基于 Java EE 平台的企业遗留系统集成方案研究

摘　要

企业应用集成(Enterprise Application Integration,EAI)旨在将企业中完成不同功能、彼此相互独立的应用系统集成起来，并且不需要对现有系统做太大改造就可以让它们之间方便地共享业务数据和业务功能。虽然目前业界已有一些集成方案得了一定的应用，但面对集成性、开放性较差的遗留系统仍然显得力不从心。如何构建针对这些遗留系统的高效集成，又能提供开放服务体系的集成方案是当前企业应用集成面临的难点。

本文在对 Java 连接器架构(Java Connector Architecture,JCA)和 Web 服务技术深入分析研究的基础上，结合企业遗留系统集成的难点，提出了一种基于 Java EE 平台的企业遗留系统的集成方案，将 JCA 集成机制放在集成方案的后端，而把 Web 服务用在集成方案的前端接口部分，这样能使企业应用集成同时达到性能和灵活性的要求，既充分利用了 Java EE 平台的强大性能，又能以更加灵活和互操作性的方式将系统的业务功能展现给用户。

全文主要从以下三部分进行阐述：

首先，阐述了 EAI 相关概念、层次构成和一个完整 EAI 解决方案各层技术实现以及 EAI 实施的特点和实现平台的选择。

其次，对 JCA 和 Web 服务两种技术在 EAI 方案中应用进行了分析，并比较了两种技术在企业集成遗留系统中的应用，得出两种技术相结合的优势，提出一种基于 Java EE 应用服务器的企业遗留系统集成模型，并给出了设计思路以及分层技术实现，其研究成果表现在以下几个方面：

1) 完成了集成模型的技术分析、设计以及实现过程，并已成功的将开发的资源适配器部署到 Java EE 应用服务器 JBoss 中，提供了客户端应用组件、Java EE 应用服务器与异构的企业遗留系统的统一连接方法。
2) 在 Web 服务具体实现中，遵循 JAX-WS 模型的 Web 服务开发过程，最大程度上利用 Java EE 开发平台的特性以适应 Web 服务的变化，并使得 Web 服务封装、发布以及部署过程得以简便。
3) 针对集成模型服务集成层，提出了一种动态 Web 服务组合模型（D-WSCSM Dynamic Web Services Composition Model），提供了动态绑定 Web 服务和异常情况下动态修改 Web 服务的功能，增强 Web 服务的业务组合的能力。

最后，作者将该集成模型运用在淮北煤矿安全管理信息平台实际项目中，从而论证了集成模型的可行性。

关键词：EAI，企业遗留系统，JCA，资源适配器，Web 服务

Research of Enterprise Legacy System Integration Scheme Based on Java EE Platform

Abstract

Enterprise application integration (EAI) entails integrating applications and enterprise data sources so that they can easily share business processes and data. Integrating the applications and data sources must be accomplished without requiring significant changes to these existing applications and the data. At present, there are integration schemes have also obtained the widespread application; however, it also lacks the ability to do what we want when facing this openness and integration worse legacy system. How to construct an integration scheme which was effective, integrative, also can offer an opening service system was a difficult problem.

Based on the thorough analysis and research on EAI technologies-Web Services and Java Connector Architecture (JCA), and combining with the difficulty of enterprise legacy system integration, the thesis put forward an enterprise legacy system integration scheme based on Java EE platform, which lays JCA integration's mechanism on the back-end of integration scheme, and lays Web Services on the front-end of integration scheme, so that EAI can arrive the require of performance and flexibility. The integration scheme could both make full use of the strong function of the Java EE platform and make business function of system more flexible and operable to user.

Full text mainly writes from the following three parts:

First, the paper elaborate EAI related contents、layer constitution 、each layer technique realization of the complete solution of EAI and the character of EAI implement and the choice of platform.

Second, after the analysis of application in EAI between JCA and Web Services, and comparison between them in the enterprise legal system integration, I find the advantage in combining them ,and put forward the integration model of enterprise legacy system based on Java EE application server, and introduce the design thought and the realization of layer technology. The study result include following several aspects:

1) The thesis completes technology analysis、design and development process of the integration model .The resource adapter developed has been applied in the Java EE application server JBoss successfully. It provides a unified connection method relating to client application component, Java EE container and heterogeneous enterprise legacy system.
2) It follows the Web Services development process of JAX-WS model in fact design, make full use of the characteristic of the Java EE platform to adapt Web Services'

change, and simplify encapsulation, release and deployment process of Web Services.

3) The thesis put forward a dynamic Web Services Composition Model (D-WSCSM) toward integration layer of Web services, providing the function of dynamically binding Web Services and dynamically modifying Web Services and strengthening the business composition ability of Web Services.

Finally, the author apply integration model to the practical project of Huaibei Coal Mine Safety Management Information System (HBCMSMIS) to verify the feasibility of integration model put forward in this thesis.

Key words: Enterprise Application Integration, Enterprise Legacy System, Java Connector Architecture, Resource Adapter, Web Services

图 表 清 单

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 合肥工业大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：汪子涛 签字日期：2008年4月20日

学位论文版权使用授权书

本学位论文作者完全了解 合肥工业大学 有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 合肥工业大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

(保密的学位论文在解密后适用本授权书)

学位论文作者签名：汪子涛 导师签名：

签字日期：2008年4月20日 签字日期：2008年4月20日

学位论文作者毕业后去向：

工作单位： 电话：

通讯地址： 邮编：

致　谢

在本论文完成之际，我首先要向我的导师袁兆山教授致以最衷心的感谢！在论文选题、研究方案的确定以及整个写作过程中，袁老师都倾注了大量的心血并加以悉心指导。在研究生阶段的三年中，袁老师给予我巨大的鼓励、帮助和指导，提供了良好的工作学习环境的同时，更为我创造了理论提升与实践锻炼的机会，并以其严谨的治学作风，求真务实的学者风范，平易近人的生活态度深深地影响着我，使我不论在学业、社会实践还是为人处事上都受益菲浅。导师的启迪与教诲、关心与爱护，我将铭记于心。

向所有在我读研期间给予我无私帮助的老师们表示深深的谢意，向参与本论文送审、评审和答辩的老师致敬，感谢他们所付出的辛勤劳动。

感谢我的父母和亲人，是他们在我攻读硕士学位期间给我创造了潜心学习的条件，时时督促我要好好学习，在生活上和精神上都给予我极大的关心和支持。

感谢软件工程实验室的兄弟姐妹们，他们为我的研究工作和论文写作提出了很多宝贵的意见，在与他们平日的共同学习和朝夕相处中，能时刻感受到集体的温馨、友情的可贵。

汪 正 海

2008 年 3 月

第一章 绪论

1.1 研究背景

据 IDC 统计，在过去的 10 多年里，全球企业在信息系统上一共投资二十多亿美元。巨大的投资为企业建立了众多复杂的信息系统，以帮助企业进行内外部业务的处理和管理工作。

由于历史及其他多方面的原因，大部分企业信息系统是在不同的时期或者面向不同的需求进行开发的，这些系统往往都是异构的，并且它们由不同的厂商开发，采用不同的实现平台、不同的数据结构和实现方式。我们把那些不是按照新的软件技术和方法开发的，然而在现实中，又难以退出舞台的应用系统称为遗留系统。很多遗留系统都是用传统的方法开发的，并没有按组件的思想进行设计开发。大多数都是一些孤立的系统，很少与其他系统之间有信息的集成，而且界面表示逻辑通常与业务逻辑、数据访问逻辑混杂在一起[1]。一方面遗留系统是经过时间检验和实用的，其中倾注了数十年的努力和定制，成为整个企业 IT 战略过程中可以信赖的组成部分；另一方面，遗留系统还可能有许多缺点：不够灵活，维护费用昂贵，并且有些遗留系统并不向外提供 API[2]。

重要的是这些信息系统之间不能共享数据和业务逻辑，形成一个个的“信息孤岛”。当企业规模扩大时，这就限制了管理水平的提高，同时也使管理成本居高不下。

那么企业如何提供一个有效的机制，既能保留已有的投资，又能让如此众多的“信息孤岛”之间联系起来一起协同工作是众多企业目前所急需要解决的问题。

一种可行的解决方法是，根据实际需要，对这些遗留的应用系统进行总体规划，选择一个合适的集成平台，把企业现有的系统与这些遗留系统有机集成起来，如图 1-2 所示，这就是企业应用集成发展的源动力。

遗留应用系统与其他系统集成之前，必须对其进行包装。要集成没有接口的或者提供的 API 不适合需求的遗留应用系统，首先要对遗留应用系统进行分析，分析要考虑的因素有很多，核心是梳理需要整合的系统功能，然后通过封装来扩展应用，即组件封装。封装实质上是用一个掩藏老系统不需要的复杂性和输出一个现代接口的软件层来包装遗留系统。封装提供了必要的接口，通过这些接口可以存取这些遗留应用系统，如：EJB 组件。

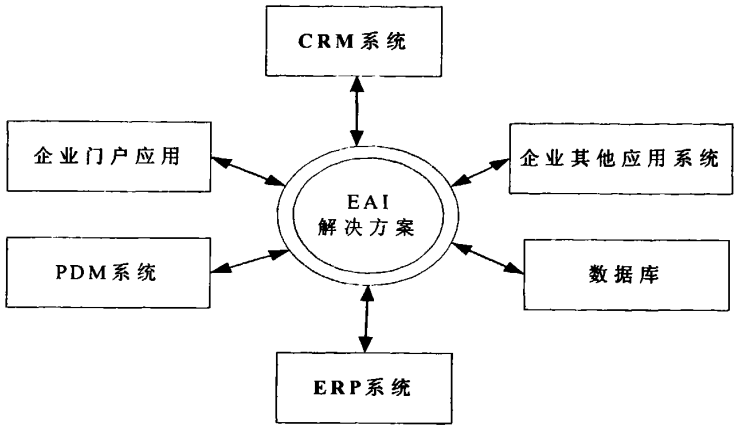

图 1-1 EAI 应用模式

1.2 国内外研究现状

1.2.1 国外研究现状

遗留系统自二十世纪九十年代兴起后，一些研究机构和个人从不同的方面进行研究。目前，从综合各国已发表的一些文献来看，研究主要集中在如下几个方面。

1.2.1.1 遗留系统再工程

软件再工程关心的是通过对遗留系统改造使它们的可维护性得到提高。再工程包括对系统重新建立其文档，组织并改造系统，用一种更先进的程序设计语言转换系统、修改和更新系统的结构和系统的数据。一般来讲，软件的功能不被改变，同时系统的体系结构也是保持不变的。

从技术的角度来看，软件再工程似乎是系统进化问题和一个次级解决方案；从业务角度来看，软件再工程可能是保证遗留系统能继续提供服务唯一可行的方法。再工程是一种相对新的软件进化方法，它界于系统替换和继续维护之间，它的系统成本比替换一个系统的成本要低，能降低与替换系统和继续维护所带来的巨大风险。再工程不是从写系统描述开始，而是将遗留系统作为新系统的描述，因此你能降低由于丢失包含在系统中关键的业务知识所带来的风险。再工程的目的就是以低成本进化一个能满足用户新需求的系统。目前，在这方面的研究比较多，Harry M.Sneed[44]提出了以遗留系统分析开始的五步再工程计划过程。Vijay K. Madisetti [45] 等人对遗留实时系统进行再工程。Ettore Merld[46]等人撰文得出一种再工程方法让遗留系统接口进化成新的接口技术，延长系统寿命并提高整体质量。Ian Warren[47]等人的Renaissance研究项目目的在于提出一个软件进化和再工程的动态方法，该项目定义了一系列活动和任务来支持全部再工程项目和识别活动之间的控制流。

