基于软件体系结构的集成适配器集成模式研究*)

徐 罡 黄 涛 刘绍华 叶 丹

(中国科学院软件研究所软件工程技术中心 北京 100080)

摘 要 集成适配器是构建分布应用集成的重要组件之一,能够完成应用程序间的连接、介入及转换等功能,屏蔽了应用集成复杂性。尽管集成适配器在分布应用集成中起到关键性的作用,但是目前缺少对集成适配器系统性的研究。本文通过调查分析已有的适配器,给出集成适配器的基本特征并从多个视角归纳总结了集成适配器的类型,进一步分析了集成适配器的集成模式。通过对集成适配器集成模式的研究,可以为不同集成场景提供解决方案并决定其适用范围,支持适配器的快速实现。最后,以资源适配器为例说明集成模式的应用。

关键词 集成适配器,分布应用集成,集成模式,软件体系结构

Research on the Integration Patterns of the Integration Adapters Based on the Software Architectures

XU Gang HUANG Tao LIU Shao-Hua YE Dan

(Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080)

Abstract Integration adapter is the key component in distributed application integrations. Currently, the most parts of integration functions are accomplished in integration adapters, such as connecting, accessing, transformation between distributed applications. Though integration adapter plays an important role in distributed application integrations, the systematic research on integration adapter is still shortage. Based on the past investigating and analyzing the existing adapters, we systematically explain the basic characteristics of integration adapters and summarize the categorization of integration adapters from several different views, further more present adapter integration patterns. The research of adapter integration patterns can serve the common emerging integration scenarios by adopting the different integration patterns and help to decide the scope where each adapter should be applied. At last, we illustrate how to utilize the adapter integration patterns to resolve the integration problem by using the resource adapter.

Keywords Adapter, Distributed application integration, Integration pattern, Software architecture

目前,软件系统的开发方法正逐步由面向功能的开发方法转向面向流程和面向服务的开发方法,那么如何利用分布应用集成技术构建大型信息系统就成为关键问题。在分布应用集成中,各个子系统之间往往存在通讯方式的不一致性、接口的不匹配、介入方式的多样性、数据的多样性等众多问题。例如,流程协调系统如何协同底层的信息系统,来自不同开发商的组件或 COTS(commercial off-the-shelf)产品之间又如何交互。这些问题都导致了分布应用集成的困难性,而集成适配器正是解决这些集成问题的关键组件,它为系统的其他组件屏蔽了这些复杂性,因此使分布应用集成系统能够在不对应用程序代码和数据结构做大的改动的前提下,具有集成其他应用程序和组件的能力。

集成适配器是分布应用集成的关键技术之一,但它常常像计算机硬件的外围设备一样被人们所忽略。对适配器的研究最早始于包装器[1],其后适配器的功能逐步扩大并趋于标准化,如 Sun 公司提出的资源适配器,并在 J2EE(Java 2 Enterprise Edition)规范下制定了 JCA(J2EE Connector Architecture Specification)规范[2]。对于适配器类型的划分,目前大都采用面向特定应用的划分方法,如针对特定 ERP(Enterprise Resource Plan)适配器、针对某种主机的适配器,这种分类方法导致不同类型适配器的解决方案混淆在某种类型的适配器中,不利于对适配器的分析与设计。而对于适配器集成模式的研究,可以说文[3]是最早提出适配器集成模式的概念,但它只是停留在工程总结的阶段,并没有深入系统的分析。

本文针对这些问题,首先讨论了集成适配器的基本特征,归纳总结了集成适配器的定义;在此基础上,从多个视角对集成适配器进行了分类;并依据在不同的分布应用集成场景,对适配器集成模式进行了分析与总结,从而定义了多种适配器集成模式;最后,以 JCA 资源适配器的设计作为应用实例,指出适配器集成模式在集成适配器设计和开发中的应用。

1 集成适配器

在分布应用集成中,集成中间件往往需要通过集成适配器介入应用系统内部的功能和数据,而集成的应用系统本身具有多样性,并且各种集成中间件所使用的介入方式不同,导致适配器本身的复杂性和多样性。简单地讲,适配器是一种组件,它屏蔽了进入应用系统的集成复杂性。集成适配器也

*)本课题得到国家"863"高技术发展技术发展计划项目(2002AA413610,2003AA413010,2003AA115440)和国家"973"重点基础研究发展规划(2002CB312005)的资助。徐 罡 博士,主要研究领域为软件工程、分布式计算、企业应用集成;黄 涛 博士,研究员,博士生导师,主要研究领域为软件工程、分布式计算;刘绍华 博士生,助理研究员,主要研究领域为分布式计算、工作流技术、服务协作;叶 丹 博士,副研究员,主要研究领域为软件工程、分布式计算、虚似企业。

