他节点接收到该消息后，首先将该消息还原为一个事件对象，并根据该对象的信息更新全局的消息路由表。同样，节点的加入和失效消息也采用类似的方法进行处理。

4.3 基于随机等待-失效消息广播的节点失效处理机制

在本文提出的分布式 ESB 架构中每个节点采用超时等待的方法来发现其他节点的失效问题。当某节点失效时，其他节点都能够通过超时等待的方法发现该节点的失效问题。如果所有节点都对该次失效进行广播，则存在失效消息泛滥的问题。我们引入了随机等待-失效消息广播的机制来防止节点失效消息的泛滥。具体方法是：当节点 N 通过超时等待的方法发现节点 I 失效时，节点 N 首先随机等待一段时间，如果在等待时间内收到了其他节点广播的节点 I 失效消息，则节点 N 放弃对该次失效消息的广播。否则，节点 N 在等待时间结束后广播节点 I 失效消息。在实验的数据表明，采用这种机制，在能获得较好的失效消息响应时间的同时能够很好地抑制失效消息的泛滥。

5 试验结果

基于以上设计思想，我们基于 JBI 规范对分布式企业服务总线做了原型实现。在该平台上我们进行了如下实验。

实验 1 我们取 20 个在单 JBI 环境下编译和调试通过的应用程序。对于其中的每一个应用程序，我们让构成该应用的多个组件随机分布在不同的 Peer Server 节点上，部署完成后，启动程序运行。试验表明，所有在单 JBI 环境中的应用程

序都能不做修改地在分布式环境下正常运行。这说明:第一,我们设计的DNMR能够为应用程序提供符合JBI规范的访问接口。第二:分布式DNMR确实屏蔽了组件物理位置的差异,实现了跨节点的消息路由和消息传输。

实验2 我们将第三方的BPEL引擎进行包装,使之成为JBI环境中的一个标准的组件,用BPEL文件描述个各服务之间的组合方式。当流程任务部署完成后,启动流流程运行,在某个JBI节点故障的情况下,只要总线内部有同名的备选服务,消息仍然能够被路由至提供该服务的运行组件。这说明系统的可用性得到了提高。

实验3 为了考察消息路由表的内存开销,我们按照实验2的方法,根据某数字园区的多个企业之间应用集成需求构建了100个流程任务。参与流程的组件个数从2个到100个不等。我们逐一将流程任务加载到分布式企业服务总线的环境中运行,并考察每个节点上路由表的内存开销。试验表明,可以认为消息传输路由表的大小随流程应用的任务数的增加成线形增长。当100个任务加载完毕后,路由表的内存开销约在800k左右。在实际应用中,这种内存开销是可以被企业用户接受的。

实验4 我们采用单位时间内消息路由表变化的项数来衡量由于消息路由表的变化而带来的网络资源的开销。仍采用试验3中的实验用例进行实验。实验表明:企业内部和跨企业之间的应用集成。路由表变更带来的网络开销主要是由JBI节点数目的变更、新流程任务的加载或者节点故障引起的。我们考察了8个企业的实际应用,发现在实际应用中以上三项变动都是很少的,因此由于消息路由表带来的网络资源开销可以忽略不计。

实验5 我们对随机等待-失效消息广播的机制进行了模拟实验。节点失效的响应时间定义为从某节点失效到第一个失效消息被发送到JMS广播队列的时间。节点失效的成本定义为一个节点失效后系统产生失效消息的个数。在泛洪机制中节点失效响应时间为JMS Topic的广播时间,在有n个节点的情况下,一个节点失效的广播消息个数为$n-1$。采用随机等待-失效消息广播的机制后,我们考察失效响应时间相对于泛洪机制的增量并考察失效消息的个数。模拟实验的结果如图5、6。从图5可以看出,随机等待-失效消息广播的机制的节点失效响应时间总是大于等于泛洪机制的响应时间,但响应时间的增量是很小的,并且随着节点数目的增多,响应时间有收敛的趋势,逐步逼近泛洪机制的响应时间。从图6可以看出,对于随机等待-失效消息广播的机制,随着节点数目的增多,节点失效成本在增加,但远远小于泛洪机制的开销。这说明随机等待-失效消息广播的机制在保持良好的响应时间的同时有效地降低了失效的开销。

