企业级 Java Web 服务的研究与实现*

王芝虎，葛 声，张力军
(北京航空航天大学 计算机学院，北京 100083)

摘 要：企业级 Java Web 服务是一种基于 Java 技术实现的企业级分布式组件技术，解决了 Web 服务访问过程中的状态保持问题，基于 HTTP 和 JMS 协议支持对 Web 服务的多种调用方式，采用灵活的配置获得多种中间件服务的支持，并利用 Web 服务的包容性实现了对多种异构遗留系统的集成，从而为将 Web 服务应用到企业级的环境中提供了足够的保障。利用一个企业级 Java Web 服务组件模型，并基于此实现了一个运行时系统，系统由客户端、消息中间件和运行时容器三部分组成，支持业务客户端与服务端企业级 Java Web 服务的运行交互过程。

关键词：Web 服务；企业级 Java Web 服务；对话；会话
中图法分类号：TP393.09 **文献标识码**：A **文章编号**：1001-3695(2005)01-0128-06

Research and Implementation of Enterprise Level Java Web Service

WANG Zhi-hu, GE Sheng, ZHANG Li-jun
(School of Computer Science, Beihang University, Beijing 100083, China)

Abstract: The Enterprise Level Java Web Service is an enterprise distributed component technique based on Java, which solves the status persistent problem during the process of accessing Web services. It supports many invoking method based on HTTP and JMS protocol and adapts flexible configurations to support various middle-wares. In addition, it can integrate many old systems with different architectures by compatibility of the Web service so that Web services are ensured to be applied into enterprise level environments. Use defines a component model, and implements a runtime system, which is made of client, message oriented middleware and runtime container.

Key words: Web Service; Enterprise Level Java Web Service; Conversation; Session

1 引言

十多年来，软件开发方法从早期的面向模块化设计、面向对象设计发展到基于分布式对象技术的组件化设计，计算模式也从主机计算、基于客户/服务器的分布式计算发展到基于客户/互联网的网络计算。但传统的企业级分布式组件技术(如 J2EE, CORBA, COM+)规范体系相对独立，将信息分割成了互联网环境下的一个个“孤岛”。随着互联网应用的深入普及，松散耦合环境下有效的资源共享成为一个重要的技术问题和应用的核心难点。Web Services 技术应运而生，体现了“软件变服务”的思想，将互联网下的资源封装为一个个 Web Service，实现了资源的有效共享。根据 W3C 的定义，Web Service 是一种通过 URI 标志的软件应用，其接口及绑定形式可以通过 XML 标准定义、描述和检索，并能通过 XML 消息及互联网协议完成与其他应用的直接交互[2]。从其概念，我们在广义上可将其理解为一种通过 Web 进行数据及功能共享的面向服务技术；而在狭义上将其理解为一种通过 Web 及标准接口进行调用的分布式组件，类似 EJB, CORBA 或 COM 组件。

收稿日期：2004-01-16；修返日期：2004-03-24
基金项目：国家“863”计划资助项目(2001AA113030, 2001AA115110, 2001AA414020)

近年来，Web Services 技术发展迅速，已经定义了一系列基于 XML 描述的国际标准，如简单对象访问协议(Simple Object Access Protocol, SOAP)[3~5]统一了 Web Service 的调用方式；Web 服务描述语言(Web Services Description Language, WSDL)[6]对 Web Service 接口进行统一描述；统一描述、发现和集成协议(Universal Description, Discovery and Integration, UDDI)[7]定义了一套 Web Service 的发布、查询机制，从而构建了一个面向服务的体系结构(Service Oriented Architecture, SOA)，实现跨越互联网的数据和功能的共享。同时，Web Service 作为一种企业级分布式组件技术，还显得不够成熟，目前主要是一种基于 HTTP 协议进行同步请求/响应调用的无状态的简单应用。企业级的应用要求 Web Service 支持一对一/多、同/异步有响应、无响应的多种调用方式；支持访问过程中的状态保持；支持对多种异构遗留系统的集成。因此定义一个精确的企业级 Web Service 组件模型和建立一个支持上述模型的运行时系统，以满足上述企业级应用的需求，将是决定 Web Services 能否最终成为一项现实可用技术的关键。

本文首先从分析 Web Services 技术和传统的企业级分布式组件技术入手，给出了企业级 Java Web 服务(Enterprise Level Java Web Service, EJWS)的定义；然后定义了一个 EJWS 组件模型，对 EJWS 的静态结构和处理模式进行详细刻画和约束；并在此基础上设计实现了一个企业级 Java Web 服务运行时系统(Enterprise Level Java Web Service-RunTime System,

EJWS-RTS), 系统由客户端、消息中间件和运行时容器三部分组成, 支持业务客户端与服务端 EJWS 的运行交互过程; 最后描述了一种该系统的典型应用场景: 企业域内基于面向消息中间件(Message Oriented Middleware, MOM) 的 SOA 集成应用。