1.2.1.2 遗留系统移植

在过去三十年间，大量使用旧技术（例如使用过程化语言）的软件得到了

发展，这类系统已经历了很长一段时间严格的代码修改。结果，不完整的设计文档和体系结构使得维护变得更加困难，花费时间更多，维护费用更高。另一方面，这些系统有着重要的经济价值，它们中的大部分对于企业来说都是非常重要的。对于这些投资高、专用性强的系统来说，简单抛弃它和重新使用新技术来开发新系统都不是一个好的选择。将遗留系统移植到新兴技术上来才是一个好的选择。虽然移植是一个复杂的生活系统工程，但是如果移植成功，它的长远效益远远大于再工程。移植使系统具有更多的灵活性、更易理解、更易维护、成本更低。遗留系统的移植必须将一个已存在的且能运行的系统移植到新平台上，同时要保持遗留系统的功能并且能使系统继续工作，这是一个非常大的挑战，它包含软件工程中许多领域，包括程序和数据理解、系统开发和测试。目前，有关这方面研究相当多。Jesus Bisbalm[48]等人对遗留系统移植方法进行调查研究，对各种移植方法进行了比较，并开发了一系列工具。Lei Wu[49]等人对怎样降低遗留系统移植的复杂性进行研究。

1.2.2 国内研究现状

1.2.2.1 遗留系统评价

遗留系统的进化方式可以有很多种，根据系统的技术条件、商业价值以及维护和运行系统的组织特征不同，可以采取继续维护、某种形式的重构或替代策略，或者联合使用几种策略。究竟采用哪些策略来处理遗留系统，需要根据对遗留系统的所有系统特性的评价来确定。

对遗留系统评价的目的是为了获得对遗留系统的更好的理解，这是遗留系统进化的基础，是任何遗留系统进化项目的起点。张友生[50]提出的评价方法包括度量系统技术水准、商业价值和与之关联的组织特征，其结果作为选择处理策略的基础。

1.2.2.2 遗留系统集成

在企业应用集成中，相当大的比重是与遗留系统集成，有人估计将占70%的工作量，这是因为出于保护已有投资资源的考虑。在IT应用基础较好的发达国家，这一问题更加突出。遗留系统的集成也不能把所有希望都寄托在软件再工程上，因为这需要很高的投入和开发资源，并不是企业都能负担得起的，并且遗留系统的概念也是相对的，今天的新系统，一段时间后又会成为新的遗留系统，因此，与遗留系统的集成，将是一个永恒的课题。在对遗留系统集成过程中，通过开发针对遗留系统的适配器组件来实现对遗留系统的封装和包裹（wrap）。适配器是提供对各种不同数据源支持的动态插件，它能屏蔽各种不同的数据源的底层技术区别，为数据源提供统一的访问接口，增强了应用和技术协同工作的能力。遗留系统经过适配器组件包裹就能够以Web服务的形式与系统的其它服务进行交互。西北工业大学计算机学院和协同时光软件公司开发的业务集成中间件，提供了丰富的适配器组件来实现对遗留系统的集成。哈尔滨

理工大学的姜晨[51]等人提出的基于CORBA遗留系统集成模型，并提出对遗留系统的解决方案，但是它只是对结构遗留数据源进行集成。

本文就针对目前集成方案中带来的高成本和专有特性，不能实现跨平台集成，以及集成系统的安全可靠运行，提出了一种基于 Java EE 平台的企业遗留系统应用集成的解决方案，并进行了具体的应用研究，为中小型企业实施企业应用集成提供一种更加完善易行的 EAI 解决方案，并在具体实施上提出指导方法，期望推动企业应用集成的发展。

1.3 研究内容与组织结构

本文通过对这些研究成果和现有的主流集成技术的分析与比较，并针对遗留系统的特点，提出了一种基于 Java EE 平台的企业遗留系统应用集成的解决方案，方案主要利用 JCA 和 Web 服务两种集成技术相结合的优势，利用两种技术对这些遗留系统的对外接口进行封装，实现遗留系统与现有系统进行有效地集成。把 JCA 集成机制放在集成方案的后端，而把 Web 服务用在集成方案的前端接口部分，这样能使应用集成同时达到性能和灵活性的要求，既充分利用了 Java EE 平台的强大性能，又能以更加灵活和互操作性的方式将系统的业务功能展现给用户。本文主要工作如下：

- 运用了多种设计模式的思想，使得开发出的资源适配器具有软件的可配置性和可复用性，并已成功的将开发的资源适配器部署到 Java EE 应用服务器 JBoss 中，提供了客户端应用组件、Java EE 应用服务器与异构的企业遗留系统的统一连接方法。
- 研究了利用现有 Java EE 开发平台的特性——标注(Annotation)，进行 Web 服务的开发过程，并使得 Web 服务封装、发布以及部署过程得以简便。
- 提出了一种动态 Web 服务组合模型，弥补了传统 BPEL 业务流程只能对 Web 服务进行静态绑定的缺陷，并提供了动态绑定 Web 服务和异常情况下动态修改 Web 服务的功能，增强了 Web 服务的业务组合能力。

由于文中涉及到了多方面的知识，如 Web 服务、工作流等技术，前期的技术准备、多种开源平台(如： Ant,Wsgen,Active-BPEL 等)的使用都需要大量的工作，也是本文的难点之一。本文的章节组织结构如下：

第一章，绪论。阐明本课题研究背景以及研究目的和意义，分析了 EAI 当前的研究现状和面临的问题，并介绍了论文的主要研究内容。

第二章，EAI 理论与技术基础。主要内容包括：阐述了 EAI 的概念，回顾了 EAI 构成，以及一个完整的 EAI 层次的划分和各层技术实现，并分析 EAI 实施的特点和实现平台的选择。

第三章，基于 Java EE 平台的应用集成方案。主要对 JCA 和 Web 服务技术应用在 EAI 方案中进行了探讨，并比较了这两种技术在 EAI 中的应用；然后研究了 JCA 技术和 Web 服务技术两者相结合在集成企业遗留系统中的优势；最

后将两种技术相结合，给出了一种基于 Java EE 应用服务器的企业遗留系统集成模型，并分析了该模型的设计思想。

第四章，集成模型中关键实现技术分析。对第三章提出的基于 Java EE 应用服务器的企业遗留系统集成模型中的基于 JCA 的 EAI 后端集成机制，基于 Java EE 的 Web 服务实现以及基于 BPEL 业务流构建的设计和技术实现进行了详细的分析。

第五章，方案在实际项目中的应用。结合作者对集成模型在实际项目运用中，论证了集成模型的可行性。

第六章，总结与展望。对全文工作进行总结，列举了论文工作的主要贡献，并且对进一步的研究提出了展望。

第二章 EAI 理论及技术基础

2.1 EAI 概述

2.1.1 基本概念

EAI 的概念在 IT 界提出和讨论已经有几年的历史了，最初大家谈到的 EAI 的概念，相对后来 EAI 的发展来看，可以说是一个狭义上的 EAI，正如其字面上的含义“Enterprise Application Integration”，即企业应用集成，仅指企业内部不同应用系统之间的互连，以期通过应用整合实现数据在多个系统之间的同步和共享。

随着 EAI 技术的发展，其内涵亦随之日益丰富，它已经被扩展到业务集成(Business Integration)的范畴，业务集成相对 EAI 来说是一个更宽泛的概念，它将应用集成进一步拓展到业务流程集成的级别。业务流程集成不仅要提供底层应用支撑系统之间的互连，同时要实现存在于企业内部应用与应用之间，本企业和其他合作伙伴之间的端到端的业务流程的管理，它包括应用集成，B2B 集成，自动化业务流程管理，人工流程管理，企业门户以及对所有应用系统和流程的管理和监控等方方面面[17]。

2.1.2 EAI 构成及各层技术实现

如何划分和规范 EAI 层次的定义，业界并没有一个统一的标准。当前从最普遍的意义上来说，比较宽泛的对 EAI 概念的理解是认为 EAI 可以包括数据集成、应用集成和业务流程集成等多个方面。

具体到技术层面上的划分，业界普遍认为一套完整的 EAI 技术层次体系应该包括应用接口层，应用集成层，流程集成层和用户交互层四个大的层面[17]。

（1）应用接口层

EAI 技术层次体系的最底层是应用接口层，它要解决的是应用集成服务器与被集成系统之间的连接和数据接口问题。其涉及的内容包括：企业应用系统适配器、Web 服务接口以及定制适配器等。

（2）应用集成层

应用集成层位于应用接口层之上，它要解决的是被集成系统的数据转换问题，方法是通过建立统一的数据模型来实现不同系统间的数据转换。其涉及的内容包括：数据格式定义、数据转换以及消息路由等。

（3）流程集成层

流程集成层位于应用集成层之上，它将不同的应用系统连接在一起，进行协同工作，并提供业务流程管理的相关功能，包括流程设计、监控和规划。其涉及的内容包括：业务流程管理（Business Process Management，BPM）、业务行为监控（Business Activity Monitor，BAM）以及企业间集成等。

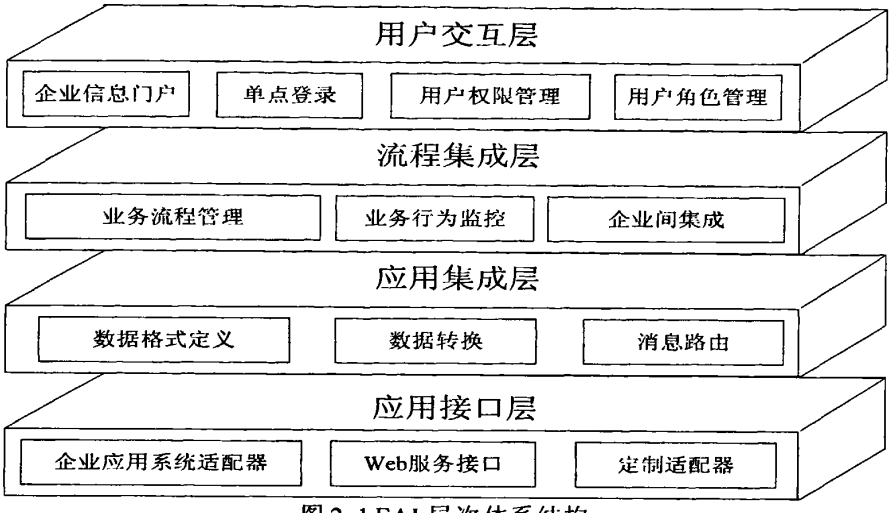

图2-1 EAI层次体系结构

（4）用户交互层

最上端是用户交互层，它为用户在界面上提供一个统一的信息服务功能入口，通过将内部和外部各种相对分散独立的信息组成一个统一的整体，保证了用户既能够从统一的渠道访问其所需的信息，也可以依据每一个用户的要求来设置和提供个性化的服务。其涉及的内容包括：企业信息门户（Enterprise Information Portal，EIP）、单点登录（Single Sign On）、用户权限管理以及用户角色管理等。

本文所研究的内容主要集中在应用接口层和流程集成层中，所讨论的集成方案主要针对于解决现有系统与企业遗留系统之间互连的问题，然后在此基础上实现一个业务流程的集成，旨在建立一个灵活的、标准的企业应用集成模型，以期给出较完整的EAI解决方案。

