第 36 卷 第 8 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.8 Computer Engineering April 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)08—0258—03 文献标识码：A 中图分类号：TP391

JCA 适配器基础开发框架的设计与实现

周 伟，时俊苓，徐 罡，叶 丹
(中国科学院软件研究所，北京 100190)

摘 要：针对当前 JCA 适配器开发难度大、效率低问题，提出一种 JCA 资源适配器的基础开发框架。该框架运用良好的设计模式，抽取并实现 JCA 规范中的系统契约和应用契约的核心部分，从而使开发人员只需实现和具体企业信息系统密切相关的功能特性，给出一套基于该基础框架的开发工具进行适配器的开发。实验结果表明，该框架能够有效地提高适配器的开发效率和质量。

关键词：JCA 适配器；基础开发框架；企业信息系统

Design and Implementation of Foundational Development Framework for JCA Adapter

ZHOU Wei, SHI Jun-ling, XU Gang, YE Dan
(Institute of Software, Chinese Academy of Sciences, Beijing 100190)

【Abstract】 Aiming at the high difficulty and low efficiency problem of the development of JCA adapter, this paper proposes a foundational development framework for JCA. This framework provides implementation of the most part of the system protocol interface and common client interface in the JCA specification. Developers only need implement the function points related to the specific Enterprise Information System(EIS). An example of development tool using this framework is given. Experimental results show it greatly improves the efficiency and quality of JCA adapter development.

【Key words】 JCA adapter; foundational development framework; Enterprise Information System(EIS)

1 概述

随着企业信息化建设的深入展开，异构信息系统整合(企业应用集成 EAI[1])成为企业当前实施的重点问题。J2EE 应用服务器是当今企业应用集成中使用最广泛的一种集成平台。Sun 公司提出的 J2EE 连接器架构(JCA)[2]定义了一种连接 J2EE 平台和企业信息系统(EIS)的标准架构。

JCA 采用分层的思想将系统集成分层进行处理，资源适配器作为中间层，一方面需要从应用服务器中获取 J2EE 服务支持，另一方面需要与 EIS 进行数据交互处理。开发这样一个系统级的组件非常困难：(1)JCA 资源适配器的开发并不是所有的开发者都能够胜任，它的开发模式与编写普通代码不同，JCA 的实现需要运用很好的软件设计模式。(2)资源适配的开发一方面需要深入了解 Java EE 应用服务器，实现连接管理、事务管理、安全管理的系统级 API，另一方面需要实现应用程序契约和接口，支持 J2EE 应用与 EIS 之间信息的交互，从数据上把 EIS 系统集成到 J2EE 应用中。(3)JCA 作为一个统一的规范，它的实现也需要很多的标准与规范来支持，如 XA 分布式事务等。(4)连接不同的 EIS 的 Adapters 有很多共性，如果为每个不同的 EIS 都从头开始开发适配器，将会浪费大量的资源。

为了方便开发人员进行适配器的开发，本文提出一种可扩展的高效适配器基础开发框架：AFC(Adapter Foundation Class)。AFC 在 JCA 规范的基础上，抽象出资源适配器之间可以重用的基本功能特性，提供了 JCA 规范中的系统契约和应用契约的抽象实现，而将和具体 EIS 密切相关的功能特性留给具体的资源适配器开发人员来实现，从而大大减轻了资源适配器的开发工作。另外，AFC 还提供了 Assured Event Delivery 的 QoS 服务[3]，提高资源适配器的信息流服务的可靠性。利用 AFC，开发人员不需要太多了解规范协议，只要实现资源适配器中与具体 EIS 相关的特有部分，就能快速开发一套适配器。

2 适配器基础开发框架的设计

JCA 目前是 1.5 版本，分为 Outbound 和 Inbound 2 大部分。Outbound 是指从 J2EE 应用服务器中调用外部 EIS 程序。而 Inbound 则相反，是外部程序访问 J2EE 应用程序，能够让资源适配器异步地向应用服务器中的消息处理节点传输消息，而不依赖于消息的格式、语义、消息的内部结构等。JCA 规范中定义了 2 种契约：一部分为系统契约；另一部分为应用程序契约。

Adapter Foundation Class 作为一套基础开发框架，它定义了整个体系结构、协作构件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间的协作，它为构件复用提供了上下文关系。所有的框架都是遵循好莱坞原则设计的，所谓好莱坞原则，就是 You don't call us – we will call you，意思是在一个框架下的代码，都是被动地被

基金项目：国家“863”计划基金资助项目(2007AA01Z149, 2007AA04Z148, 2007AA010301)；国家“973”计划基金资助项目(2009CB310704)