被称为适配器、资源适配器、连接器和包装器。资源适配器是由 Sun 公司提出的一种基于 JCA 接口协议的、运行在 J2EE 平台上的特定集成适配器，用来集成企业信息资源；连接器本质上与集成适配器不同，两者是两个独立的组件，连接器实现组件间或应用程序间的连接，用来传递控制和数据，如 RMI (Remote Method Invocation)、MQM(Message-Queue Middleware)、隐含调用，有些集成适配器也将连接器的功能作为一个功能模块封装在集成适配器中；包装器用于对采取不同打包方式的组件进行二次打包，实现接口和数据的转换。包装器是适配器的原型，适配器是在包装器的基础上进一步发展而成的[4,5]。关于适配器的定义，目前还没有统一的标准。本文采用自描述的方法，将集成适配器定义为：

集成适配器是一个软件服务的集合，它通过特定的 API、数据库访问或其他的集成点，连接和进入应用程序内部的数据和功能，从而达到集成应用程序的目的。

正是由于分布应用集成的复杂性导致了集成适配器的多样复杂性，因此对集成适配器类型的研究有助于对集成适配器功能的圈定。本文采用多视角的方法依据集成适配器的多个显著正交特性划分集成适配器类型。集成适配器的正交特性可以归纳化分为服务对象、非功能服务对象和拓扑结构三方面。服务对象是指集成适配器的通讯和协同能力，它又可细化为数据传输和控制传输两方面。数据传输是指集成适配器所支持的数据传输类型，如应用程序间传递信息、处理的数据及计算结果；控制传输是指适配器所支持的控制传输类型，如应用程序间通过传递执行线程进行交互。非功能服务特性是指适配器所提供的数据和控制之外的其他高级服务，这些服务是构建在数据和控制传输之上，面向特定应用程序的服务，如会话适配器、面向服务的适配器。拓扑结构特性考察集成适配器在分布应用集成结构中的位置，可分为直联和星形。依据以上集成适配器的自身特性，集成适配器分为以下几类：

· 文件适配器。文件适配器是面向数据类型的一种，它限定传递的只是文件，不包含控制流。文件适配器的显著特点是一般不需要特定连接器(如 RMI、MQM)的支持，只是简单地使用系统本身的 Socket 或 Stream，适配器本身通常需要对文件进行必要的数据类型和数据格式的转换。

· 查询适配器。查询适配器是另外一种面向数据类型的适配器，它通常与数据库建立远程连接，返回查询结果，其本质上是采用"拉"的方式。适配器的安全管理提供必要的界面登录信息或安全证书，永久存储机制用来管理查询结果的选择和抽取，一般查询适配器不需要解析和映射数据。

· 事件适配器。事件适配器是面向控制类型的一种，与事件连接器相连。其重要功能是为非事件系统实现一个事件机制，封装为一个事件系统。适配器可以检测系统内部的状态，产生特定的事件，或者依据收到的事件触发特定的功能。

· 消息适配器。消息适配器也是一种面向控制的类型，与消息中间件相连接。适配器将不匹配的消息进行转换，或者将应用系统的 API 封装为一个消息系统。

· 远程调用适配器。远程调用适配器是另一种面向控制的类型，它与 RPC、RMI 等 API 控制传输方式相联接，提供对接口、参数的转换或二次包装。

· 直连适配器。直连适配器是拓扑类型的一种，在这种适配器使用中，通常需要集成的两个应用程序通过适配器直接连接，适配器进行基本的数据转换和映射，并且其中封装了特定的连接器。直接适配器适用于需要集成的应用程序的数量较少，或者应用程序位于同一命名空间(或同一主机)之上。

· 星形适配器。星形适配器是拓扑类型的另外一种，它通常与集成代理组合使用，其集成拓扑结构是集中的星型结构，实现多个源程序与多个目标程序间的多对多的连接。这种适配器一般只做较少的补充工作，补充工作决定于特定的目标程序的需求，集成代理处理大部分工作，包括数据转换、数据和控制信息的路由、信息过滤等。

· 服务适配器。服务适配器是近年来随着 Web 服务的广泛使用而出现的一种新型适配器。几乎所有应用都可以包装成 Web 服务并以 WSDL 的格式提供给外部世界；SOAP 消息通过绑定到 HTTP、SMTP 等公用的底层通信协议，可以跨越企业的防火墙，实现最广泛的应用集成；UDDI 则提供了一套服务的动态发现机制，满足松散耦合的应用集成需要[6]。

对集成适配器的分类，有助于理解和规范集成适配器的使用。适配器的使用及其功能特征随着集成场景的不同而不同，可以是多种类型的适配器综合使用，并且适配器的类型也随着需求的深入而需要不断地扩充和完善。