2.2 EAI实施特点及实现平台的选择

由于企业信息系统的建设是一个长期的过程，已经部署的系统是企业的重要资产，它已经让企业投入了大量的资金，而且企业应用集成也需要大量的资金与人力。系统之间集成的成本随着信息系统的建设在逐步攀升，当企业的第一个信息系统建设时，其成本为原始成本，当第二个信息系统建设时，其成本为原始成本加上与系统1集成的成本，当第三个系统建设时，其总成本为原始成本加上与系统1、系统2集成的成本。随着信息系统的逐步增加，集成的成本也越来越高。应用集成的费用的另一个影响因素是技术难度，并且平台差异越大，难度越高。仅从技术层面上考虑，异构平台集成成本远远高于同构平台，所以为了降低系统集成的成本，企业应该从整体上规划信息系统开发和部署的平台，减低EAI的实施成本与技术难度，上面是我们必须要考虑的EAI实施特点。

对于选择一个基于规范的、开放的、现代的、标准化的平台，用于实现企

业应用集成，目前的平台主要有SUN公司的Java EE平台，Microsoft公司的.NET平台[42]。

Java EE (Java Platform, Enterprise Edition)是SUN公司定义的一个开发分布式企业级应用的规范。它提供了一个多层次的分布式应用模型和一系列开发技术规范。Java EE 平台是支持企业分布式应用的主流平台之一，是一个基于开放式标准的企业应用集成平台，目前受到绝大多数的企业软件服务提供商的支持。使用 Java EE 平台作为企业集成服务平台，简化了基于组件的应用模型的开发。

.NET 框架 Microsoft 公司提出的下一代企业应用计算平台，它简化了在高度分布 Internet 环境中的应用程序开发。

选择二者之一 Java EE，作为企业 IT 基础架构主要从以下几个方面考虑[42]。

- 移植性和扩展性：.NET 目前只能用于 Windows 环境，Java EE 是跨平台，如果企业的 IT 核心设备是基于 Unix 的主机系统，那只能选择 Java EE，若是基于 PC Server 的中小型服务器，以在两者之间根据其它因素做出选择。
- 企业规模与成熟度：Java EE 在大规模高端系统的优异表现已经得到证实。.NET 是在 2002 年发布的，其高端性能有待考验。Java EE 是一个相当成熟的经过检验的企业应用平台。
- 架构：Java EE 另一个重要特征就是它的架构开放性，它已经得到了广泛的支持。而.NET 在设计时就紧紧把平台规范与产品联系在一起，只有很少的开放性保护已有的投资，企业必须投资新的商业需求，但是必须利用已有的企业信息化建设的投资，而 Java EE 架构可以充分利用用户原有的投资，它拥有了广泛的业界支持和一些重要的企业计算领域的供应商的支持，并且 Java EE 是跨平台的，它能在任何操作系统和硬件上运行，所以它能保留现有操作系统和硬件，并且在现有的应用系统进行升级开发和集成。
- 保护未来的 IT 投资：如今选择在何种平台上实现，对以后的进一步升级和维护有重要作用。Java EE 能让一个公司具有最强的适应变化的能力，而.NET 的可伸缩性和可移植性没有 Java EE 的性能强。

此外，Web 服务作为新一代面向服务的体系结构(SOA)软件技术的代表，它在 EAI 中与生俱来的得天独厚的优势。Web 服务不是 EAI 或者是 EAI 的一部分，Web 服务能够使 EAI 成为真正可行的、便捷实效的，同时引人注目的解决方案。

2.3 小结

本章首先对 EAI 的概念进行了阐述，并从技术层面对 EAI 进行了划分，然后从五个层面介绍一个完整 EAI 解决方案及其各层的技术实现，最后阐述 EAI 实施的特点以及实现平台的选择。

第三章 基于 Java EE 平台的应用集成方案

3.1 JCA 技术在 EAI 方案中的探讨

3.1.1 JCA 架构

Java EE 应用服务器是如今企业应用集成中使用的最广泛的一种集成平台。使用 Java EE 应用服务器作为集成 EIS 平台的关键是如何实现 EIS 与 Java EE 平台的连接以及如何在 Java EE 平台下开发和部署企业的业务逻辑。

JCA(Java Connector Architecture)是用来简化 Java EE 应用服务器和 EIS 之间的集成而提出的一种规范。它定义了将 Java EE 应用服务器连接到不同的 EIS 的标准体系结构，解决了人们在面对与 EIS 系统集成时面临的许多问题：

- 简化了集成。

在 JCA 出现之前，每个 EIS 应用都有自己的编程接口，不同的 EIS 系统之间以及应用服务器和 EIS 系统之间的交互都意味着要对自身进行修改从而针对一组特定的 API 编程。而 JCA 框架使得任何实现了 JCA 规范的应用服务器都可以通过实现了 JCA 规范的某一 EIS 的资源适配器与该 EIS 资源相连；同时，也可使任意 EIS 资源通过它们自身实现了 JCA 规范的资源适配器与同样实现了 JCA 规范的某一应用服务器相连[24]。

- 为连接提供了 QoS 管理。

JCA 框架利用 Java EE 应用服务器中内建的事务、安全等服务，为连接操作的事务及安全等特性提供了支持。

JCA 规范由以下三个部分组成：资源适配器、系统协议和通用客户接口[25]。JCA 定义了两套标准的接口：一套接口是用于让资源适配器把兼容的应用服务器无缝的整合起来；另一套接口(通用客户接口)允许客户端用一种统一的方法使资源适配器与 EIS 连接。资源适配器为 EIS、应用服务器和应用组件之间的交互提供连接。应用组件通过通用客户接口与资源适配器交互来访问 EIS。应用服务器和资源适配器一起提供对 EIS 访问时的连接、事务和安全的控制及管理。JCA 的体系架构如图 3-1 所示：

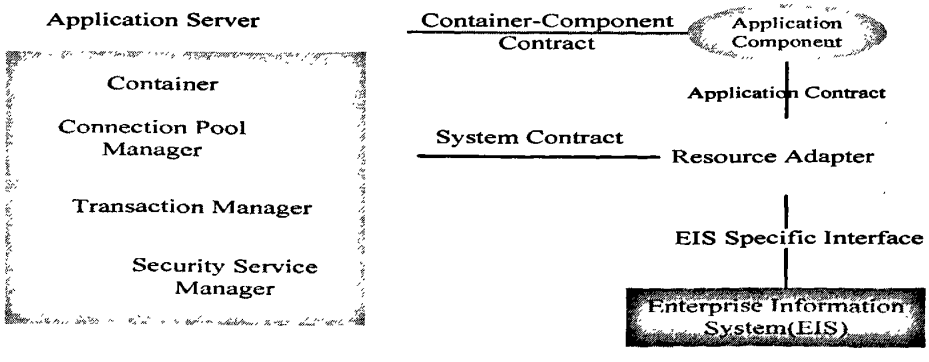

图 3-1 JCA 体系结构图

从图中可以看出，JCA 框架中主要定义了以下协议和元素[24, 25]：

- 系统级协议和服务：定义了 Java EE 应用组件、应用服务器和 EIS 系统之间的标准接口。这些协议和服务是由 Java EE 应用服务器提供者实现的，并且也位于 EIS 厂商提供的资源适配器中。这些协议和服务的实现在应用服务器与资源适配器的系统级角色和责任之间定义了一个逻辑划分，这样就使 Java EE 服务器和资源适配器能够彼此协作。不仅如此，它还使得一个遵守 JCA 规范的资源适配器可以插入到任何 Java EE 服务器中。
- 应用级协议和服务：定义了一组通用客户接口(Common Client Interface)，它是 Java EE 组件(如 JSP, EJB)与 EIS 系统连接或交互的一个客户 API。除了 Java EE 客户组件之外，它还允许非管理的应用程序(如 Java Applet 和应用程序客户端)使用一个遵守 JCA 的资源适配器与一个 EIS 集成。
- 资源适配器的打包和部署：允许各种 EIS 资源适配器插入 Java EE 应用服务器中。

EIS 通过资源适配器与 Java EE 应用服务器交互，资源适配器利用系统协议将与数据库连接等复杂的处理过程交给应用服务器来处理从而满足了可伸缩性、集成性和安全性方面的要求。系统协议包括了六个元素：连接管理，事务管理，安全管理，生存周期管理等。连接管理采用了连接池管理，从而使得应用服务器能够创建和共享 EIS 应用的连接，使得应用服务器能够更高效，更稳定地使用连接资源。事务管理可以让 EIS 应用能够获取应用服务器提供的事务环境的支持，使得服务器能够把 EIS 系统的事务作为一个事务单元管理，从而保证了数据的完整性。安全管理提供应用服务器到 EIS 的认证和机密性服务安全，维护应用服务器和 EIS 之间的安全通信并保证应用服务器到资源适配器间信息传递在有安全认证的安全环境下进行。

通用客户接口 CCI 是 JCA 定义客户端使用与企业信息系统交互的接口。客户端可以通过 CCI 来创建和管理与 EIS 资源的连接，执行对 EIS 资源的操作，管理输入、输出和返回的数据对象或数据记录。

JCA 的优点很明显，它为 EIS 厂商提供了一种按照开放的产业标准定义 EIS 接口的途径。通过使用公共的可调用接口以及继承 JCA 提供的 QoS 机制，程序员能够在不牺牲性能和系统完整性的前提下，简化 EIS 的集成工作。同时，JCA 建立在已经取得极大成功的 Java 语言之上，采用 JCA 技术能有效地进行软件复用，提高开发人员的效率，降低软件的开发和维护成本，提高软件的质量，控制系统的复杂性。

JCA 的局限性和缺点在于[43]：调用 EIS 应用时，JCA 采取同步消息传输方式，它不能处理来自 EIS 应用的异步消息或向 EIS 应用传递异步消息；JCA 没有提供基于 XML 的接口来实现与非 Java 的异构系统的集成。

3.1.2 JCA 的调用层次

前面已经说明了采用 JCA 架构能将各种 EIS 和 Java EE 应用服务器连接起来，在连接(集成)的过程中调用层次如图 3-2 所示。

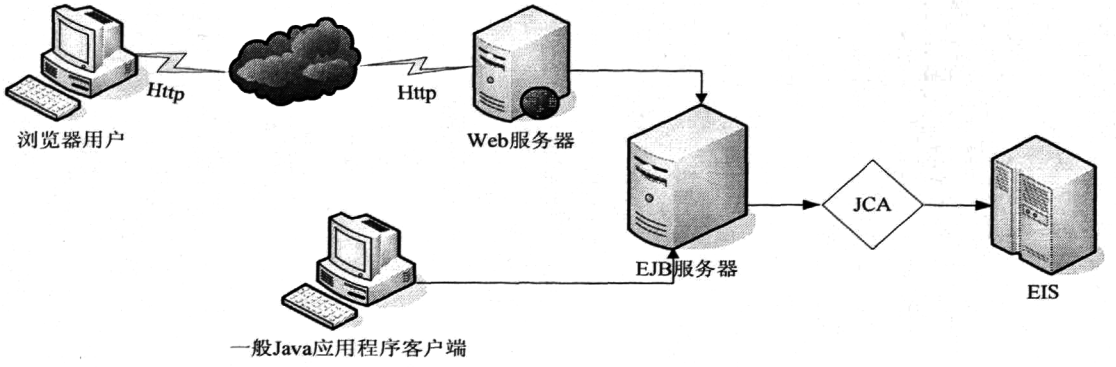

图 3-2 JCA 调用层次图

在这里，用户通过 B/S 或 C/S 访问 EJB，通过 EJB 调用 JCA 来连接 EIS。

3.2 Web 服务技术在 EAI 方案中的探讨

3.2.1 Web 服务

Web 服务[21]是一种新的面向服务的体系架构(Service Oriented Architecture,SOA)，指各个企业可以把和自己业务相关 API 发布到 Internet 上，并可被 Internet 上其它客户通过一定的协议和标准进行检索、调用以完成基于 Internet 的互操作，并且封装了实现细节。