作者简介：周 伟(1985－)，男，硕士研究生，主研方向：网络分布计算；时俊苓，硕士研究生；徐 罡，副研究员；叶 丹，高级工程师

收稿日期：2009-12-12 **E-mail：**wuwang000@gmail.com

framework 调用，而不是相反。通过这种方式，大量重复的代码就可以隐藏在框架里面，需要特别设计的代码以预定的接口方式交给开发人员，写好后由框架调用。基于以上原则，Adapter Foundation Class 分别针对 Outbound 和 Inbound 2 个部分进行分析、组件划分和责任分配，在 AFC 中进行 2 种消息通信的流程定义并实现。对于组件中与具体 EIS 相关的部分进行抽象化，开发人员需要实现这些抽象部分，实例化和启动整个框架，其基本结构如图 1 所示。Inbound 和 Outbound 就是对 JCA1.5 中消息流出和消息流入 2 种交互模式的部分实现，资源适配器开发人员通过扩展 Adapter Foundation Class 中定义的“扩展点”(抽象类，接口或者抽象方法)，实现具体适配器的特有部分。

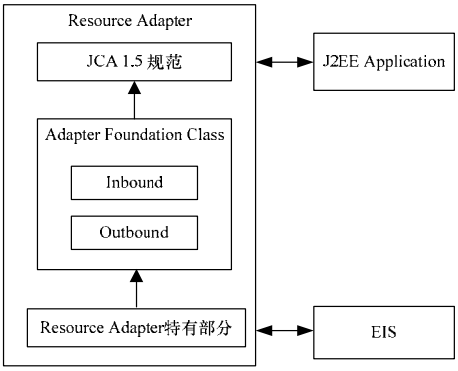

图 1　AFC 基本结构

下面将分别从图 1 中的 Outbound 和 Inbound 2 个方面来说明 AFC 的设计。

2.1　Outbound

Outbound 是指 Java EE 应用程序对 EIS 的调用。根据 JCA 规范，要实现 Outbound 消息流入通信，需要实现以下 4 类接口：连接接口，交互接口，交互数据接口和元数据接口[4]。这 4 类接口的实现在很大程度上是为了和 J2EE 应用服务器相集成而定义的系统契约，这些接口实现起来虽然复杂但是比较固定，主要是为了利用应用服务器提供的连接池功能来池化管理连接，更好地实现与 EIS 的交互。和具体 EIS 相关的主要包括连接属性、交互参数以及 Outbound Adapter 的元数据信息。这些接口定义了 Outbound 的连接管理和交互管理 2 个方面。

2.1.1　连接管理

在 JCA 规范中连接 EIS 的连接管理逻辑包括 4 个主要的接口：Connection, ManagedConnection, ManagedConnection Factory, ConnectionFactory。连接管理逻辑中的可变部分是连接特定 EIS 的连接参数属性。不变部分是利用应用服务器提供的连接池方式池化连接，并采用工厂模式的方式为客户端应用提供方便一致的编程接口。JCA 规范要求连接属性需要加在 ManagedConnectionFactory 接口的实现类里，这些连接属性需要在 ra.xml 配置文件中加以描述。AFC 提供了 ManagedConnectionFact-ory 的抽象实现 ManagedConnection FactoryImpl。用户只需要继承该类，实现特定的连接属性，同时需要实现方法 createConnectionFactory 供应用服务器取得 ConnectionFactory 接口的实现类的实例。对 EIS 物理连接是通过实现 ManagedConnection 接口来实现的。AFC 中提供了抽象实现 ManagedConnectionImpl。用户需要继承该类并完成对 EIS 连接的操作，如获取关于 EIS 的元信息的方法、获取连接句柄的方法以及关闭对 EIS 的连接等方法。连接句柄 Connection 代表了物理连接的连接句柄，AFC 中提供了抽象实现 ConnectionImpl，该抽象实现委托物理连接实现了自己的生命周期管理功能以及为客户端提供了 EIS 的数据信息。用户需要继承该类，提供一个接受 ManagedConnection 实例的构造函数以及创建交互实例的方法。ConnectionFactory 为客户端应用提供了一个获取连接句柄的工厂接口，AFC 提供了该工厂的基本逻辑实现，如获取连接句柄、获取管理的连接工厂以及获取适配器的元数据等。

2.1.2　交互逻辑

JCA 中 Interaction 接口代表了和 EIS 的交互，AFC 提供了抽象实现，如获取连接句柄等。用户需要继承该类实现 execute 方法，提供实际的 EIS 的交互。在具体的交互中，资源适配器需要对 EIS 执行的操作信息封装在 Interaction Spec 实现类中。AFC 给出 InteractionSpec 的抽象实现，提供一个基本的属性 functionName，用户需要继承该类，实现具体的操作属性信息。