2 集成适配器集成模式

在软件领域中，模式的方法被广泛地采用，如面向对象的设计模式、软件开发的过程模式、软件体系模式等等。引用 Christopher Alexander 的定义，模式可以定义为"每个模式描述一个在特定的环境下重复发生的问题，模式能够对此问题提供核心解决方案，该方案能够多次使用，而不仅仅一两次"[7]。适配器集成模式也类似，它是面向特定的分布应用集成场景，能够反复多次地解决该场景下出现的集成适配问题。

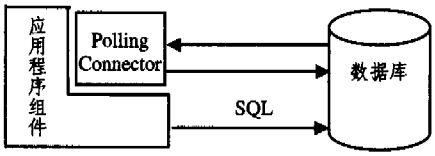
本文引入软件体系结构的基本问题(如基本设计词汇、结构模型、不变量、优缺点等)作为构建适配器集成模式的基础，这是因为适配器本身也是一种软件组件。其次，我们将适配器集成模式的抽象层次与软件体系结构中的软件结构模式相对应[8]。在适配器模式中，使用组件、连接器及说明两者如何组合的约束作为基本设计词汇，并在三者的基础上进行了细化，如组件可以细化为元数据组件、数据映射组件、配置组件等，连接器可以细化为消息连接器、RMI 连接器、事件连接器等，这种细化是依据特定的适配器集成模式进行的。在此基础上，我们提出七种适配器集成模式，分别为数据库访问模式、文档交换模式、遥远方法唤起模式、消息模式、事件模式、事务代理模式、用户界面协议模式。

2.1 数据库访问模式

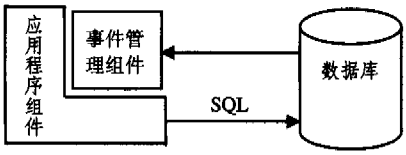
数据库访问模式是一种最基本的应用程序集成模式，大部分不提供对外 API(Application Programming Interface)的应用程序，但基本上都采用关系数据库存储数据。因此在缺少其他技术的情况下，数据库访问模式是缺省的分布应用集成方法。在数据库访问模式中，通常不需要对数据进行进一步的处理，因此组件相对简单，其连接器可以采用轮询连接器和事件连接器两种方式。在使用轮询连接器方式中，固定时间间隔发送 SQL 语句，监控数据库中数据记录的状态，判断应用程序消息包含的数据是否变化。在使用事件连接器方式中，采用数据库中特定状态的改变触发事件，由事件通知应用程序，再发送查询语句，获得数据库数据。数据库通常提供存储过程来实现这种事件触发机制，存储过程本身是一个可执行的代码，由数据库的特定事件触发执行，如图 1 所示。

采用轮询连接器介入数据库的缺点是效率低、潜伏期较长。如果数据量不大、对潜伏期要求不高，可以通过扩大查询

周期、最小化空轮询的可能性，则采用轮询连接器介入数据库还是一种有效的方法。采用事件连接器可以克服轮询连接器的上述不足，但其实现机制较为复杂。



(a) 轮询模式



(b) 事件模式

图 1　数据库访问模式

其次，对于轮询连接器是采用“拉”方式，而事件是“推”方式。事件系统是一种低耦合的、能够柔性地变更集成组件；而轮询连接器耦合程度较高，变更集成组件需要修改代码。

数据库访问模式的优点是简单易于实现，被认为是一种缺省的分布应用集成方式。但是，它本身也有不足，这种不足主要是数据库访问模式绕过了应用程序的应用逻辑，直接进入应用程序的数据库，这样有可能破坏数据库数据的完整性。

2.2　文档交换模式

文档交换模式可以说是数据库直接访问模式的一种改进方式，也是一种普遍采用的集成方法。应用程序提供特定的文件或输入/输出表作为数据交换的方式。在这种模式中，适配器和应用程序间是特定格式文件和输入输出表，集成的应用程序间通过这些文件或外部表交换数据。如 Oracle 数据库提供开放界面表(OIT，Open Interface Table)，来交换数据库内部数据。在文档交换模式中，适配器可以需要连接器作为文件和输入输出表的载体，或直接使用系统本身的 Socket 或 Stream。集成的应用程序间的文档常常存在较大差异，需要对数据的类型、句法及语义进行转换。对于特殊的情况，还需要文档与对象的转换，这就需要在文档中建立对象引用和继承关系。在文档交换模式中，适配器包含的组件主要有数据映射组件、原数据获取组件，如图 2 所示。