2　企业级 Java Web 服务

2.1　Web Services 协议栈分析

在互联网环境下, 采用 Web Services 技术将资源进行封装到面向终端用户的应用, 需要自底向上的经历四个层次: 资源层、Web Service 层、Web Service 组合层、Web Service 展示层。①资源层表现为互联网下的各种异构遗留系统; ②Web Service 层负责将这些异构系统封装为具有一定业务功能的 Web Service; ③组合层将各个 Web Service 组合为满足一定业务流程的应用程序, 同时该应用程序亦表现为一个 Web Service; ④在展示层进行 Web Service 的功能表示, 实现与多种终端用户的交互。此外, 一个企业级分布式应用系统还需获得事务、安全、质量保证等中间件服务的支持, 中间件服务贯穿了上述四个层次, 自顶向下逐层映射。企业级 Java Web 服务是 Web Service 层的具体实现, 既向上支持 Web Service 的组合与展示, 又向下包容多种异构遗留系统, 同时调用对应层次的中间件服务以获得事务、安全、质量保证的支持。

2.2　传统的企业级分布式组件技术分析

传统的企业级分布式组件模型主要有 SUN 公司发布的 EJB(Enterprise JavaBeans, 企业级 JavaBeans) 规范[8], OMG 随 CORBA3.0 规范发布的 CCM(CORBA Component Model, CORBA 组件模型)[9] 及微软公司发布的 MTS/COM + 模型[10]。

(1) EJB2.0 规范定义了三种组件, 分别为对话型组件、实体型组件和消息驱动型组件。①对话型组件(SessionBean) : 封装了商业过程的处理逻辑, 并进一步区分定义为有/无状态两种对话型组件(Stateful/Stateless SessionBean), 分别支持有/无状态的访问过程。②实体型组件(EntityBean) : 将商业处理过程中的商业数据单独封装, 并采用 BMP(Bean-Managed Persistence, Bean 管理持久化) 或 CMP(Container-Managed Persistence, 容器管理持久化) 方法将处于内存临时态的商业数据自动持久化到数据库中。③消息驱动型组件(Message-Driven-Bean, MDB) : 由于前两种组件规定采用 RMI-IIOP 协议进行远程调用, 而 RMI - IIOP 协议仅是一种 RPC 型协议, 因此 EJB2.0 规范引入了 MDB, 由其扮演一个消息侦听者的角色, 负责侦听 JMS 消息, 然后或者自己作商业逻辑处理, 或者将消息转发到相应的对话型组件, 由其完成商业逻辑处理。

(2) CORBA/CCM 规范是 EJB 规范语言无关的超集, 它分为两部分: 一部分为核心规范, 与 EJB 规范等同; 一部分为扩展规范, 语言无关, 实现了对多种面向对象语言的支持, 并对组件的组合机制进行了规定。

(3) MTS/COM + 是集成在 Windows 操作系统中的产品。它定义了无状态的对话型组件; 商业数据封装于对话型组件内部, 而不是单独封装为一个组件, 采用手动编程(如 ODBC) 的方式实现数据从内存态到数据库的持久化。

对于企业级 Java Web 服务来说: ①Web Service 作为一种面向服务的应用方式, 必然是一种对话型组件, 它封装了商业过程的处理逻辑。②同样由于面向服务的应用方式, Web Service 不应向其调用者暴露内部数据; 而在实现上, 实体型组件往往是一种细粒度设计, Web Service 需通过 SOAP 进行调用, SOAP 调用的低效率加上细粒度的组件设计将导致整个系统的效率过低。因此 EJWS 未引入实体型组件, 而采用手动编程的方式实现商业数据的持久化。③同时, SOAP 采用 XML 来统一数据描述格式, 动态绑定于多种传输协议(如 TCP/IP, HTTP 和 JMS) 之上, 可在容器中设置一个专门的请求接收层侦听绑定于多种传输协议上的 SOAP 报文, 因此消息驱动型组件对 EJWS 没有意义。

2.3　企业级 Java Web 服务

综合上述, 我们把企业级 Java Web 服务定位为一种对话型组件, 具有如下特征: ①EJWS 封装了商业逻辑和商业数据, 通过采用 WSDL 协议进行统一接口描述以对外暴露商业逻辑的商业方法; 采用手动编程(如 JDBC) 持久化的方式将商业数据从内存态转换到数据库中。②为了实现对多种异构遗留系统(如 J2EE, CORBA 和 COM + 等) 的集成, EJWS 需封装异构遗留系统的客户端, 而此时对应的商业逻辑和商业数据由异构系统封装, 商业逻辑对外暴露的商业方法仍通过 EJWS 的 WSDL 文档进行描述发布。③EJWS 通过建立运行时容器的独立请求侦听层, 可接收来自业务客户端的基于多种传输协议(如 HTTP, JMS 协议) 的调用。④通过调用中间件服务, EJWS 获得事务、安全、质量保证等支持。⑤EJWS 支持业务客户端与服务端 EJWS 之间的有/无状态对话过程。⑥EJWS 采用 Java 技术实现。