2.2　Inbound

Inbound 是指从 EIS 到应用组件的异步消息流入。在 Java EE 中解决异步输入的问题，一般有 2 种方式：一种是使用 JMS，这样就要求外部系统也使用 JMS 来发送消息，从而限制了外部系统必须是 Java 程序，并且必须是一直运行的。另一种是容器采用定时器定时访问外部系统[5]。Inbound 消息流入模型采用第 2 种即主动“拉”的方式，主要由 3 个方面来实现：事件获取机制，消息订阅机制，消息传输机制。当 EIS 产生事件消息时，Inbound Adapter 通过事件获取机制来获取事件消息，并进行消息数据格式转换，事件管理器根据事件订阅关系，将消息传输给部署在应用服务器中的 Message Endpoint。在消息的传输过程中，应用程序、应用服务器以及 EIS 都可能发生错误，为了保证消息的一次可靠传输，在传输过程中需要有相应的可靠传输保证。整个消息流入模型如图 2 所示。

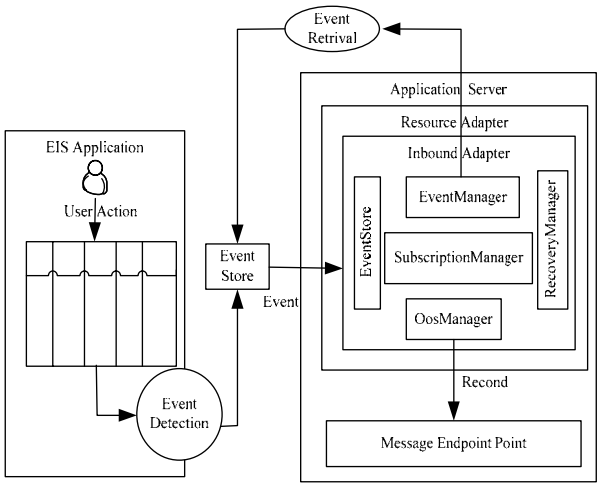

图 2　Inbound 体系结构

2.2.1　事件获取机制(Event Retrieval)

(1)EIS 在 User Action 的触发下产生事件消息，EIS 自带的事件检测机制(Event Detection，如数据库触发器等)会将该事件记录存放进 EIS 的 Event Store。Event Store 是和具体 EIS 应用程序相关的一种事件存储机制，它是一个持久化的存储器，会一直存储这些消息直到资源适配器正确获取并处理这

些消息。可以根据具体 EIS 的不同，分别采用数据库表、平面文件、消息队列、邮件系统等。

(2)资源适配器通过应用服务器提供的工作流管理服务的支持，分配到一个定时任务线程，该线程驱动 Event Manager 组件每隔一段时间去轮询 EIS 的 Event Store，其中，轮询的间隔时间通过资源适配器配置文件是可配置的。EventManager 是整个 Inbound 消息流入的调度中心。EventManager 首先通过 EventStore 组件来访问 EIS 的 Event Store 存储器。这样 EIS 消息事件就从 EIS 流入资源适配器。EventStore 是 AFC 中定义的访问 EIS Event Store 存储器的抽象接口类。开发人员根据具体 EIS Event Store 存储器类型实现该接口，提供具体的 EIS 存储器的访问器。

2.2.2 事件订阅机制

Message Endpoint 可以只接收感兴趣的事件消息。Message Endpoint 在部署时可以在部署描述文件中发布需要订阅的事件的属性信息。为了降低 EIS 同 Java EE 应用之间的耦合关系，资源适配器的事件订阅机制的设计采用了白板模式，SubscriptionManager 就是这块白板，它保存有 MessageEndpoint 的事件订阅关系。应用服务器在启动资源适配器时会将 MessageEndpoint 订阅的事件属性信息注册给 SubscriptionManager。在资源适配器获取到 EIS 的 Event 以后，可以根据白板中的订阅关系来决定 Event 的发送。在 AFC 中，为了提供定阅机制的可扩展性，SubscriptionManager 采用模板方法模式预留可扩展的接口，具体适配器的开发人员可以实现自定义的订阅关系。

2.2.3 消息传输机制

(1)在 JCA1.5 中，Message Endpoint 可以采用 Message-Driven Bean 来接收异步消息。EJB2.1 以后 Message-Driven Bean 可以实现任意接口，这样消息驱动 Bean 就可以接收任意格式的消息。