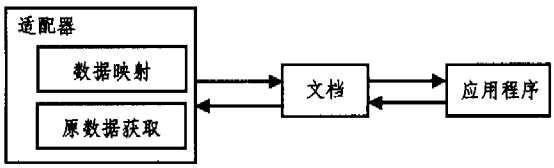


图 2　文档交换模式

使用文档交互模式的好处在于能够保护数据的完整性，并且易于实现，但存在集成反应期较长的问题。

2.3　远程 API 调用模式

远程 API 调用模式也可以称作远程方法调用(RMI)模式或远程过程调用(RPC)模式，要求被集成的应用程序提供可调用 API 接口，通过 API 接口调用内部的功能或访问内部的数据。远程 API 调用模式支持较高层次的集成，它向外开放的是应用程序内部的业务逻辑，是一种面向服务和功能的集成方法。在远程 API 调用模式中，连接器的类型可以细化为 API 调用、回叫(CallBack)及数据获取三种方式。API 调用方式是同步的通讯方式，对集成的应用系统的反应期要求较高，需要实时响应。回叫机制为遥远方法唤起模式提供了异步通讯机制，它本身是一种基于事件的扩展，由应用程序的执行结果唤起 CallBack 方法。典型的适配器组件类型包括接口转换组件、回叫绑定机制。接口转换组件实现不匹配 API 接口间的转换，包括不同函数的映射和参数的映射，其映射关系可以是一对多和多对一等形式；回叫绑定机制实现回叫事件与特定函数的绑定，如图 3 所示。

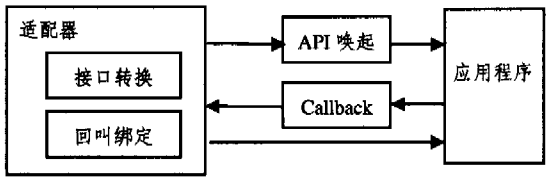


图 3　远程 API 调用模式

使用远程 API 集成应用程序是一种高级的集成方式，它能够保持数据的完整性，提供对外开放的业务逻辑，组件开发模型能够对外提供 API，其实现机制多采用 RMI、CORBA 及 DCOM，也有的应用程序使用特定的 API。但问题是大部分现有的信息系统(特别是遗留系统)不提供可集成的远程 API 调用。

2.4　基于消息的适配器模式

在基于消息的适配器集成模式中，应用程序间通过消息实现集成，是一种异步的、多对多的集成方式，能够克服其他集成方法的同步紧耦合和点对点特性的不足。消息模式使用消息中间件作为连接器，消息适配器位于消息中间件的两端，分别为消息发送适配器和消息接收适配器，应用程序通常同时拥有这两种消息适配器。在基于消息的模式中，有的消息中间件不包含消息代理功能，而只是实现对消息的路由，这就需要在适配器中扩展消息映射功能。此外，适配器的映射不只是局限于消息与消息的映射，适配器还要完成其它消息映射，如消息与对象的映射、消息与文件的映射。基于消息的适配器通常包含消息收发、消息映射、消息分发等组件。其中，消息收发组件实现按着特定的渠道(Channel)或主题(Subject)发送和接受消息；消息映射组件实现不同格式的消息间的转换及消息与对象、文档间的转换；消息分发组件(或者说是功能激活组件)实现消息激活特定的应用程序功能。其结构如图 4 所示。

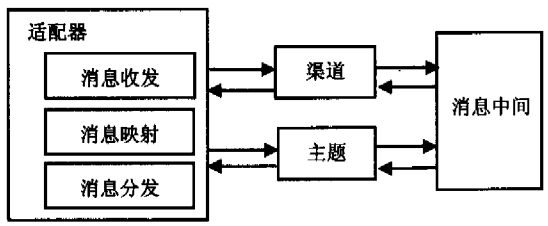


图 4　基于消息的适配器集成模式

消息集成是一种异步通讯、多对多的连接，异步方式适用于对实时性要求不高的集成场景，多对多连接方式为多个应用程序集成提供了很好的拓扑结构。消息连接器一般作为可靠数据传输的载体，可以分为文档消息和命令消息两种类型，满足应用集成对数据传输的需求。

2.5　基于事件的适配器模式

基于事件的分布应用集成是利用事件的隐含唤起特性。这种事件的隐含唤起可以为集成系统提供更大的柔性和可扩展性，增加和减少集成的应用程序，不会对其他应用程序和事件系统产生影响[9]。在事件系统中，可以使用独立的事件中间件，也可以是应用系统本身具有事件系统。在基于事件的适配器模式中，连接器为事件或事件系统，集成的应用程序通过事件来传递控制流和数据流，多采用广播或发布/订阅通讯方式。适配器所包含组件为事件提交、事件通知、事件绑定等基本组件。其中，事件提交组件用于检测应用系统的内部状态，产生事件，并将事件提交给事件通知组件。事件通知组件负责将事件分发给集成的应用程序，可以采用广播的方式将事件发送给所有集成的应用程序，集成的应用程序根据其内部的逻辑过滤出其感兴趣事件，也可以采用发布/订阅方式，将特定的事件发送给订阅者，或者采用更高级的基于内容的路由，根据事件通知组件内部包含的业务逻辑分发事件。事件绑定组件完成事件与应用程序内部功能或过程的绑定，即事件所触发的过程。其结构如图 5 所示。