3　企业级 Java Web 服务的组件模型

我们将企业级 Java Web 服务的组件模型定义为一个二元组 < Structure, Operation >, 分别从组件的静态结构和处理模式的角度, 对 EJWS 的对话型组件进行刻画和约束。

3.1　企业级 Java Web 服务的结构(Structure)

企业级 Java Web 服务的结构在物理上表现为一个. WSAR 压缩文件包, 包括的信息文件有: ①作为 EJWS 具体实现的 Java 类; ②相关的辅助 Java 类库; ③描述商业操作接口、访问形式的 WSDL 文档; ④描述 EJWS 功能实现的相关特性和处理需求的部署描述符; ⑤各种相关的资源文件, 如文本、图像。作为 EJWS 具体实现的 Java 类是我们关注的重点, 根据使用情况的不同, 表现为两种实现形式:

(1) 当 EJWS 封装了商业逻辑和商业数据时, EJWS Java 类表现为满足 EnterpriseWebService 接口的 Java 类。EnterpriseWebService 接口描述如下:

```
Public interface Enterprise WebService extends java.io.Serializable{
   public WebServiceContext getContext( ) ; //获取实例上下文
   public void setContext( WebServiceContext ctx) ; //设置实例上下文
   public void wsPassivate( ) ; //实例钝化前调用
   public void wsActivate( ) ; //实例激活后调用
}
```

EnterpriseWebService 接口表明基于其实现的 Java 类是可序列化的, 并且它定义了两类四种操作以对 EJWS 实例的设置/获取上下文、换入/换出机制进行支持: ①实例上下文

(WebServiceContext) 包含运行时环境信息, EJWS 实例可通过其实现对容器的回调功能。在 EJWS 实例的创建阶段, 容器调用 setContext(WebServiceContext ctx) 方法, 将上下文传递给实例; 此后 EJWS 实例可通过调用 getContext( ) 方法获得该上下文。②一个封装了商业逻辑的 EJWS 实例往往会占用许多系统资源, 同时一个多用户的有状态对话过程需要缓存大量 EJWS 实例, 考虑到系统资源有限, 因此 EJWS 实例需要在内存态和持久态之间换入/换出。从内存换出之前, 容器将调用 wsPassivate( ) 方法关闭无需持久化或无法序列化的资源; 而在换入之后, 容器将调用 wsActivate ( ) 方法重新获得上述关闭的资源。

除上述 EnterpriseWebService 接口定制的操作外, EJWS Java 类还包括 EJWS 开发者自由定义的商业操作: 由 EJWS Java 类自身实现具体商业逻辑; 并采用 WSDL 描述发布出, 以供 EJWS 客户端调用。

(2) 当采用 EJWS 来集成各种遗留异构系统(如 J2EE, CORBA, COM +) 时, 商业逻辑和商业数据由遗留系统封装, EJWS Java 类仅仅是对应遗留系统的 Java 客户端程序。它无须回调容器, 所占用的资源在换入/换出时也无须关闭/重建, 因此不必支持 EnterpriseWebService 接口所定义的四种操作; 商业逻辑对外暴露的商业方法仍通过 EJWS WSDL 文档进行描述发布。

3.2 企业级 Java Web 服务的处理模式(Operation)

企业级 Java Web 服务是一种对话型组件, 要求支持有/无状态的对话过程, 同时对话过程是一个业务客户端与服务端 EJWS 发生一次或多次调用的过程。EJWS 的处理模式主要体现为 EJWS 对话过程和调用过程的处理方式。

3.2.1 企业级 Java Web 服务的对话过程

对话过程是对话型组件提供业务功能, 供其客户端调用以完成一定商业处理目的的过程。在这个过程中, 客户端与服务端组件将会发生一次或多次调用, 每次调用后, 对话状态发生改变, 由本次调用的初始对话状态转换到本次调用的结果对话状态。由于本文所描述的 EJWS 对话过程只涉及双方, 对话状态可由参与对话的业务客户端程序和服务端 EJWS 实例来保存, 即服务端 EJWS 所具有的对话状态等价于该 EJWS 实例的状态。因此, 对话过程中的状态保持等价于对服务端 EJWS 实例状态的保持。

(1) 无状态的对话过程无状态的对话过程不需保存对话状态, 那么服务端 EJWS 的实例状态也不需保持, 每次调用请求到来时, 服务端容器均新创建一个 EJWS 实例, 在其上执行调用。因此一个具有 N 次调用的无状态对话过程, 可以分解为一个客户端程序与任意 N 个具有创建时初始化状态的 EJWS 实例分别进行一次调用。相应的 EJWS 实例的生命周期仅持续一次调用过程。