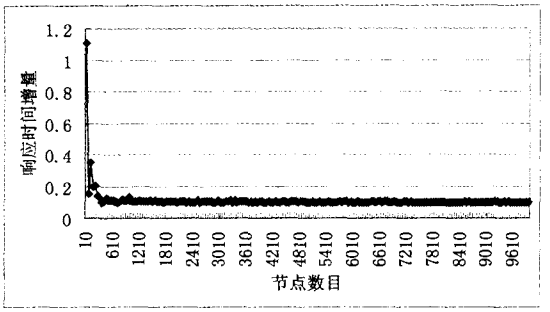

图5 失效消息响应时间图

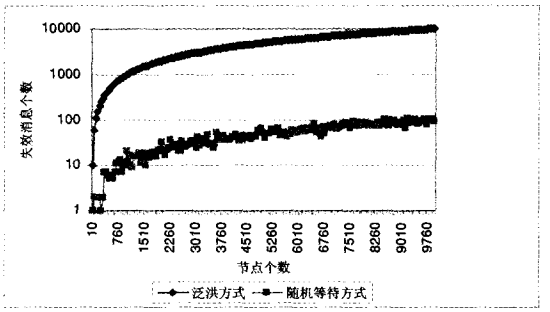

图6 两种失效机制产生失效消息个数试验结果

结论 基于JMS的分布式企业服务总线在满足JBI规范的基础上对企业服务总线的能力进行了扩充,突破了单JBI环境中的总线负载的性能瓶颈,实现了消息传输的持久性,克服了单点故障,提高了系统的可用性。充分利用了JMS发布订阅模式,在不依赖全局消息路由器的情况下实现了全局范围内的消息路由同步和消息转发。实验表明,我们提出的分布式企业服务总线的设计方案有良好的应用效果。

参考文献

1 Cherbakov L, Galambos G, Harishankar R, et al. Impact of Service Orientation at the Business Level [J]. IBM Systems Journal, 2005, 44(4)

2 Schmidt M-T, Hutchison B, Lambros P, et al. The Enterprise Service Bus: Making Service-oriented Architecture real [J]. IBM Systems Journal, 2005, 44(4): 781～797

3 Sun Microsystems Inc. Java Business Integration (JBI)1.0, 2005

(上接第113页)

由上表数据,SOM与PowerQuest相比,速度优势明显,占用系统资源少,且迁移时不影响正常的生产活动,因此在性能与功能方面,均占有优势。该系统目前已应用于中科院计算所工程中心蓝鲸服务部署系统,为生产前端应用服务器(AS)提供容灾服务,达到了预期效果。

结束语 设计并实现了服务在线迁移系统,为Windows生产中心提供容灾。SOM能够对包括系统数据与业务数据在内的系统服务进行完备的备份,灾难发生后能实时恢复生产环境与应用数据,HRF机制保证了迁移状态的一致性以及与生产中心生产活动的独立性。经检验在性能、兼容性等方面都达到了预期效果。

参考文献

1 Barkley P. Disaster-recovery Plans Focusing on Business Continuity [Z]. E-Commerce Times. http://www.ecommercetimes.com/story/35 993.html, 2 004-08-23

2 Ma Yili, Fu Xianglin, Han Xiaoming, et al. The Separation Between Storage and Computation. Journal of Computer Research and Development, 2005, 42(3)

3 Jiang X, Xu D. SODA: A service-on-demand architecture for application service hosting utility platforms. In: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003. 174～183

4 Lu Chenyang, Alvarez G A. Aqueduct: Online Data Migration with Performance Guarantees. In: USENIX Conference on File and Storage Technologies, Monterey, CA, 2002-01. 219～230

5 Solomon D A, Russinovich M E. Inside Microsoft Windows 2000, 3rd edition

6 Microsoft®Windows® XP Driver Development Kit (DDK)