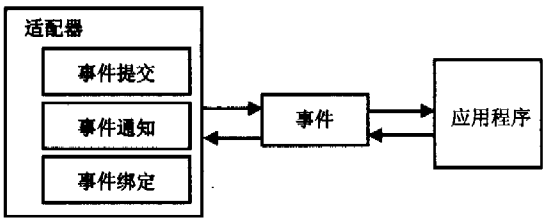


图 5　基于事件的适配器集成模式

基于事件的适配器模式将非事件系统对外包装为事件系统，能够接受处理事件，能够检测系统内部的状态的变化产生特定的事件，这些改变可以为数据的更新、特定时间的到达或特定功能的唤起。基于事件的适配器模式最大的好处是实现了应用程序间隐含的唤起，为集成的系统提供了最大的柔性和可扩展性。

2.6　基于事务代理的适配器模式

基于事务代理的适配器实际上是通过事务监控器 TPM (Transaction Processing Monitor)的客户端实现与应用程序集成。适配器通过 ATMI(Application Transaction Monitor Interface)进入应用程序、管理事务界限等。典型的事务监控客户端 Bea 的 Tuexdo 及 IBM 的 CICS 的客户端。在基于事务代理的适配器模式中，适配器可以使用多种连接器实现适配器与 TPM 客户端的连接。适配器的组件较为简单，直接调用 ATMI 界面提供的远程方法，其结构如图 6 所示。

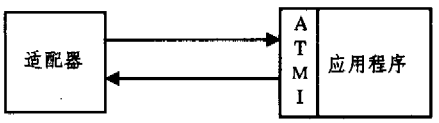


图 6　基于事务代理的适配器集成模式

基于事务代理的适配器模式局限于集成运行在事务监控器 TPM 的应用程序，其特点是利用 ATMI 提供的接口访问应用程序。

2.7　用户界面协议模式

用户界面协议方式是一种集成遗留系统的方法，特别是针对于某些大型主机。这些遗留的主机系统一般不提供对外 API 接口，而应用程序可以利用用户界面协议，使用与终端用户相同的机制与主机系统进行交互。在用户界面协议模式中，适配器相当于一个协议模拟器，如 TN3270 协议模拟器，允许通过 TCP/IP 与主机建立连接。

用户界面协议适配器对应用程序的操作与用户通过用户界面执行的操作相同。因此，用户界面协议模式是数据访问相对安全的方式，它没有绕过应用程序逻辑而与目标程序数据直接交互，能够保护数据的完整性。

表 1 给出基于分布应用集成特性对上述几种集成适配器集成模式的比较，主要包括是否需要使用连接器、实时特征、数据获取方式、功能调用方式、可用性、绑定特性、粒度、系统间的耦合性及拓扑结构的基数，通过这些分布集成特征能够从多方面反映各种集成模式。

表 1　集成适配器集成模式分布式特性比较

	连接器		实时性		获取方式		调用方式		可用性		绑定		粒度		耦合性		基数	
	使用	未使用	同步	异步	推	拉	显式	隐式	瞬时	永久	绑定	未绑定	数据	功能	松耦合	紧耦合	点对点	多对多
数据库访问模式		√	√		√	√	√			√	√		√			√	√	
文档交换模式	√	√	√			√	√			√	√		√			√	√	
远程 API 调用模式	√		√	√		√	√		√		√			√		√	√	
基于消息的适配器模式	√			√	√			√	√			√	√	√	√			√
基于事件的适配器模式	√			√	√			√	√			√	√	√	√			√
基于事务代理的适配器模式	√		√			√	√		√		√			√		√	√	
用户界面协议模式		√	√				√		√		√			√		√	√	

3　适配器集成模式扩展 JCA 资源适配器

本节使用 JCA 资源适配器的设计作为例子，说明适配器集成模式在适配器的设计和开发中的使用。在该例子中，我们只对 J2EE 平台所能够支持的模式进行扩展而没有涉及 J2EE 平台所不支持的模式，如对消息模式可以直接利用 J2EE 平台中的 JMS 组件。