(2)在消息传输过程中，应用服务器、Message Endpoint 以及资源适配器都可能发生错误，导致消息传输的漏传或多传。为了保证消息的一次正确传输，本文采用了阶段表(staging table)的方式实现消息的可靠传输。AFC 中定义了阶段表的抽象接口和阶段表需要的抽象方法，开发人员可以根据实际情况采用自定义的数据库实现阶段表。可靠保证模型示意图如图 3 所示。

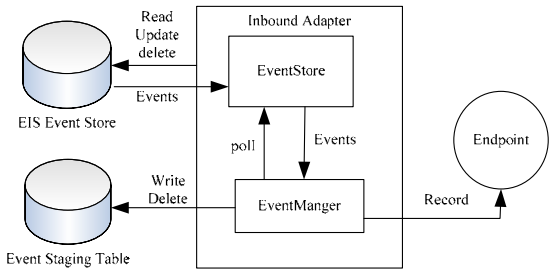

图 3 可靠传输模型

如图 3 所示，当一条新的 Event 被添加到 EIS 的 Event Store 存储器中时，这条事件记录的状态被标识为 New。当资源适配器轮询 EIS 的 Event Store 存储器时，适配器将会获取所有标记为 New 的记录。EventManger 获取到状态为 New 的记录后，首先根据 SubscrpitonManager 维护的订阅关系去除未被订阅的消息，然后将被订阅的消息记录的 Event ID 以及订阅该消息的 Endpoint ID 存入适配器中的 Event staging table。另外还须将 Event Store 存储器中的消息记录的状态更新为 In progress。接着 EventManger 向 Endpoint 发送消息，MessageEndpoint 收到消息后，将 Event Store 中对应消息记录的状态更新为 deletion，并将该消息记录从阶段表中删除。这样就完成了一次完整的消息传输。

(3)EventManager 在每次轮询 EIS Event Store 存储器前，EventRecovery 首先会获取阶段表中的记录信息。如果阶段表中存有事件记录，EventRecovery 根据该事件记录的 EventID 去查询 Event Store，获取对应 EventID 的事件记录。如果存储器中对应 EventID 的事件记录的状态为 New，则表示 EventManager 把事件记录从 Event Store 中存储到 Staging table 中之后被中断操作。此时，需要进行的恢复操作为更新事件的状态为 Inprogress 后，执行接下来的发送操作。如果存储器中对应 EventID 的事件记录的状态为 In progress，则表示事件在发送给 Endpoint 的过程中发生故障而中断。此时需要重新发送事件到 Endpoint，然后删除阶段表中的记录，最后更新 EIS Event Store 中的该事件状态为 deletion。如果存储器中对应 EventID 的事件记录状态为 deletion，则表示消息已经正确传输，只需要将该记录从阶段表中删除即可。

3 基于 AFC 的开发工具

为了方便用户开发实际的资源适配器，本文提出并实现一个 Eclipse 插件 Adatper Development Tool(ADT)，集成在 Eclipse 环境中。ADT 利用本文提出的 AFC 来快速生成一个适配器原型。该原型系统包含所有 AFC 中预留的需要扩展的类和接口的框架实现类。用户仅需要实现这些生成的框架类和一些辅助类就能完成一个具体适配器的开发。ADT 提供了 2 个工具帮助用户开发定制的资源适配器：项目向导和部署描述符编辑器。项目向导以友好的界面引导用户生成一个资源适配器原型，包括导入 AFC 开发类库，生成 AFC 中需要扩展实现的类、接口、方法等和生成部署描述符 ra.xml 等。部署描述符编辑器可以为部署描述符文件提供图形化的界面编辑器，使 XML 文件的编辑更加容易。

4 相关工作比较

当前 Eclipse 的 Java EE 开发环境(WTP)中有一套适配器开发工具，但该工具需要从零开始适配器的开发，需要实现大部分的 JCA 规范，开发效率非常低。本文提出的新的适配器开发工具利用适配器通用基础框架降低了难度，减少了工作量，可以很好地集成在 eclipse 开发环境中，解决了企业应用程序开发中的适配器开发的瓶颈问题。

文献[6]针对 JCA 规范本身存在的一些不足之处，如不支持长时间运行的事务、仅支持 DTC 类型的短时间事务、对 XML 的支持不够、没有提供基于 XML 的接口来实现与非 Java 的异种系统的集成等，对 JCA 架构进行扩展，提出一套高可用性松耦合的扩展架构，进一步提升 JCA 架构的可用性。

5 结束语

本文提出一种 JCA 资源适配器的基础开发框架，提高 JCA 适配器的开发效率和质量，满足 EAI 过程中 JCA 适配器开发的需求。在下一步工作中，将对适配器和 Message Endpoint 的数据交互以及应用组件和适配器之间的数据交互进行改进，定义一种通用可扩展的数据交互模式。

(下转第 263 页)