(2) 有状态的对话过程。有状态的对话过程需要保存对话状态, 那么服务端 EJWS 的实例状态也需要保持。当第一次调用请求到来时, 服务端容器新创建一个 EJWS 实例, 在其上执行调用, 然后将其缓存保持; 此后每次调用都将在该缓存实例上被执行。一个具有 N 次调用的有状态对话过程, 只能是一个客户端程序与一个被缓存的 EJWS 实例进行 N 次调用的过程, 该 EJWS 实例的生命周期持续整个有状态对话过程。

基于此, 我们采用了如下机制来保障一个有状态的对话过程: ①将一个有状态的对话过程划分为三个不同的阶段(开始阶段、持续阶段和结束阶段) , 采用扩展的 SOAP 报文头 "ConversationState"属性来表示。②利用会话机制的全局唯一值将对话相关的客户端程序和服务端 EJWS 实例一一关联起来, 扩展的 SOAP 报文头 "SessionToken"属性表示该值; 并通过会话机制的超时控制能力对对话过程的持续时间进行控制, 超过一定时限时, 强制结束本次对话过程。③引入了缓存池机制做 EJWS 实例缓存, 一个缓存池缓存一组同一类型的 EJWS 实例, 实例之间采用 SessionToken 唯一标志区分。考虑到系统资源有限, 缓存池不宜过大, 因此使用了最近最少使用算法(Least Recently Used) 对 EJWS 实例进行优先级排序, 当所需缓存的 EJWS 实例数量超过缓存池上限时, 低级别的 EJWS 实例将从缓冲池中置换出, 序列化到持久存储介质中; 而在需要对其调用时, 从持久存储介质中重新反序列化置入缓冲池中。

3.2.2 企业级 Java Web 服务的调用过程

一个业务客户端与服务端 EJWS 的一次调用过程, 依次经历了三个行为阶段: 客户端与服务端之间的调用、服务端对中间件服务的调用和服务端对 EJWS 的调用。

(1) 客户端与服务端之间的调用。在客户端与服务端之间的一次调用过程中, 客户端首先向服务端发送请求信息, 根据实际需求的不同, 服务端在完成请求处理之后, 可能需要返回响应信息; 也可能不返回响应信息。可将前者称为有响应的调用, 后者称为无响应的调用。而对于有响应的调用, 客户端发送请求信息后, 在接收到响应信息之前, 可能会阻塞以等待响应信息对其处理; 也可能不阻塞, 先处理其他业务, 当响应信息到来时再转而对其处理。可将前者称为同步有响应的调用, 后者称为异步有响应的调用。此外, 一个客户端发送的一条请求信息可能会只发送到一个服务端; 也可能会被发送到多个服务端。此时前者称为一对一的调用, 后者称为一对多的调用。根据上述分析, 客户端与服务端之间的调用类型可细分如下: ①一对一的同步有响应的调用; ②一对多的同步有响应的调用; ③一对一的异步有响应的调用; ④一对多的异步有响应的调用; ⑤一对一的无响应的调用; ⑥一对多的无响应的调用。

目前主流的远程调用机制为 RPC(Remote Procedure Call, 远程过程调用) 和消息。RPC 机制是一种客户/服务端的同步阻塞调用, 仅允许一个客户端和一个服务端对话, 只支持请求/应答模式; 消息机制在客户与服务端之间加入了一个消息中间件层, 它从一个或多个消息生产者接收消息, 然后将这些消息转发到一个或多个消息消费者, 从而实现多个客户端和多个服务端之间的调用, 它支持发布/订阅、点到点和请求/应答模式。RPC 机制可以实现上述除一对多的调用外的所有调用类型; 消息机制可以实现上述所有调用类型。

企业级 Java Web 服务通过 SOAP 进行调用, SOAP 采用 XML 来统一数据描述格式, 可动态绑定于多种传输协议(如 TCP/IP, HTTP 和 JMS) 之上。HTTP 协议是一种 RPC 型协议, JMS 协议是一种消息型协议。EJWS 通过将 SOAP 绑定于 HTTP 和 JMS 协议之上, 可实现对上述多种调用类型的支持。同时, 考虑到效率因素, EJWS 采用 RPC 型的 HTTP 协议实现一

对一同步有响应的调用; 采用消息型的 JMS 协议实现其余的调用类型。

为此, 我们采用了请求侦听器机制, 基于 Servlet 和 MDB 分别实现了 HTTP 侦听器和 JMS 侦听器。在实现上, 客户端对 Servlet 的 HTTP 调用直接表现为一对一同步有响应的调用; 而对 MDB 的 JMS 调用, 我们通过如下机制来实现对其余多种调用类型的支持: ①服务端 MDB 同时扮演请求消息消费者和响应消息生产者的角色; 客户端由 JMS 生产者和 JMS 消费者分别扮演请求消息生产者和响应消息消费者的角色。消息生产者将消息发送到消息中间件层的队列/主题中; 消息消费者负责侦听某个特定队列/主题, 从中取出所需的消息。② 采用 JMS 消息 "ReplyTo"属性以指定调用是否有响应: 值为空时, 表示不需响应; 值为某个特定队列/主题名时, 表示需响应。③有响应时, 通过客户端在 JMS 生产者发送请求消息后是否阻塞等待 JMS 消费者处理响应消息, 以实现调用的同/异步。④当消息发送到队列时, 调用为一对一; 发送到主题时, 调用为一对多。