目前，资源适配器是由 SUN 公司在 J2EE 规范中提出的唯一适配器标准 JCA，它是面向 J2EE 平台和 Java 应用程序的集成企业信息资源的解决方案。JCA 主要定义了两个接口集合，即通用用户接口 CCI(Common Client Interfaces)和系统协议 SC(System Contracts)。用户(包括 J2EE 可管理的应用程序和非管理的应用程序两类)通过 CCI 接口与资源适配器连接，对于可管理的应用程序，与资源适配器间是本地方法调用；对非管理的应用程序，它本身不是驻留在 J2EE 应用服务器上，需要通过远程方法调用(RMI)调用 CCI 接口。资源适配器通过 SC 接口与 J2EE 平台交互，使用 J2EE 平台提供通用基础服务，如适配器生命周期管理、连接管理、存储管理、

安全服务、事务服务等。

问题是 JCA 规范只是规定了用户与资源适配器、J2EE 平台与资源适配器的接口，没有对资源适配器内部功能模块的构成及资源适配器与 EIS(Enterprise Information System)连接做出特定规定，允许用户根据集成场景自由扩展，这一点也符合企业信息资源多样性的特点。

J2EE 平台目前可以提供数据库连接、远程方法调用 RMI 及消息系统 JMS 三种服务。因此，在 J2EE 平台上，我们可以方便地扩展三种类型的适配器集成模式，分别为数据库访问模式、远程 API 调用模式和基于消息的适配器模式。扩展后的资源适配器如图 7 所示。

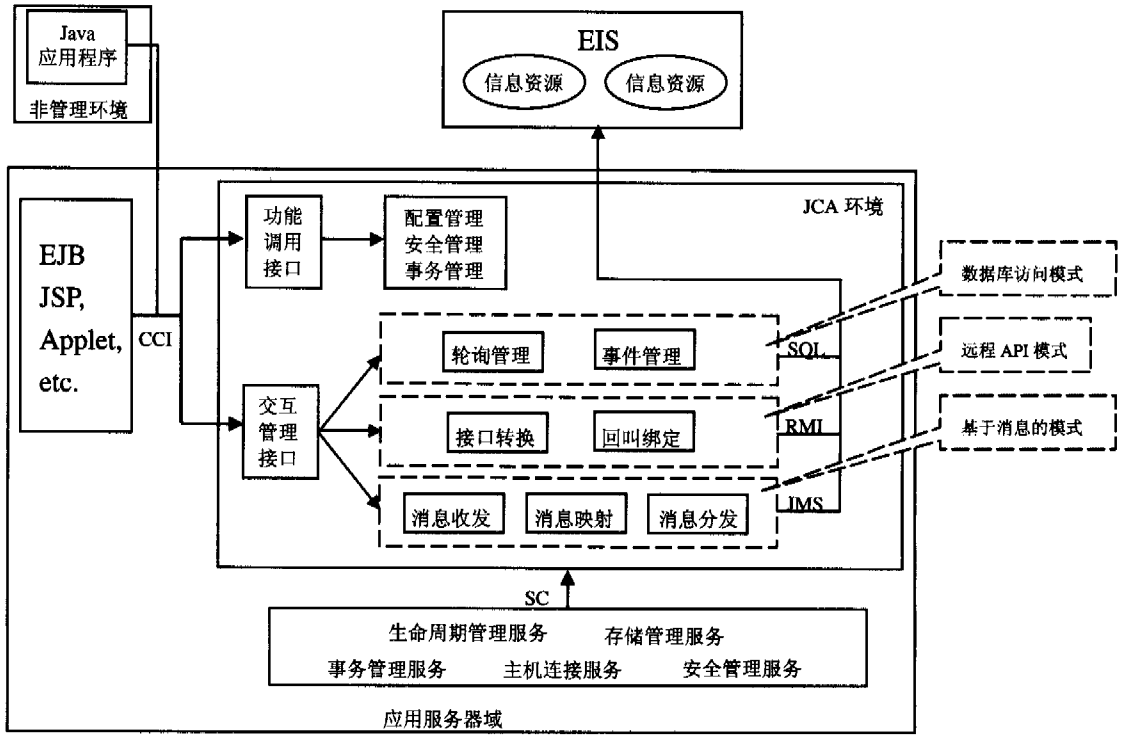


图 7　JCA 资源适配器的扩展

扩展后的 JCA 资源适配器模型包括适配器组件、JCA 运行环境、应用服务器域、客户应用程序及企业信息系统五部分。在客户应用程序调用中，将 CCI 接口划分为交互管理接口和功能调用接口。交互管理接口开放接口给外部管理工具，主要包括配置管理、安全管理及事务管理。配置管理提供对适配器的静态和动态配置，也包括启动/终止适配器服务、追踪适配器异常。安全管理提供安全服务，如用户登录、注销及能够接受用户安全信用状的能力。事务管理对客户应用程序提供事务服务，包括内部事务和外部事物两种类型。