Web 服务提供了一种分布式的计算技术，用于在 Internet 上通过使用标准的 XML 协议和信息格式来展现应用服务。使用标准的 XML 协议使得 Web 服务平台、开发语言和 Web 服务发布者能够互相独立，这个技术是 EAI 解决方案的一个理想的候选者。通过开放的 Internet 标准：Web 服务描述语言(WSDL，用于服务描述)[18]，统一描述、发现、集成规范(UDDI，用于服务的发布和集成)[22]和简单对象访问协议(SOAP，用于服务调用)[23]，Web 服务消除了现存解决方案中的互用性问题。

3.2.2 基于 Web 服务的 EAI

传统的 EAI 是一种紧耦合集成模式，比较适用于那些对性能要求较高的、需要多种层次集成的应用集成系统。

Web 服务是一种标准化的松耦合集成模式，比较适用于那些需要更大的灵活性、改动频繁的应用集成系统；同时，由于 Web 服务基于 XML 等开放协议，因此它能被广泛接受并向前兼容，潜在地消除了企业日后为支持新技术而需要进行二度投资的风险；再者，Web 服务的服务接口可以动态改变，因此可以实现一个动态的集成，这在不断扩大和变动的用户需求和商业应用的情况下，对企业实现其应用集成也有很大的益处。

结合 Web 服务的 EAI 系统则实现了一种面向服务层的松耦合的企业应用集成系统，可以最大限度的同时满足性能和灵活性的要求。另外，Web 服务是一

个正在快速发展的技术方法，它的不断改进和完善必将带来企业应用集成领域格局上的变化，当逐渐融合 Web 服务和 EAI 时，这种变化不仅对 Web 服务有利，对于 EAI 也是同样有利的。根据 IDC 公司的调查，Web 服务在未来将有非常大的市场，会带动软件、硬件和服务等各方面的发展，在未来 Web 服务的发展与竞争中，企业信息系统将更加灵活可靠、具有更强的可伸缩性，开发和部署也更加方便，企业将真正成为网络化的企业。

在 EAI 中使用 Web 服务技术，具有以下的优势：

- 简单性。相对于典型的 EAI 解决方案， Web 服务更便于设计、开发、维护和使用，它使得创建跨越多个应用程序的业务流程处理将变得相对简单。
- 基于开放协议标准。Web 服务基于开放标准，如 HTTP，XML，SOAP 及 UDDI，这个可能是导致 Web 服务被广泛接受的最重要的因素。事实上基于现存的开放标准可以节省企业的开发费用。
- 黑箱实现。Web 服务是黑箱操作，并且可以在不知道 Web 服务是如何实现(采用的是何种编程语言和平台等)的情况下被重用。
- 动态性。企业系统集成的过程中，若采用 Web 服务的方式来进行，则只需从服务提供者所公布出来的服务中选择适用的商业服务，并纳入企业流程中，即可达到系统集成的目的。相对于传统式的集成方法，Web 服务是以动态整合界面来集成系统，较具动态性。

基于 Web 服务的集成模型如图 3-3 所示。

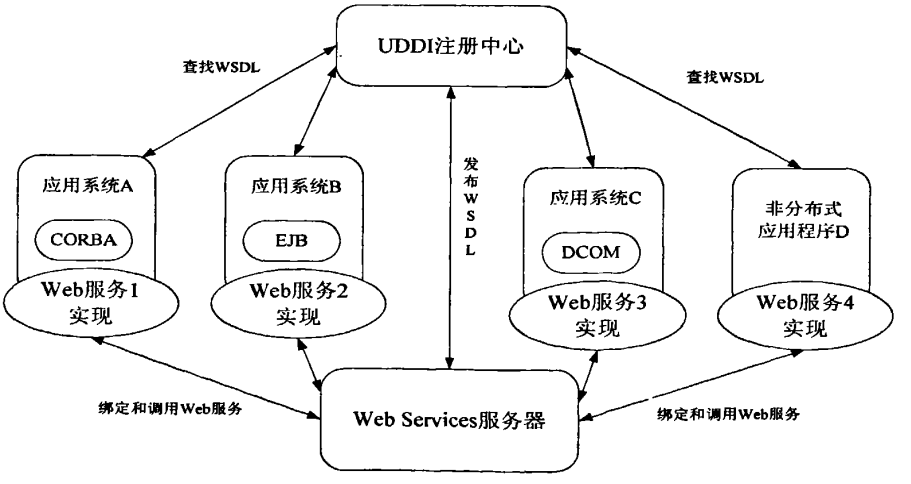

图 3-3 基于 Web 服务的应用集成模型

3.3 Web 服务和 JCA 在 EAI 中应用比较

Web 服务和 JCA 都是解决集成问题方面的标准，但它们却并不互相竞争。开发者和用户都需要知道这两者是不同的，并且需要知道这两种技术的差别所在，这一点很重要。

Web 服务被定位于用来标准化集成的技术，而 JCA 规范则用来标准化企业

应用集成（EAI）。其中 EAI 是广义上集成问题的一个子集。

本文将从以下几个方面对它们进行全面的比较：

1) 对遗留系统的侵入

侵入（Intrusion）是指为了支持集成技术而需要对遗留系统做的改动。

JCA：该技术需要很少或者不需要对遗留系统的侵入。JCA 适配器可以利用本地 API、套接字、数据访问以及其他不需要更改现有代码基础的技术来访问遗留系统。于是公司就可以利用在已部署系统中的投入，而不必再投入额外的资源来更新系统以支持集成。

Web 服务：为了让遗留系统能本地支持将 Web 服务用作 EAI 方案，必须需要很高程度的侵入。遗留系统必须被更新并且修改，以包含一个 Web 服务栈。这个栈将处理所有的 SOAP 消息分析和遗留系统调用，这是一项很庞大的任务。

2) 上下文传播

在应用集成中，需要考虑的一个严重的基础结构问题就是从使用遗留系统的客户到遗留系统本身的事务和安全上下文传播（Context Propagation）。

JCA：JCA 支持从 JCA 应用服务器到遗留系统的安全和事务传播。这是 JCA 设计中最主要的基础结构概念之一，也是被很多集成环境采用的一个强有力的原因。这意味着如果一个标识用户的 XA 事务上下文或者安全上下文是在应用服务器中创建的，那么当应用服务器调用遗留系统上的服务时，遗留系统就会感染上事务和安全上下文。就 XA 事务来说，这意味着如果遗留系统是 XA-compliant 的资源管理器，它就能参与 XA 事务。这样设计师们就能设计出 EAI 系统，这种系统能保证具有遗留系统以及使用该系统的客户的 ACID 事务。至于安全来说，这意味着用于个人的单点登陆（single sign-on）可用于客户和遗留系统之间。在设计和遗留系统的交互时，这些都是需要考虑的关键特性。

Web 服务：Web 服务可以将客户从服务器上分离出来，跨过 Web 服务消息传播 XA 事务上下文是有可能的，但 Web 服务的本性意味着谁也不能预测响应消息究竟何时发送或抵达目的地。XA 事务是一个寿命很短的事务，通常每 30 秒自动进行一次 transaction rollback。Web 服务的这种不可预测和不连贯的本性使它成为 XA 事务上下文不太现实的传输手段。系统级的安全标识符同样有这个问题。这意味着利用 Web 服务来访问远程系统的客户不能使用 XA 事务来实现同样的目的。

3) 代码绑定

代码绑定是 EAI 简单但重要的一面，可以将 Web 服务和 JCA 做一个比较。

JCA：JCA 是基于 Java 的技术，这并不意味着遗留系统必须是用 Java 编写的，但是使用 JCA 的应用必须用 Java 编写。

Web 服务：Web 服务 EAI 方案允许用任何语言编写的客户端都能访问遗留系统，这些客户端可能是用 Java，C#，Fortran，或者 COBOL 编写的。

4) 数据绑定

数据绑定用来将遗留系统数据的格式变成客户端应用可用的一种格式。

JCA：JCA 需要对一个 Java 数据类型（Java 类）的数据绑定，这对于在本地类型和 Java 类型之间有一个清晰映像的简单数据类型来说。但如果遗留系统有复杂的相关或者二进制数据必须被处理，那么很快就会有麻烦。JCA 1.0 没有一个表示遗留系统中数据格式的标准做法，并且动态客户端也不能连接到 JCA 适配器上以“发现”遗留系统的数据格式是什么。在 JCA1.5 版本中新增的 ManagedConnectionMetaData 接口允许客户端查询 JCA 适配器，以便动态地发现遗留系统中的数据形状和类型。

Web 服务：Web 服务给遗留系统带来了一个有趣的问题，因为遗留系统中的数据必须首先被转换成 XML 来传输，然后再转换成使用 Web 服务的客户端的编程语言格式。如果 EAI 系统使用客户端->Web 服务->服务器中间件->本地集成适配器->遗留系统这种途径，那么遗留系统中的数据将沿着一种不良的路径传输，即本地格式->本地集成适配器转换的语言格式->XML 转换->使用 Web 服务的客户端的语言格式，在这个过程中很可能造成数据的丢失（更不用说系统变慢这样显著的性能了）。

3.4 一种基于 Java EE 应用服务器的应用集成模型

3.4.1 设计思想

通过上面对两种技术的分析与应用比较，可以看到，基于 Web 服务的集成方案虽然带来许多好处，但是也可以看到其带来的一些问题。比如缺乏对于集成的管理机制，有些安全问题没有得到很好的解决，另外，还有 Web 服务对遗留系统的封装需要高度的入侵，以及对于许多没有提供访问接口的遗留 EIS，无法直接将它们展现为 Web 服务，而这些不足可以由 Java EE 规范来弥补，JCA 较好地处理了对遗留 EIS 进行集成时面临的许多问题，同时由于 Java EE 应用服务器在各种领域内创建了适用于企业需要的一系列开放的标准，绝大多数供应商产品的基础架构或部件都是遵循 Java EE 标准来开发的，因此企业不需要再为单独的应用程序平台投资，而且可以选择 Java EE 应用服务器中的组件来更好的满足自己的需要，使应用程序具有很好的扩展性和性能。JCA 使用 Java EE 平台中一些系统级的服务(如 JDBC、Java 消息服务、Java 事务 API 以及基于 Java 的安全技术)来实现其可扩展、安全和事务等机制，从而解决了 EAI 中的关键问题和需求。

把两种集成机制有效地结合起来，将 JCA 集成机制放在集成方案的后端，而把 Web 服务用在集成方案的前端接口部分，这样能使企业应用集成同时达到性能和灵活性的要求，既充分利用了 Java EE 平台的强大性能，又能以更加灵活和互操作性的方式将系统的业务功能展现给用户。因此，本文提出了一种基于 Java EE 应用服务器的应用集成模型，它把两种技术的优势相结合，在应用

集成领域具有更大的优势，不仅可以通过较低的成本解决企业中的信息孤岛问题，以及企业中系统更新方面的异构性问题，而且保护了早期企业在信息化建设中的投资，并能够适应在未来 SOA 的发展趋势中对面向服务的企业应用集成，因此这种新的应用集成方案对企业的发展具有很好的承上启下的作用。

3.4.2 集成模型的设计

3.4.2.1 需求分析

根据前面几章的分析，集成模型主要用来解决以下几个问题：

- 资源适配器如何实现对遗留 EIS 的访问；
- 应用组件如何以一种统一的方式获得不同的企业遗留系统的连接；
- 应用服务器可以为连接过程提供事务、安全等 QoS 管理；
- 对业务组件进行 Web 服务封装，实现 Web 服务的动态组合。