(2) 服务端对中间件服务的调用。服务端调用中间件服务以获得事务、安全、质量保证等支持。服务端对中间件服务的调用一般采用两种方式: ①显式, 即通过 EJWS 直接对中间件服务 API 进行手动编程, 导致商业逻辑和调用中间件服务 API 的逻辑交织在一起, 难以编码、维护和支持; ②隐式, 即引入请求拦截器机制, EJWS 只包含商业逻辑, 由服务端容器负责创建一个请求拦截对象, 该对象将拦截来自客户端的请求, 并根据配置信息调用所需的中间件服务和 EJWS 适配器以完成一次调用。这种方式易于编码、维护和支持, 被目前主流分布式系统(J2EE, CORBA 等) 所采用, EJWS 也将采用此种方式。

为此, 我们引入了如下机制来实现上述拦截/调用过程: ①通过将中间件服务和 EJWS 适配器暴露以相同的调用接口, 从而采用无差别的链式结构将它们组织起来; ②一个请求拦截对象对应一种 EJWS, 反映了该种 EJWS 的部署描述符所要求的功能特性和处理需求, 对链式结构进行调用; ③对同一类型 EJWS 的请求可使用同一请求拦截对象, 引入缓存池机制做请求拦截对象缓存, 并采用最近最少使用算法(Least Recently Used) 对其进行置换, 超过缓存池上限的请求拦截对象被丢弃。

(3) 服务端对 EJWS 的调用。服务端通过调用 EJWS 实例执行所请求的商业方法。由于 EJWS 的商业逻辑采用多种实现形式, 因此我们引入了适配器机制, 根据不同的实现形式, 创建相应的 EJWS 实例, 执行商业方法, 生成 SOAP 响应报文。在本课题中, EJWS 商业逻辑的实现形式分别采用了 Java 类、EJB、CORBA、COM 组件, 当实现形式为 Java 类时, EJWS 实例表现为该 Java 类的实例; 而当实现形式为 EJB/CORBA/COM 组件时, EJWS 实例表现为对应远端对象的引用。

因此, 对应 EJWS 实例的创建过程分别如下: ① Java 类——直接加载 Java 实现类; ② EJB 组件——通过 JNDI 查找 EJB 主接口, 然后创建 EJB 远程对象, 获得其引用; ③ CORBA 组件——通过 CORBA 命名服务器中查找获得 CORBA 对象的引用; ④COM 组件——在 Windows 注册表中查找获得 COM 组件的引用。

当执行商业方法时: ①实现形式为 Java 类, 直接调用该 Java 类实例对应方法; ②实现形式为 EJB/CORBA/COM 组件时, 将 SOAP 请求信息转换为分别符合 EJB/CORBA/COM 规范的请求, 并向远端对象发出调用, 在其上执行商业方法。

4　企业级 Java Web 服务运行时系统的设计与实现

一个 EJWS 运行时系统的体系结构由 EJWS 调用过程中的三要素: 客户端、消息中间件和 EJWS 运行时容器/服务器(EJWS-RT) 共同组成, 涵盖了 EJWS 调用的请求和响应过程所需的各种功能模块, 如图 1 所示。

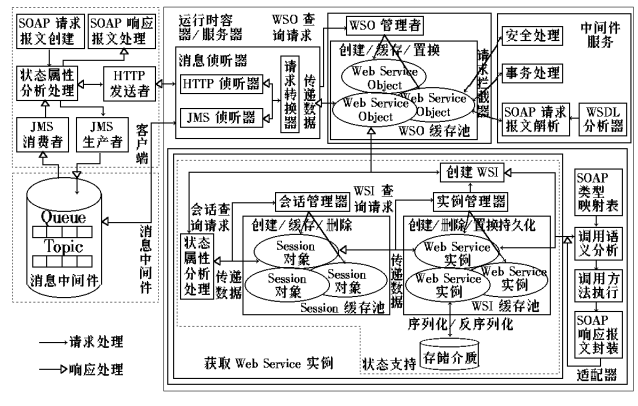

图 1　EJWS 运行时系统的体系结构

(1) 客户端: 作为一次调用过程的请求发起者, 负责 SOAP 请求报文的创建和发送; 当本次调用为有响应的请求时, 客户端还需负责接收和处理响应消息。

①SOAP 请求报文创建和 SOAP 响应报文处理: 分别负责本次调用涉及的 SOAP 请求报文的创建和 SOAP 响应报文的解析。

②状态属性分析处理: 当调用为一个有状态对话过程中的某次交互时, SOAP 报文需在发送之前和接收之后进行状态属性分析处理。状态属性为扩展的 SOAP Header 属性, 包括:

- "ConversationState"取值可为 "Start", "Continue", "Finish", 分别表示对话过程的开始、持续和结束阶段。
- "SessionToken"一个全局唯一值, 将对话双方一一关联起来。

③HTTP 发送者: 当传输协议为 RPC 型的 HTTP 协议时, 负责将 SOAP 请求报文绑定于其上, 并发送到指定的 EJWS 访问点; 之后线程阻塞, 直到接收到对应响应消息, 从中剥离出 SOAP 响应报文。从而支持一对一同步有响应的请求。

④JMS 生产者和 JMS 消费者: 当传输协议为消息型的 JMS 协议时, 分别扮演请求消息生产者和响应消息消费者的角色。JMS 生产者将 SOAP 请求报文绑定到 JMS 协议, 发送到欲访问 EJWS 指定的队列/主题。当 JMS 生产者未指定该调用的响应队列/主题时, 调用为无响应的调用; 指定时, JMS 消费者负责侦听该队列/主题, 获取本次调用的响应消息, 从中剥离出 SOAP 响应报文。在 JMS 生产者和 JMS 消费者之间线程可阻塞或不阻塞, 从而可支持同/异步有响应的请求。

(2) 消息中间件: 传输协议为消息型的 JMS 协议时, 客户端与 EJWS-RT 调用的关键环节, 负责接收来自消息生产者的 JMS 消息, 存储于队列/主题中, 然后转发给相应的消息消费

者。

①队列(Queue) : 每一条消息只转发给侦听本队列且符合指定条件的某个消息消费者, 从而支持一对一请求。

②主题(Topic) : 每一条消息转发给所有侦听本主题且符合指定条件的消息消费者, 从而支持一对多请求。

(3) EJWS 运行时容器/服务器: EJWS 寄宿的运行时环境。

①消息侦听器: 作为一个独立的消息接收层, 接收剥离出绑定于多种传输协议上的 SOAP 请求报文, 并对其进行转换和封装, 以对请求拦截器屏蔽访问请求格式的差异性。

a) HTTP 侦听器: 侦听来自 HTTP 发送者的本 EJWS 的请求消息, 剥离出 SOAP 请求报文, 传递给请求转换器; 之后等待 SOAP 响应报文返回, 并将其绑定发送回 HTTP 发送者。

b) JMS 侦听器: 扮演了 JMS 请求消息消费者和响应消息生产者的角色。首先从本 EJWS 所指定的侦听队列/主题中获取针对本 EJWS 的所有请求消息, 分离出 SOAP 请求报文, 传递给请求转换器; 通过读取 “ReplyTo ”属性以判断有无响应要求, 无响应要求时, 不返回 SOAP 响应报文; 有响应要求时, 则等待 SOAP 响应报文返回, 然后将其绑定于 JMS 协议, 设置 “CorrelationID ”属性为请求消息指定标志值, 发送回请求消息 “ReplyTo ”属性所指定的队列/主题。

c) 请求转换器: 充当消息侦听器和请求拦截器之间的调用接口, 将 HTTP/ JMS 侦听器传递来的 SOAP 请求报文以请求拦截器指定的数据格式进行封装。

②请求拦截器: 拦截来自消息侦听器的请求, 调用该请求指定 EJWS 的部署描述信息所要求的中间件服务和适配器。

a) WSO 管理器: 以 WSO 为管理单元, 负责 WSO 的创建和存储管理。为了避免频繁访问同一类 EJWS 时创建大量的 WSO 实例, 提供了 WSO 缓存池, 并采用最近最少使用算法(Least Recently Used) 以 URI 作为唯一标志对 WSO 进行调度/置换。同时本管理器负责处理来自请求转换器的 WSO 查询请求, 返回其对应 WSO 的引用。

b) WSO(Web Service Object) : WSO 管理器在运行时读取 EJWS 部署描述符所动态生成的可控代理对象, 与 EJWS 是一一对应关系, 通过自身的相关属性和调用中间件服务及适配器, 以使部署描述符所描述的 EJWS 功能特性和处理需求在运行时得到实现。

③中间件服务: 对 EJWS 的调用过程提供诸如安全、事务等共性支持, 采用统一的调用接口供 WSO 循环链式调用。

a) 安全服务: 对 SOAP 报文的加/解密、访问控制等安全问题提供支持。

b) 事务服务: 提供对访问过程的事务支持。

c) SOAP 请求报文解析: 采用 Java 对象描述 SOAP 请求报文中的各个元素, 分析出该次调用的操作名及参数, 并通过 WSDL 分析器获得调用操作的类型和参数的类型、个数以及返回值的信息, 供后续操作使用。