功能调用界面对外提供一组服务，表示适配器实际提供的服务。功能调用界面直接与三种扩展的核心集成模式相联接，是适配器的核心功能。其中，JCA 支持的同步通讯机制包括 RMI 和数据库连接 JDBC 两种，RMI 提供对企业信息系统功能 API 的调用，接口转换模块对接口的处理包括对接口关系的映射，例如一个功能调用接口可以对应多个底层同步 API 接口、对接口参数的映射，例如对输入参数的增减和分配。对数据库集成机制，使用 JCA 提供的主机连接服务能够优化适配器连接性能、对数据返回的查询结果。

对于异步通讯，可以直接使用 JCA 的消息通讯机制 JMS (Java Message Service)，因此适配器不需要扩展基本的消息组件，只需要扩展必要的映射功能。对于资源适配器使用消息通讯，映射功能实现消息与消息、消息与对象及消息与文件的映射。资源适配器可以使用文档消息来传递文档文件，在这种方式中，使用解析和映射功能对消息中的文档解析和映射。目前的 JCA 规范对异步通讯只支持消息系统，不支持事件系统。如果企业信息使用事件系统，资源适配器需要扩展事件机制来处理事件的提交、通知、绑定。

资源适配器扩展模型除基本的功能外，还可以利用 JCA 提供大部分其他服务。资源适配器通过 SC 接口规范来调用这些服务，包括生命周期管理模块、安全管理服务、事务管理服务、存储管理服务。生命周期管理模块支持单周期的适配器和多周期的适配器。单周期适配器响应外部请求，处理完外部请求后，适配器被消除；多周期适配器能够预先配置，多次使用。安全管理服务提供与外部安全基础框架的交互；存储管理服务为适配器提供存储功能，用来存储原数据及业务规则。

结论　集成适配器是分布应用集成的关键组件之一，但它并没有获得人们的足够重视，常常像计算机硬件的外围设备一样被人们所忽略。事实上，集成适配器是分布应用集成中更具多样化、更富挑战的组件，在实际中它也完成分布应用集成的关键功能。本文依据集成适配器的特征给出了集成适配器的定义和分类；在此基础上，对集成适配器的集成模式进行了研究，运用集成模式有助于解决特定环境下的集成问题；最后，以资源适配器为例示意性地说明了集成模式的运用。随着应用场景的不断开拓及集成技术的新发展，还会出现其它类型的集成适配器和适配器集成模式，需要进一步的总结归纳和分析。

参 考 文 献

1　Linthicum D S. The evolution of adapters. EAI Journal, 2002. 37～40

(下转第 238 页)

metaprogramming)。这两种不同最大的区别是对目标程序的构造、分析、控制等的时机不同。前者是在程序实际运行时完成的,后者是在程序编辑阶段完成的。采用不同的方式是因为设计系统的目的不同。

MetaML是一个运行时元编程系统,为产生多阶段程序设计而设计,主要目的是提高程序针对具体问题的性能。Template Haskell是一个编译时元编程系统,为Haskell语言增加了编译时元编程机制,目的是为了让程序员能够在不改变编译器的条件下为定义新的语言特征,增强了语言的表达能力。

Template Haskell与C++模板元编程类似,都是在编译的时候执行。在执行时所运行的程序是由元程序在编译时执行所产生的普通程序,然后再编译一次,得到可执行程序。但是它们也有所不同,C++ Templates的发明是为了支持generic programming,其静态计算(static computations)和代码产生的功能是意外的发现,因此用C++ Templates进行元编程并不是很自然,需要一些技巧[26]。与之不同,Template Haskell是专门为元编程而设计的,元元语言采用Haskell本身,并增加了一些语法构造符和库函数,表达能力更强,更简洁易用。

总结和发展趋势 元编程系统的目的是为程序员提供方便、安全的构造和控制程序的手段。它把一些构造和控制程序的普遍的操作抽象出来实现,并添加到系统中,提供给程序员使用。这样,既可以减少程序员的重复劳动,又容易保证程序的正确性。因此提供目标程序简单的、自然的、表达能力丰富的表示方法,是元编程系统的主要问题。从MetaML和Template Haskell的共同点可以看出一些对元编程系统的需要:好的元编程系统应该提供友好的人和系统交互的接口。除提供目标程序的构造、目标程序的组合的基本能力以外,还应该有合理的类型系统,保证安全的变量使用,并且能够执行目标程序,能够输出目标程序,观察和分析目标程序[3]。