对于第一点，对没有提供 API，或者提供的 API 不适合需求的遗留 EIS，需要通过现有封装技术进行扩展，提供必要的接口；对于第二点，需要资源适配器端为应用组件提供统一的通用客户接口(CCI)的实现，当然也可以使用某类资源已经具有的公共 API，如 JDBC API， JMS API 等；对于第三点，需要应用服务器端和资源适配器端共同实现一组服务提供接口(SPI)，从而可以使用 Java EE 应用服务器本身所具有的功能为资源连接提供各种服务质量管理；对于第四点，需要构建业务流，通过业务流引擎，实现 Web 服务的动态组合。

3.4.2.2 集成架构

针对以上需求，本文提出的集成模型结构图如图 3-4 所示。从图中可以看出在集成模型中，EIS 遗留层，业务集成层，服务集成层，表示层构成了明显的层次关系，下层应用为上层提供了独立完整的功能，因此上层应用无需了解下层应用内部的实现细节，只需调用下层应用明确定义的接口和方法来实现自己的功能。

从上面集成模型结构图中可以看出，对于企业各种遗留的 EIS 系统是先通过 JCA 连接器把它们集成到 Java EE 应用服务器，也就是把遗留系统的业务功能封装成了应用服务器中的 Java EE 组件，然后通过 Java EE 平台中支持 Web 服务的各种接口将这些组件发布成服务在容器中调用。对于要集成的在 Java EE 平台上开发的应用系统，与集成遗留系统的差别就在于它不需要 JCA 连接器作为桥梁，而可以直接与 Java EE 组件兼容被集成到应用服务器中。

下面介绍该集成模型中各层的主要功能：

1) EIS 遗留层（EIS legacy layer）

EIS 遗留层：这层是企业中遗留的信息系统，它包括 CRM、SCM、ERP，还有其他的一些遗留系统，这些遗留系统有些已经提供了 API，有些没有提供 API，或者提供的 API 不适合需求，此时必须通过封装扩展现有应用，以提供必要接口，以支持对遗留系统的访问。

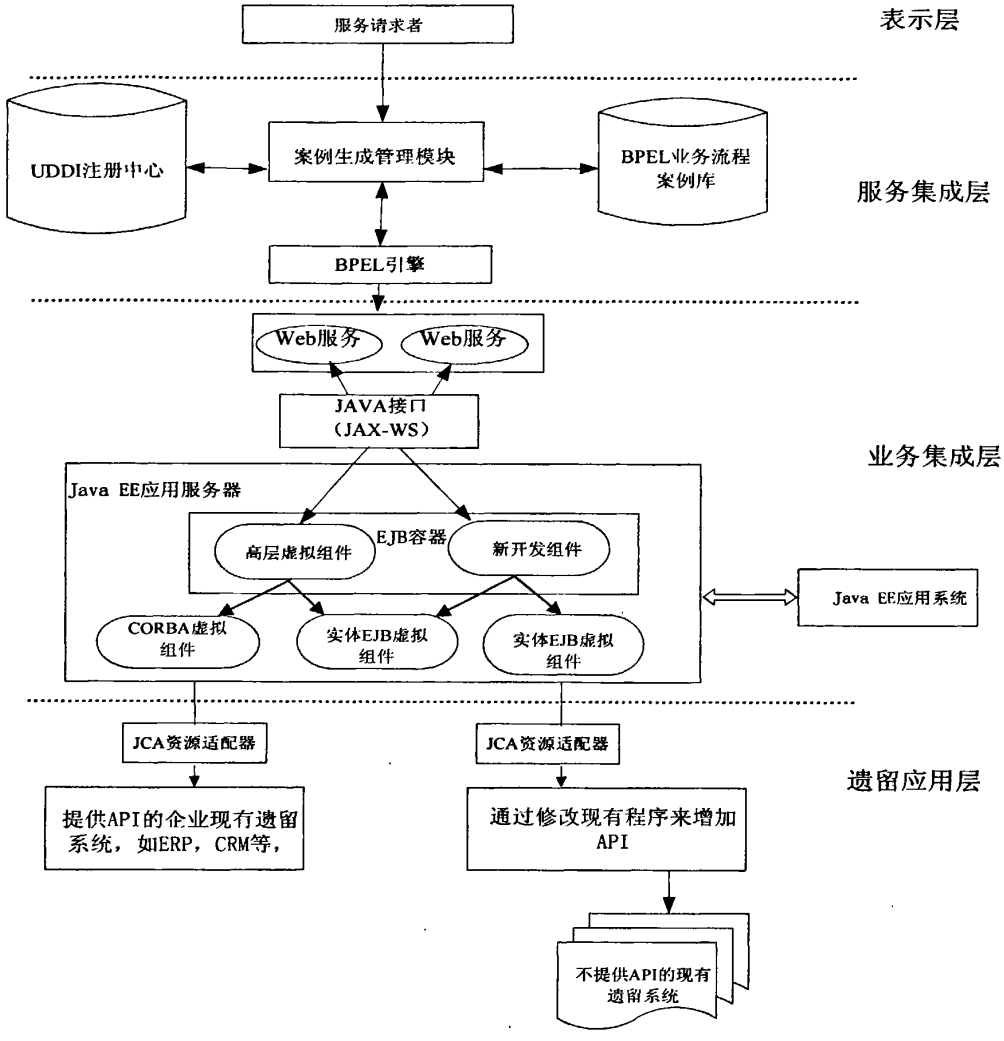

图 3-4 基于 Java EE 应用服务器的应用集成模型

2) 业务集成层（Integration layer）

主要包括：

运行环境/服务容器：即为该集成模型架构提供环境支持，常见的 Java EE 应用服务器有 Tomcat，JBoss，Weblogic 等，本论文后面选取 JBoss 容器作为运行环境。

JCA 集成环境：该集成运行在服务容器中，为应用程序提供如连接、事务、安全等服务，是 JCA1.5 规范的具体实现，也是本论文研究的重点之一。

Web 服务代理：这是该集成模型的另外一个重点，它是 JCA 集成框架的 Web 服务映射，用于包装企业应用，将企业应用以 Web 服务的方式进行表述，使得客户应用程序能够以 Web 服务的方式使用 JCA 集成框架提供的连接服务，事务服务和安全服务，并将描述文件 WSDL 发布到企业私有的 Web 服务注册中心。它通过 JCA 资源适配器获取企业信息系统所能提供的功能。论文后面将

有详细的描述，它同 JCA 底层集成一起构成了本论文的核心。

3) 服务集成层（Integration Tools layer）

该层位于业务集成层的上面，是连接表示层与集成层的桥梁，主要实现对流程集成的统一管理。本文在该层提出一个动态 Web 服务组合模型，增强了 Web 服务的业务组合能力，提高了流程的灵活性，增强了流程的可用性。

4) 表示层(Presentation layer)

主要以基于 Web 的客户端应用程序，它们利用 Web 服务集成层通过业务集成层同企业信息系统向连接，并在一个安全、稳定的环境中使用它们，达到企业应用集成(EAI)的目的。

3.4.2.3 操作流程

图 3-5 是根据本文提出的模型针对用户通过 Web 服务访问 Java EE 系统同时访问底层遗留系统，画出的具体操作流程图。在用户发出访问请求后，如果在 UDDI 注册中心中存在此服务，则通过相应的函数获得服务并调用，而后通过 Java 应用服务器中的 EJB 容器与 JCA 资源适配器交互，调用遗留管理系统中相应的功能模块，调用完毕，系统终止并退出。

①客户端向 Web 服务服务器发出服务请求；

②Web 服务服务器用 SOAP 消息向 UDDI 注册中心发出查询请求；

③如果注册中心注册了该服务，则注册中心将该方法 WSDL 描述返回给 Web 服务服务器，否则返回调用失败信息；

④Web 服务服务器用得到的 WSDL 描述生成 SOAP 请求消息，绑定服务提供者；

⑤SOAP 服务器通过监听器收到的 SOAP 请求，并转发给 Web 服务适配器。

⑥Web 服务适配器分析 HTTP 头信息，将其传递到指定的 EJB 容器高层 EJB 组件中。

⑦高层 EJB 组件将高层方法转换成一系列的对低层虚拟组件的调用，低层虚拟组件进行参数和数据类型的转换，并通过 JCA 资源适配器与遗留系统进行交互，遗留系统处理请求，并将结果返回给它的调用者低层虚拟组件，直到将结果传到 Web 服务提供者；

⑧Web 服务提供者将结果打包成 SOAP 消息返回至 SOAP 服务器；

⑨SOAP 消息由 SOAP 服务器返回到 Web 服务器；

⑩客户最终得到包含执行结果的 SOAP 消息。

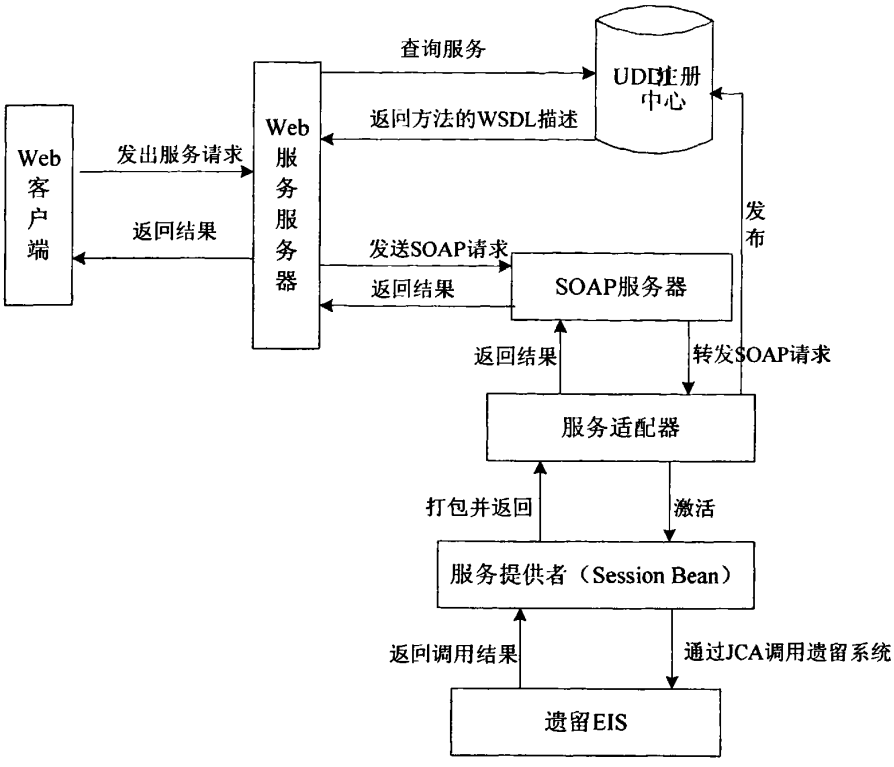

图 3-5 具体操作流程

3.5 小结

本章首先对 JCA 和 Web 服务技术应用 EAI 方案中进行了探讨，并比较了两种技术在 EAI 中的应用；然后研究了 JCA 技术和 Web 服务技术两者相结合，在集成企业遗留应用系统中的优势；最后将两种技术相结合，给出了一种基于 Java EE 应用服务器的企业遗留系统集成模型，并分析了该模型的设计思想。

第四章 集成模型中关键实现技术分析

本章将对集成模型中关键实现—基于 JCA 的 EAI 后端集成机制，基于 Java EE 的 Web 服务实现，基于 BPEL 业务流构建的设计技术实现进行详细的分析与介绍。