④适配器: 适配多种实现形式的 EJWS 实例, 完成商业调用处理, 生成 SOAP 响应报文, 并支持有/无状态的对话过程。

a) 获取 Web Service 实例: 当对话过程无状态时, 直接调用 WSI 创建模块新创建一个 WSI; 有状态时, 需获得状态支持模块的支持。

● 状态支持模块

状态属性分析处理: 对请求消息和响应消息的状态属性进行分析处理。

会话(Session) 管理器: 以会话对象为管理单元, 负责会话对象的创建、缓存和删除; 同时负责处理状态属性分析处理模块的会话对象查询请求, 返回对应会话对象的引用。

会话对象: 提供全局唯一的 SessionToken 将对话双方一一关联, 并对有状态对话过程的时间期限进行控制。

WSI 管理器: 以 WSI 为管理单元, 负责 WSI 的创建、删除和存储管理。通过 WSI 创建模块创建 WSI, 采用 WSI 缓存池缓存一组同一类型 EJWS 的不同 WSI, WSI 之间以 SessionToken 唯一标志。为避免缓存的 WSI 过多占用太多系统资源, 采用最近最少使用算法(Least Recently Used) 对 WSI 进行置换, 超过上限的 WSI 以序列化/反序列化的方式换出/换入。同时本管理器负责处理会话对象的 WSI 查询请求, 返回其对应 WSI 的引用。

WSI(Web Service Instance) : EJWS 的运行时实例, 与其对应的 WSO 是多对一关系。同时 WSO 采用多线程实现, 保证了 EJWS 以单线程实现, 简化开发。

● WSI 创建模块

基于不同的 EJWS 实现形式采用相应的创建方式创建 WSI。

b) 调用语义分析: 通过 SOAP 类型映射表完成 SOAP 数据类型到 Java 数据类型的转换。

c) 调用方法执行: 利用 Java 反射机制实现对调用方法的执行。

d) SOAP 响应报文封装: 调用方法执行后, 将返回值封装为 SOAP 响应报文。

5 企业级 Java Web 服务运行时系统的典型应用场景

在一个大型企业中, 常常会出现这样一种情况, 各个不同的部门采用了不同的系统来实现本部门的业务运作, 部门之间的通信变成了异构系统之间的通信。目前解决这种异构遗留系统间集成与互操作问题的主流技术是企业应用集成(Enterprise Application Integration, EAI) 技术, 但是它定制复杂, 并且造价昂贵。企业级 Java Web 服务运行时系统(EJWS-RTS) 为解决这一问题提供了一种操作简单, 造价低廉的方法, 如图 2 所示, 我们将其称为企业域内基于 MOM 的 SOA 集成。

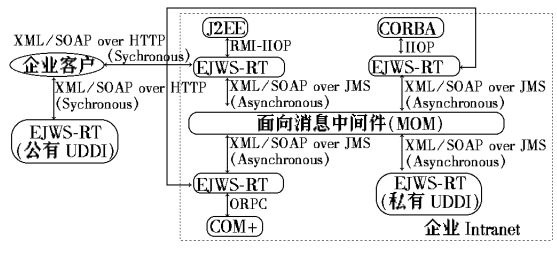

图 2 企业域内基于 MOM 的 SOA 集成

(1) 在企业域内, 各种异构遗留系统(如 J2EE、CORBA 和 COM +) 的调用客户端被封装为 EJWS, 部署于 EJWS - RT 中, 并统一发布到企业域内的一个私有 UDDI 注册中心之中; MOM 作为异构系统间的通信 Bus。当各异构系统之间需要相互调

用时: ①调用者将在私有 UDDI 注册中心中查询被调用者的 EJWS; ②调用者向被调用者发起商业调用。整个通信过程采用 JMS 协议。

(2) 而当企业客户从企业域外访问企业所提供的服务时, 封装产生的 EJWS 将被发布到企业域外的公有 UDDI 注册中心之中。企业客户在向 EJWS 发起调用之前, 需要从公有 UDDI 注册中心中查询被调用的 EJWS, 然后实行商业调用。整个通信过程采用 HTTP 协议。

6　结束语

近年来 Web Service 技术的研发与应用发展很快, 如 .NET, SUN ONE, IBM 的动态电子商务等等, 但这些工作的主体是把 Web Service 作为一种 Web 组件的抽象接口来看待; 企业级 Java Web 服务主要是通过将原有的 Web Service 抽象接口映射成为一种分布式组件实体。同时, 企业级的应用要求 Web Service 支持一对一/多、同/异步有响应、无响应的多种调用方式; 支持访问过程中的状态保持; 支持对多种异构遗留系统的集成。因此本文首先通过定义了一个 EJWS 组件模型, 对 EJWS 的静态结构和处理模式进行详细刻画和约束, 来解决上述问题: EJWS 定位为一种对话型组件, 支持有/无状态的对话过程; 基于 HTTP 和 JMS 协议来支持对 Web 服务的多种调用方式; 利用灵活的配置来获得多种中间件服务的支持; 并通过 Web 服务的包容性实现了对多种异构遗留系统的集成; 然后基于此模型本文设计实现了一个企业级 Java Web 服务运行时系统, 系统由客户端、消息中间件和运行时容器三部分组成, 支持业务客户端与服务端 EJWS 的运行交互过程; 最后描述了一种该系统的典型应用场景: 企业域内基于面向消息中间件的 SOA 集成应用。目前在本系统中, SOAP 报文采用的传输协议只有 HTTP/JMS 协议, 可以进一步扩展为基于 SMTP 及 TCP/UDP 等多种协议; 而 EJWS 商业逻辑的实现形式也可在 Java 类, EJB/CORBA/COM 组件的基础上进行进一步扩展, 从而使本系统的集成适应能力更强。同时, 下一步还将该系统的性能优化和应用推广方面展开工作。