程序设计语言一直处于不断的发展中,对程序的控制从宏到模板再到程序自身,越来越方便,越来越安全。随着新的应用和挑战不断地出现,今后的元编程的应用将越来越丰富,元编程系统也会向有更灵活的语义、更强的表达能力、更模块化、更丰富的库函数方向发展。

参考文献

1 Walid T. Multi-Stage Programming: Its Theory and Applications. Oregon Graduate Institute School of Science \ Engineering, 1999
2 Robert D C, Ito M R. Grammar-Based Definition of Metaprogramming Systems. ACM Trans Program Lang Syst, 1984, 6: 20~54
3 Tim S. Accomplishments and Research Challenges in Meta-programming. Proceedings of the Second International Workshop on Semantics, Applications, and Implementation of Program Generation, Springer-Verlag, 2001
4 Hughes J. Why functional programming matters. Comput J, 1989, 32: 98~107
5 Multi-stage Programming Homepage: http://www.cs.rice.edu/~taha/MSP/
6 Czarnecki K, O'Donnell J, Striegnitz T, et al. DSL Implementation in MetaOCaml, Template Haskell, and C++. DSPG'04, 2004
7 Resource Aware Programming (RAP) Homepage: http://www.cs.rice.edu/~taha/RAP/
8 MetaML Home Page: http://www.cse.ogi.edu/PacSoft/projects/metaml
9 Template Haskell Homepage: http://www.haskell.org/th/
10 Cantwell S B. Reflection and Semantics in a Procedural Language: [PhD thesis]. MIT Laboratory for Computer Science, 1982
11 Cantwell S B. Reflection and semantics in LISP. Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. Salt Lake City, Utah, United States: ACM Press, 1984
12 Ming-Yuan Z. Computational reflection in PowerEpsilon. SIGPLAN Not, 1994, 29: 13~19
13 Anurag M, Daniel P F. Towards a Theory of Reflective Programming Languages. Informal Proceedings of the Third Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming, OOPSLA'93, 1993
14 Hook T S J. A Semantics of Compile-time Reflection. Dept of Computer Science and Engineering, Oregon Graduate Institute, Portland, Oregon Technical Report 93-019, 1993
15 Tim S. Guide to using CRML: Compile-Time Reflective ML
16 Tim S, Martel M. Introduction to Multi-Stage Programming Using MetaML Revision 2
17 Walid T, Tim S. Multi-stage programming with explicit annotations Proceedings of the 1997 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation. Amsterdam, The Netherlands: ACM Press, 1997
18 The OCaml Language Homepage: http://www.ocaml.org/
19 MetaOCaml Homepage: http://www.metaocaml.org/
20 Cristiano C, Walid T, Liwen H, et al. Implementing Multi-stage Languages Using ASTs, Gensym, and Reflection. Generative Programming and Component Engineering (GPCE'13), 2003
21 Tim S, Peyton J S. Template meta-programming for Haskell. SIGPLAN Not, 2002, 37: 60~75
22 Richard K, William C, Jonathan R. Revised5 Report on the Algorithmic Language Scheme. ACM SIGPLAN Notices, 1998, 33: 26~76
23 Jim G, Tom M, John O I. A Reflective Functional Language for Hardware Design and Theorem Proving. European Joint Conferences on Theory and Practice of Software (ETAPS), 2004
24 Tim S, Peyton J S. Notes on Template Haskell Version 2. November 7 2003
25 Philip W. Monads for functional programming. Program Design Calculi: Proceedings of the 1992 Marktoberdorf International Summer School, 1993
26 Template Metaprograms: http://osl.iu.edu/~tveldhui/papers/Template-Metaprograms/meta-art.html

(上接第233页)

2 J2EE Connector Architecture Specification, version 1.5. http://www.jcp.org
3 Yee A, Apte A. Integrating your e-Business enterprise. Sams Publish, 2001
4 Mehta N R, Medvidovic N, Phadke S. Towards a taxonomy of software connectors. In: Proc. of the 22nd Intl. Conf. on Software Engineering, Limerick, Ireland, 2000. 178~187
5 Deline R. A catalog of techniques for resolving packaging mismatch. In: Mili A, Mittermeir R, eds. Proceedings of the Symposium on Software Reusability. New York: ACM Press, 1999
6 Papazoglou M P, Georgakopoulos D. Service-oriented computing. Communication of ACM, 2003, 46(10): 25~28
7 Gamma E, Helm R, Johnson R, et al. Design Patterns-Elements of Reusable Object-oriented software. Addison-Wesley Publish, 1995
8 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996
9 Barrett D J, Clarke L A, Tarr P L, et al. A framework for event-based software integration. ACM Transactions on Software Engineering and Methodology, 1996, 5(4): 378~421