4.1 基于 JCA 的 EAI 后端集成机制

4.1.1 遗留 EIS 的改造

要集成没有接口的，或者提供的 API 不适合需求的遗留的 EIS，首先必须通过封装来扩展应用，即组件封装。封装提供了必要的接口，通过这些接口，Java EE 组件（如 EJB）可以存取这些遗留的 EIS。对遗留系统的封装，即对遗留系统的改造过程。对遗留系统的改造，采取白盒法进行。白盒法需要对遗留代码的内部工作机制进行研究。通过研究原程序模块，使得开发者掌握了基础业务流程，进而对它们重新编写代码，生成相应对象组件来为遗留系统提供 API。

现有的封装技术主要有 CORBA。CORBA 规范用抽象接口定义语言(IDL)描述接口，而接口可由任何语言来实现，这为组件封装提供了极大的方便。

经过封装的系统有了新的接口，而有的系统保留原有接口，这些不统一的接口带来了访问上的麻烦。为了屏蔽接口上的差异性，要进行下一步工作就是这些接口映射为统一的虚拟组件。JCA 资源适配器就充当了这个角色。虚拟组件在业务逻辑层表示现有应用的功能。一方面，它们展示与 Java EE 兼容的接口，另一方面它们通过现有应用程序提供的 API 和现有应用系统连接。虚拟组件掩饰了不同现有系统的技术差异，客户端可以通过虚拟组件来存取现有系统。在 Java EE 集成体系中建立虚拟组件可以使用以下技术：EJB，CORBA，JMS。

4.1.2 业务功能集成

前文已经提到，本文主要考虑业务功能集成，即考虑如何重用遗留的 EIS 的业务功能。完成以下 3 个目标：

①遗留应用程序的功能性利用；

②消除访问接口的差异性；

③把遗留系统的高级接口提供给用户。

上述的虚拟组件提供的接口是代码级的操作，如图 4-3 所示。而客户更需要的是展现应用系统功能的高级接口。为此还需要把这些低层虚拟组件组合成高级虚拟组件。EJB 组合虚拟组件的功能以提供一个实现较高级功能的抽象层。EJB 组件将会把高层的事务请求转换成一系列对低层虚拟组件的调用。EJB 组件也进行此数据转换，并且通过集成企业遗留应用实现企业的业务功能。

通过业务功能集成，用户可以在不熟悉现有应用系统的详细情况下调用其

业务功能。

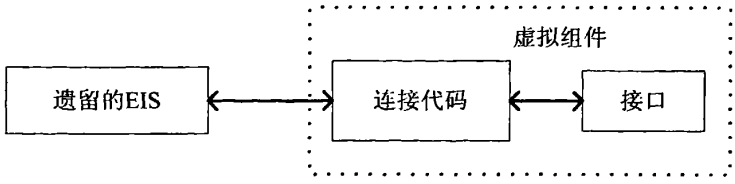

图 4-1 虚拟组件结构

4.1.3 虚拟组件对遗留 EIS 的访问

前面已经把 EIS 的接口映射为 Java EE 中的低层虚拟组件，现在考虑虚拟组件对 EIS 的访问问题。Java EE 平台提供了多种访问方式，如 JCA, JMS 和 RMI-IIOP 等。其中 JCA 除了能以统一的方式更加容易地实现对 EIS 的访问以外，还具有安全性和事务管理方面的优点。

使用 JCA，需要为每个 EIS 定制一个资源适配器并插入 Java EE 应用服务器中。资源适配器含有一个指定的 EIS 库，这个库同它描述的 EIS 具有连通性。JCA 还为 EIS 的访问定义了一个公共客户接口(CCI) 。Java EE 应用组件通过 CCI 访问 EIS，资源适配器通过 EIS 库将 CCI 调用转换为对 EIS 的调用。4.1.5 节将重点介绍低层虚拟组件对 EIS 的访问中的连接管理问题。低层虚拟组件对 EIS 连接后，就可以对遗留的 EIS 提供的业务功能进行访问，即相当于把遗留系统“移植”到 Java EE 平台上。这样就实现了将过去面向过程语言编写的遗留系统特定业务功能作为 Java EE 组件逐步“重新移植”到 Java EE 平台上，也可以把遗留系统中的某些模块映射和“迁移”到 Java EE 平台上的作为某一系统的表示层，这时就需要将遗留系统的“绿屏”终端窗口或 PC 的图形界面改用使用新的外观界面（如支持 Web 的瘦客户端）来替换，使遗留系统被统一集成到用户者的使用的系统界面中。需要注意是，对遗留系统的数据库的安全访问问题，可以采用数据库访问设置身份密码的方法，也可以配置基于角色的安全访问机制。

4.1.4 连接管理服务模块

连接管理模块是最核心的功能模块，它负责对 EIS 遗留层的连接和连接的管理。下面详细讨论连接管理模块的设计与实现过程。

4.1.4.1 设计模式的应用

1) 使用 Adapter 模式统一连接接口

JCA 首先面临的就是要解决统一连接的问题。由于每一个 EIS 资源都有自己的编程接口，因此无论是客户端的应用组件还是位于中间层的应用服务器，与一个异构的 EIS 交互都意味着要针对一组特定的 API 编程。JCA 中两组标准的接口，通用客户端接口(CCI)和服务提供方接口(SPI)分别定义了客户端应用组件与 EIS 资源之间，以及应用服务器与 EIS 之间的统一连接。那么所面临的问题可以归结为以下：

①在现有 EIS 接口已不能满足系统需要的情况下，仍要使用现有的接口访问 EIS 系统。

②要建立新的类，且使得这些类可重复使用，即任意满足 JCA 规范的应用服务器都能与任意满足规范的 EIS 进行互联，通用客户端也能使用 CCI 请求任意满足规范的 EIS 连接。

显然，这与 Java 中应用 Adapter 模式[26]所面临的问题一样，Adapter 模式的静态结构图。

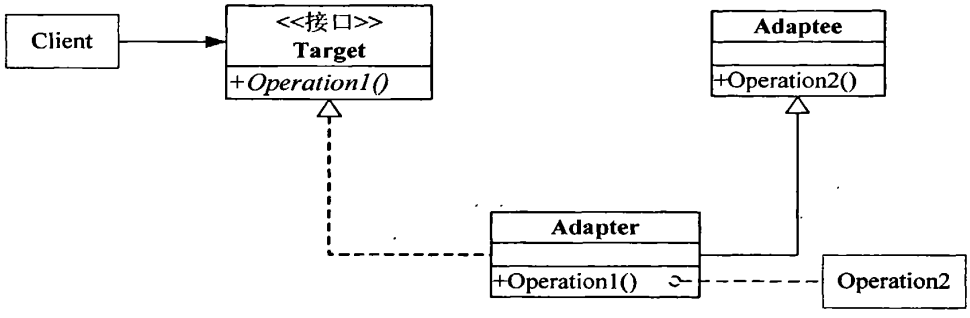

图 4-2 Adapter 模式

Adapter 模式是一种典型的结构化模式，其主要原理就是通过改变对象的接口，将类的接口改变成客户所期望的接口，从而使那些不能兼容的类可以一起工作。在该框架的底层架构中，为不同的 EIS 资源开发出特定于其自身的资源适配器，在该资源适配器中来实现可供应用服务器及应用组件访问的标准接口。这种做法也很像货物包装的过程，被包装的货物的真实样子被包装所掩盖和改变，因此有人把这种模式叫做包装（Wrapper）模式。在第五章该集成模型的实际应用中，实现了人事工资管理信息系统资源适配器，就使用 Adapter 模式对其进行包装，如图 4-3 所示。

2) 使用 Bridge 模式屏蔽底层差异

虽然通过使用 Adapter 模式，将不同的 EIS 接口统一改装成了客户和应用服务器所期望的接口，但是各 EIS 系统在内部的实现上是不尽相同的，为了增强系统的灵活性和可重用性，就应当为客户端逻辑屏蔽底层实现的差异，针对顶层共同的业务逻辑抽象编程而不是针对具体实现编程[26,27]。在 Bridge 模式[28]中，Abstraction 对象定义了客户对象访问的接口，隐藏了 Implementon 对象后面的实现细节，如图 4-4 示。

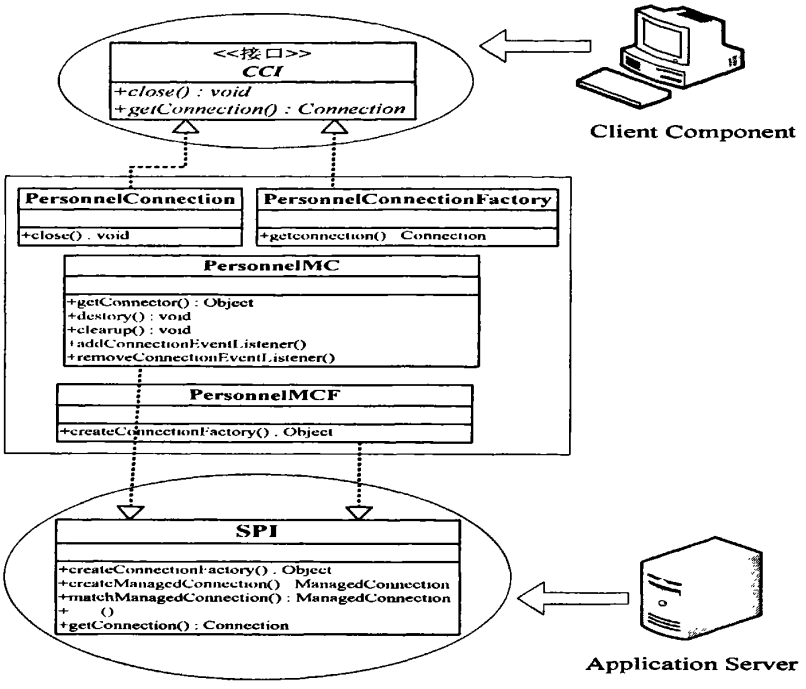

图 4-3 Adapter 模式封装资源适配器

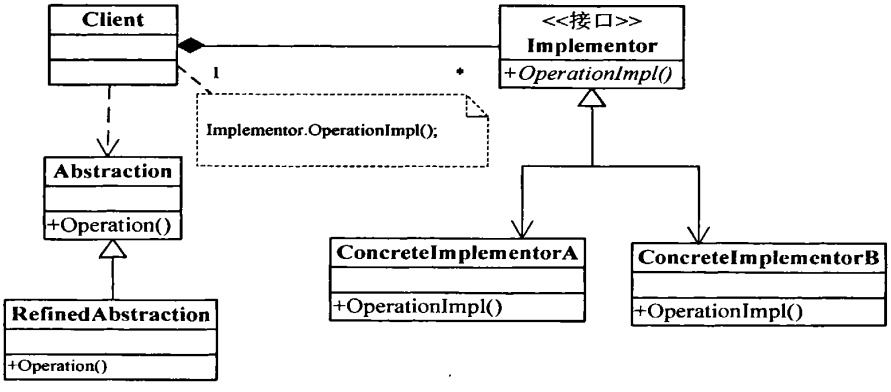

图 4-4 Bridge 模式

使用 Bridge 模式就能将资源适配器接口和它的具体实现分开。应用程序可以针对任何 EIS 资源发出连接指令，因为应用程序并不直接与底层的 EIS 系统打交道，而是通过 CCI 接口进行访问。CCI 接口就相当于模式中的 Abstraction，它通过连接管理器访问资源适配器的接口。资源适配器则负责底层的连接工作。由于使用了 Bridge 模式，应用程序可以不依赖于资源适配器的细节而独立变化，同时资源适配器也可以独立于应用程序的细节而独立演化。两个独立的等级结构如图 4-5 所示，左边是 Client API 的等级结构，右边是 Resource Adapter 的等级结构。应用组件是建立在 Client API 的基础之上的。