参考文献:

[1] 葛声, 马殿富, 等. 基于 Web 服务的网络软件运行平台研究与实现[J]. 北京航空航天大学学报, 2003, 29(10): 897-900.

[2] Austin D, *et al*. W3C Web Services Architecture Requirements [EB/OL]. http://www.w3.org/TR/2002/WD-wsa-reqs, 2002-04-29.

[3] W3C Group. Simple Object Access Protocol (SOAP) 1.2 Part 0: Primer W3C Working Draft [EB/OL]. http://www.w3.org/TR/2001/WD-soap12-part0, 2001-12-17.

[4] W3C Group. Simple Object Access Protocol (SOAP) 1.2 Part 1: Messaging Framework W3C Working Draft [EB/OL]. http://www.w3.org/TR/2001/WD-soap12-part1, 2001-12-17.

[5] W3C Group. Simple Object Access Protocol (SOAP) 1.2 Part 2: Adjuncts W3C Working Draft[EB/OL]. http://www.w3.org/TR/2001/WD-soap12-part2, 2001-12-17.

[6] W3C Group. Web Services Description Language (WSDL) version 1.2, W3C Working Draft [EB/OL]. http://www.w3.org/TR/2002/WD-wsdl12, 2002-07-09.

[7] UDDI Org. UDDI version 2.04 API Specification UDDI Published Specification [EB/OL]. http://uddi.org/pubs/ProgrammersAPI-v2.04-Published-20020719.pdf.

[8] SUN Microsystems Inc. Enterprise JavaBeans(EJB) [EB/OL]. http://java.sun.com/products/ejb/.

[9] Object Management Group. CORBA/CORBA Component Model[EB/OL]. http://www.omg.org/docs/formal/02-06-71.pdf.

[10] Microsoft Inc. MTS/Component Object Model(COM)[EB/OL]. http://www.microsoft.com/com/default.asp.

[11] Sun Microsystems Inc. Java™ 2 Platform Enterprise Edition Specification v1.4 [EB/OL]. http://java.sun.com/j2ee/1.4/download.html.

作者简介:

王芝虎(1978-), 男, 四川人, 硕士研究生, 主要研究方向为分布式计算、电子商务和企业计算; 葛声(1973-), 男, 安徽人, 博士, 主要研究方向为协同计算、电子商务; 张力军(1971-), 男, 陕西人, 副教授, 博士, 主要研究方向为计算机软件与网络。

(上接第 90 页)类型设计的数据模型、查询机制及其他算法。在功能上和适用范围上与现代其他的数据库管理系统有所区别, 但是在技术上又有着千丝万缕的联系。在这种新生的数据管理系统中, 能够发现如分布式数据库、内存数据库、主动数据库、时态数据库、实时数据库等不同数据库管理系统技术上的体现。

目前, DSMS 技术的研究和系统的开发还处于一个相对无序的状态, 大多数是针对某一特定领域的应用设计和开发的。但从另一个角度来看, 这却比较全面地从不同侧面展示了数据流数据管理系统的强大生命力, 愈发激励着研究人员进一步探索的兴趣。另一方面, DSMS 相关领域的技术研究中又出现了更有前景的方向, 那就是 DSMS 技术和 XML 技术的结合[4]。XML 技术的引入, 使得应用系统间进行复杂的实时流数据交换和处理成为可能, 为 DSMS 系统有效地管理流数据提供了理论上和实际上的规范与指导。

参考文献:

[1] tan Zdonik. The Aurora and Medusa Projects[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2003, 26(1): 3-10.

[2] Chuck Cranor. The Gigascope Stream Database[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2003, 26(1): 27-32.

[3] Abhinandan Das. Approximate Join Processing Over Data Streams [J]. ACM SIGMOD Record, SIGMOD, 2003, (6): 40-51.

[4] Dan Suciu. XML Stream Processing[EB/OL]. www.cs.washington.edu/homes/suciu. 2003-10.

[5] 刘云生. 现代数据库技术[M]. 北京: 国防工业出版社, 2001.

作者简介:

桂浩, 男, 博士, 主要研究方向为 XML 数据库及 XML 数据流、网络数据库、数据挖掘领域研究; 冯玉才, 男, 教授, 博士生导师, 主要研究方向为数据库理论研究; 李又奎, 男, 硕士, 主要研究方向为数据检测、数据挖掘领域研究。