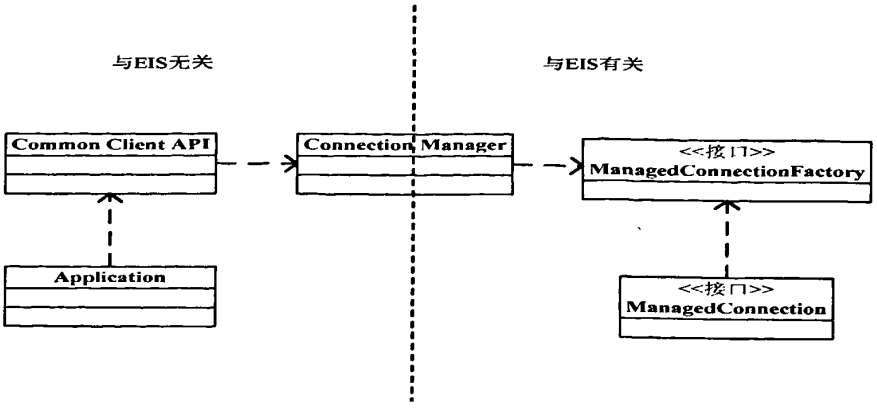

图 4-5 Bridge 模式的应用

4.1.4.2 连接管理的实现

1) 连接管理过程

在使用连接管理情况下，应用组件和 JCA 以及 EIS 之间的交互关系图如图 4-6 所示。

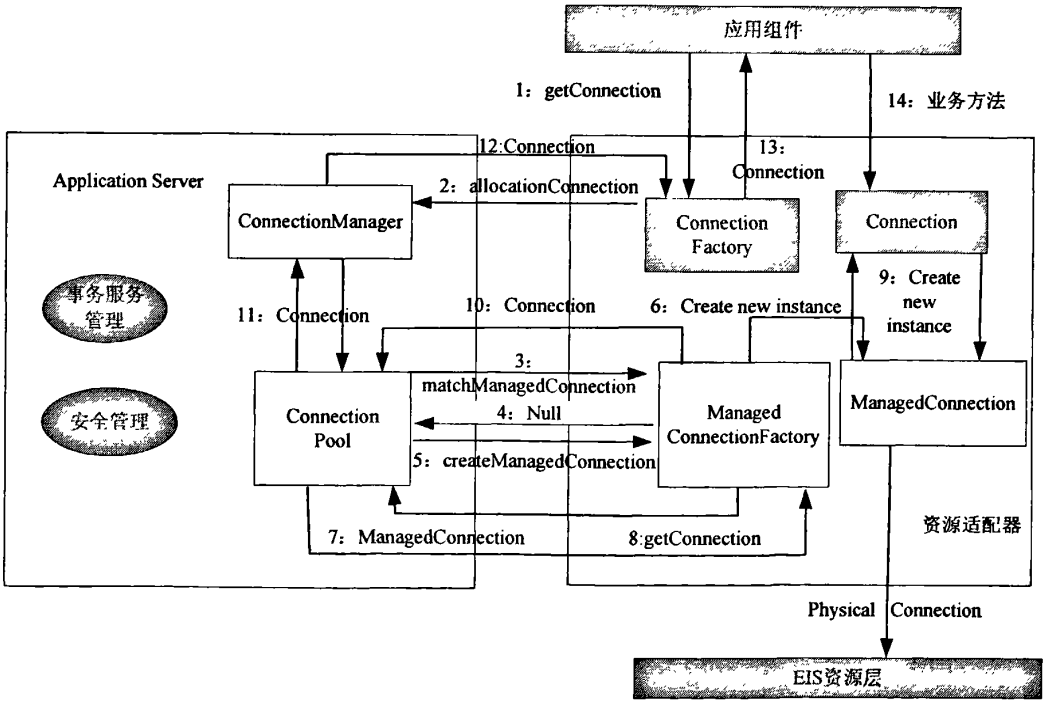

图 4-6 应用组件和 JCA 以及 EIS 之间交互图

下面简要的叙述一下整个过程的内容，它是典型的一个应用程序连接 EIS 资源的过程：

- 应用程序组件通过 JNDI 查找到 ConnectionFactory；
- 应用程序组件发出获得连接的请求；
- 连接工厂调用 ConnectionManager 的 allocateConnection;

- 连接管理器向连接池管理器发出获得连接的请求；
- 连接池管理器试图从 ManagedConnectionFactory 进行连接匹配，如果没有匹配到连接，那么返回 null；
- 由于没有匹配到连接，连接池管理器调用 ManagedConnectionFactory 的 createManagedConnection 方法来创建连接；
- ManagedConnectionFactory 接收到连接池管理器的请求后，创建一个 ManagedConnection 实例，同时 ManagedConnection 打开和 EIS 之间的物理连接，然后把这个 ManagedConnection 实例返回给连接池管理器；
- 连接池管理器调用 ManagedConnection 实例的 getConnection 方法以获得一个 Connection 实例；
- ManagedConnection 实例收到连接池管理器的 getConnection 请求后，创建一个 Connection 实例，然后把这个实例返回给连接池管理器；
- 这个 Connection 实例通过连接池管理顺次返回给应用程序组件；
- 应用程序组件通过返回的 Connection 来创建 Interaction 或者调用业务方法；
- 应用程序组件通过 Connection 调用业务方法时，实际上 Connection 使用了 ManagedConnection 的物理连接和 EIS 进行交互。

2) 连接池实现

为了完成上述的资源连接管理过程，通常需要应用服务器端和资源适配器端共同实现一组 JCA 规范中定义的服务提供接口(SPI)。对于连接池，规范中并没有做硬性的规定。但是，通常连接的创建和销毁都是复杂而又费时的工作，因此，有必要对连接进行复用，使得连接使用完毕后并不是直接删除，而是返回连接池供同样的连接请求再次使用。连接复用的好坏从很大程度上影响了系统的性能。服务器端有必要提供连接池的具体实现，且对于客户端应用组件来说，对连接池实现与管理全都是透明的。集成模型中连接池的类图如图 4-7 所示。

JbossResourcePool：连接池管理器的实现类，是以应用服务器服务的形式实现并启动的。通过它可以从配置文件中获得连接池的各种属性，如最大最小连接数目等。通过 JbossResourcePool 可以决定创建何种类型的连接池。其中的 WholePool，PoolByCri，PoolBySubject，PoolBySubjectAndCri 代表了不同种类的具体连接池，它们是作为 JbossResourcePool 的内部类实现的，统一继承 BasePool，并从 JbossResourcePool 得到连接池的配置属性。

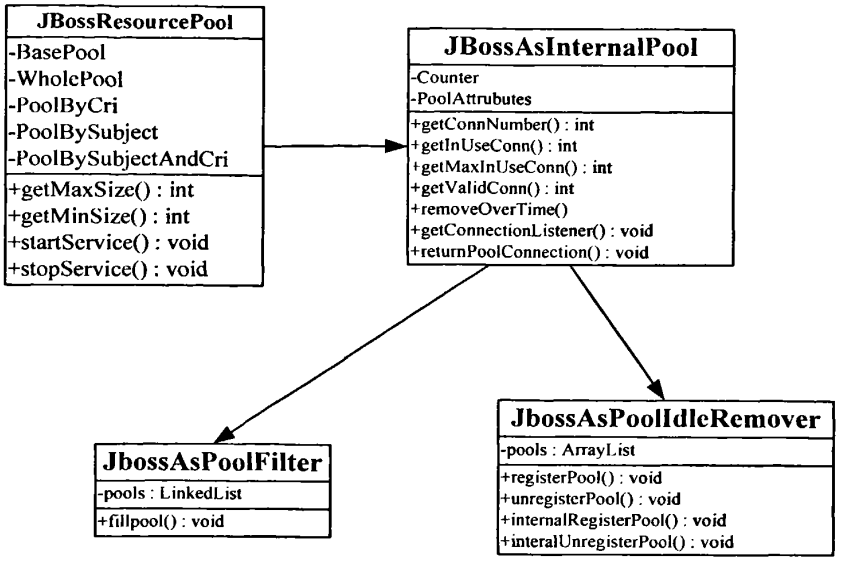

图 4-7 连接池实现类图

为了加快连接查找的速度，通常要对连接池进行划分，将具有相同属性的连接放在一个连接池当中。为此有四种划分连接池的标准，可以在JbossResoureePool 中使用常量指定，也可以在配置文件中指定，以下的四种标准分别与上面提到的四种连接池相对应。

- ByNothing：不对连接池进行划分，所有的连接对象都放在同一个连接池中。
- ByApplication：依据应用组件传递的连接请求信息对连接池进行划分。
- ByContainer：依据应用服务器中的安全配置信息对连接池进行划分。
- ByAppAndContainer：使用应用服务器中的安全配置信息和应用组件请求连接时传递给应用服务器的安全认证信息同时对连接池进行划分。

显然划分粒度越细连接匹配的速度就越快，但逻辑处理上会略显复杂，中和二者的情况，在实现中，可以选择 ByContainer 作为连接池划分的标准。

JbossAsInternalPool：从组件复用的角度考虑，使用 JbossAsInternalPool 对上述不同种类连接池中共有的属性及方法进行处理。连接池的具体实现中，包括最终从池中找到合适的物理连接以及对连接的返回，对利用率不高的连接移除等均由 JbossAsInternalPool 来完成。

JbossAsPoolFiller：根据连接池的划分方法，对于不同的 EIS 资源可能就要创建不同的连接池，通常需要将这些连接池对象维护在某种数据结构中以对其进行统一的操作。JbossAsPoolFiller 专门用于管理连接池添加的，当有新的连接池创建时它便在所维护的 LinkedList 类型的 pools 变量的末尾添加一个元素。由于不需要对元素进行定位，只需要简单的增加一个元素，所以选择 LinkedList 数据结构进行实现，便于添加和删除。

JbossAsPoolIdleRemover：用于移除空转的连接池对象，由于需要对连接池

对象进行遍历，因此选择 ArrayList 结构来维护连接池元素组 pools。由于牵涉到对配置中所要求的连接池总体最大最小连接数目的改变，所以当个体连接池中的连接数目发生改变时，如连接的删除，创建等，都应到 pools 连接池元素组中进行更新，而完成这一更新的就是 JbossAsPoolIdleRemover 类中实现的 registerPool，unregisterPool 方法。

4.2 基于 Java EE 的 Web 服务实现

由于该集成模型仍然是一个面向服务的集成模型，所以连接到 Java EE 应用服务器平台的应用组件可以被封装成 Web 服务，供 Web 服务容器调用，以适应在未来 SOA 的发展趋势中对面向服务的企业应用集成。

4.2.1 Java EE 对 Web 服务的支持

Java EE 作为一个可扩展平台，及时地加入了 Web 服务特性。SUN 公司在 J2EE 1.4 平台上实现 Web 服务，是通过 JAX-RPC API 提供了完整的 Web 服务支持，这种 API 支持基于 Servlet 和企业 Bean 的服务端点。JAX-RPC 是基于 WSDL 和 SOAP 协议提供了与 Web 服务的互操作性，它是 Java 在 Web 服务上的尝试[29, 30, 31, 32]。继 J2EE1.4 之后，作为主流中间件技术标准的 J2EE 推出了又一个新的版本—Java EE 5[33, 34]，Java EE 5 融入了 Java 5 的新特性，简化了应用的开发过程，这也是 Java EE 5 与之前 J2EE 产品的最显著区别。Java 5 中最重要的一个新的特性就是标注(Annotation)。标注提供了一种机制，将程序的元素如：类，方法，属性，参数，本地变量，包和元数据联系起来，这样编译器可以将元数据存储在 Class 文件中。这样虚拟机和其它对象可以根据这些元数据来决定如何使用这些程序元素或改变它们的行为。通过在源代码中使用 Web 服务标注，Web 容器将能够在无需满足任何其他开发要求的情况下发布 Web 服务，通过引入标注，很大程度上降低了 Java EE 5 开发成本。

JAX-WS[35, 36]作为 JAX-RPC 的后续替代者，具有更好的依托于 Java 5 的一些新的特性提供了更好的性能和更简单的服务器端和客户端开发模型，它同时支持新的 WSDL 和 SOAP 规范，虽然当前大部分的供应商还没有提供良好的支持，但是随着 Java EE 5 的深入，它必定会迅速的替代 JAX-RPC 成为一种新的 Web 服务开发模型。

虽然 J2EE 1.4 平台提出了相对完善的运行环境、大量的应用程序接口以及调度描述符号，但作为 Java EE 框架来说它更重要的关注点是比较抽象的底层业务逻辑代码，而且 Web 服务在开发和部署的过程中是十分的复杂和繁琐。在现有的技术条件下，如何简化 Web 服务的开发过程，最大程度上应用现有平台的特性以适应 Web 服务的变化，实现简便、动态的 Web 服务封装、发布以及部署过程。带着这个问题，本文研究了在使用 Java EE 5 新特性的 Web 服务开发过程中，如何实现基于 JAX-WS 模型的 Web 服务开发，以及在引入 Annotation 后如何应用这一特性简化 Web 服务的服务端封装，构建发布和客户端的部署问

题。

4.2.2 基于 JAX-WS 标准进行 Web 服务开发

4.2.2.1 JAX-WS 标注

JAX-WS 中广泛的使用标注，由于最初的 Web 服务标注是基于 JAX-RPC1.1 的，为了更好的实现对 Web 服务开发的简化，JAX-WS 定义了一些不包含在 JSR181 中的标注，尽管这些标注很多情况下不是直接使用，但是它们可能会被 JAX-WS 工具来生成。

BindingType:@BindingType

用来指定 Web 服务端点实现使用的规范，默认为 SOAP1.1/HTTP，它只有一个属性 value 用来指定 Web 服务相应的 URI。例如：@BindingType (value="http://www.w3.org/2003/05/soap/bindings/HTTP/")

RequestWrapper:@RequestWrapper

在服务端点接口中指定运行时请求包装 Bean。当选择从 Java 类开始构建 Web 服务的时候，被用来解决 Document literal 模式下的重载冲突，这种情况下只需要指定 className。

ResponseWrapper: @ ResponseWrapper

在服务端点接口中指定运行时响应包装 Bean。当选择从 Java 类开始构建 Web 服务的时候，被用来解决 Document literal 模式下的重载冲突，这种情况下只需要指定 className。

WebEndpoint:@WebEndpoint

用来声明客户端访问 Web 服务时候采用的 getPortName 采用的方法名称，对应于 WSDL 文件中的 Services 的 port 名字。

WebServiceClient:@WebServiceClient

这个标注中包含的信息足够唯一的识别 WSDL 文档中定义的 wsdl:service，wsdl:service 元素呈现生成的 Web 服务信息发送给客户端视图。Name 代表 WSDL 文件总的 wsdl:servicesName 的本地 name 信息，targetNamespace 代表 wsdl 文件中的 wsdl:servicesName 的本地命名空间信息，wsdlLocation 指定定义这个 Web 服务相关的 WSDL 文件的位置。

WebServiceProvider:@WebServiceProvider

用来标注一个实现 Provider 的类，targetNamespace 表示从这个 Web 服务的 WSDL 的 XML 命名空间，大部分 XML 元素命名空间依照 JAXB 映射规则。serviceName 对应 Web 服务中的 wsdl:service 的服务名字，portName 对应 wsdl:port Name，wsdlLocation 标明 WSDL 文件的位置。

WebServiceRef:@WebServiceRef

用来定义一个 Web 服务参考和注入目标。包括 JNDI 名字，Java 类型，产品特定名字，提供服务的类，WSDL 位置等。

FaultAction:@FaultAction

用在一个 Action 标注内部用来显式的声明 Action 寻址错误信息，@javax.xml.ws.Action(fault=@javax.xml.ws.FaultAction(className=AddNumbersException.class，value="http://example.com/faultAction")

在相应的 WSDL 文件中就会生成对应的错误信息描述

<fault message="tns: AddNumbersException" name="AddNumbersException" wsaw:Action="http://example.com/faultAction"/>

4.2.2.2 基于 JAX-WS 的 Web 服务开发过程

在 JAX-WS 中，Web 服务的操作调用都通过基于 XML 协议（如 SOAP）来呈现，SOAP 规格说明定义了信封的构成、编码规则和约定的 Web 服务调用和相应协定。这些调用和相应协定都通过基于 HTTP 的 SOAP 信息来传送。

尽管 SOAP 信息是复杂的，但是 JAX-WS 已经通过很多手段来对开发人员隐藏它的复杂性。在服务器端开发人员通过 Java 定义的接口来指定 Web 服务操作，或者需要更多的类来实现这些方法。客户端创建一个代理，本地的方法表现远程服务，在代理上调用这些方法。JAX-WS 中，开发人员已经不需要再去解析或者生成 SOAP 信息，通过 JAX-WS 的运行环境去从 SOAP 信息负责转换 API 调用和相应。通过 JAX-WS 附带的一些有用的工具，可以自动生成客户端和服务端所需要的一些运行所需文件。

开发 Web 服务终端可以选择从 Java 开始也可以选择从 WSDL 开始，WSDL 文件描述了 Web 服务终端和客户端的所有联系。一个 WSDL 文件可能包含或者导入用来描述 Web 服务所使用的数据类型的 XML Schema 文件。从 Java 类开始的模型，JAX-WS 工具生成规范定义的相关的 WSDL 等部署描述文件。从 WSDL 文件和 Schema 模型开始，JAX-WS 工具来生成服务需要的 POJO 以及终端接口。实际这两种开发模型有一种平衡，如果从 Java 开始开发，可以确保服务终端的实现具有希望的 Java 数据类型，但是开发者缺乏控制自动生成的 XML Schema。如果从 WSDL 文件和 XML Schema 开始开发，开发人员可以完全控制所需要的 XML Schema，但是却不能控制生成的服务终端和终端中包含的 Java 类。

现在最主要的开发 JAX-WS 模型有两种[36]，一种是从 Java 代码开始开发生成 WSDL 文档，称为"自底向上"模式。一种是从 WSDL 文档开始生成相关 Java 代码，然后填充 Web 服务具体接口的终端实现，称为"自顶向下"模式。本文采用"自底向上"开发模式，下面就对这种开发模式的实现过程进行一定的研究。

"自底向上"开发模式实际上将 Java 底层代码作为 Web 服务进行公开，这种情况下首先需要定义 Web 服务终端所需要的具体实现，包含相应的 JavaBean，在 JAX-WS Web 服务开发中服务终端接口不再是必须的。这种模型下可以迅速

的将遗留应用系统作为 Web 服务公开发布，同时开发人员可以最大限度的不需要掌握 WSDL 具体的细节描述，WSDL 文档将通过 JAX-WS 工具来生成，甚至不需要显式的生成 WSDL 文件，运行环境会自动生成相应的 WSDL 文件。

从 Java 开始进行基于 JAX-WS 的 Web 服务开发在服务终端的实现类方面有一些约束和限制，有效的服务终端类实现必须满足一定的要求：

- 必须要有 Java 类标注了 javax.jws.WebService 标注，它定义 Java 类为 Web 服务端点。通过在服务实现类中的@WebService 标注添加一个 endpointInterface 元素可以用来显式的指定该实现的接口，这样就必须定义一个接口包含实现类中所有可用的公用方法；
- 服务实现类必须被定义为 public 的，而不能被定义为 final 或者 abstract，实现类中必须有一个默认的 public 构造函数，不能构定义 finalize 方法；
- 实现 Web 服务的 Java 类的任何一个方法可以装载一个 javax.jws.WebMethod 标注来声明该方法为 Web 服务的一个操作；
- 所有的方法除了可以抛出服务自定义的例外，还可以抛出 java.rmi.RemoteException；
- 所有的方法的参数和返回类型必须和 JAXB 2.0 Java 到 XML schema 映射中的定义相一致；
- 所有的方法参数和返回类型不能直接或者间接实现 java.rmi.Remote 接口；

基本的开发模型流程为：首先在要暴露的类中添加@WebService 标注，声明该类用来暴露为 Web 服务，然后依照 JSR 181 和 JAX-WS 中的定义添加相关的标注。

这里要将 Web 服务作为一个 Web 应用程序发布，所以还需要在 JAX-WS 部署描述文件 sun-jaxws.xml 中注册添加的服务终端类和映射。如下所示：

```
<? xmlversion="1.0", encoding="UTF-8">
<endpoint
        version="2.0"xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime
        ">
    <endpoints implementation="com.edu.ws.basic. getUserInfo"
     Name="getUserInfo"   url-pattern="/ getUserInfo"/>
    </endpoints>
 </endpoint>
```

在 web.xml 添加监听器

com.sun.xml.ws.transport.http.servlet.WSServletContextListener 来解析 sun-jaxws.xml 中注册的服务终端实现，创建一个 Web 服务终端实现集合结果集，然后按照 sun-jaxws.xml 中的声明 URL 映射为每个服务终端声明一个 servlet URL 映射 com.sun.xml.ws.transport.http.servlet.WSServlet。

```
<listener>
    <listener-class>
     com.sun.xml.ws.transports.http.servlet.WSServletContextListener
    </listener-class>
</listener>
<servlet>
<servlet-name> GetUserInfoWS
</servlet-name><servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet>
<servlet-name> GetUserInfo
WSService</servlet-name><servlet-class>com.sun.xml.ws.transport.http.servlet
.WSServlet</servlet-class>
<load-on-startup>1</load-on-startup>
 </servlet>
    <servlet-mapping>
    <servlet-name> GetUserInfoWS</servlet-name>
        <url-pattern>/ GetUserInfoWS </url-pattern>
        </servlet-mapping>
            <servlet-name> GetUserInfoWSService</servlet-name>
            <url-pattern>/ GetUserInfoWSService</url-pattern>
    </servlet-mapping>
    </servlet>
```

到此得到了 Web 服务终端实现：GetUserInfoWS；JAX-WS 部署描述文件：sun-jaxws.xml；Web 应用描述文件：web.xml。就完成了一个简单的 JAX-WS Web 服务“自底向上”的服务端开发。余下的工作通过 Ant 脚本自动化完成。

首先创建 Ant 部署文件，创建一个任务：利用 apt 或者 JAX-WS 中附带的工具包 wsgen 依照 Web 服务终端实现类中添加的 web 服务标注自动生成 Web 服务所需要的其它的组件。JAX-WS 中提供 wsgen 和 apt 在 ant 中运行的实现任务类。通过下面的方法声明任务定义来调用 wsgen 和 apt：

```
<taskdef name="apt" classname="com.sun.tools.ws.ant.Apt">
        <classpath refid="jaxws.classpath"/>
</taskdef>
<taskdef name="wsgen" classname="com.sun.tools.ws.ant.WsGen">
        <classpath refid="jaxws.classpath"/>
</taskdef>
```

然后按照 wsgen 和 apt 中定义的属性和元素生成 Web 服务所需要的其它依赖类和所需要的 WSDL 文件。之后通过 Ant 中提供的 war 任务将所需要的文件打包成 war，发布到实现 Servlet 2.4 规范的 Web 容器中（如 JBoss），然后通过浏览器就可以测试部署的应用程序是否成功，这里通过 http://localhost:8080/UserBasic/GetUserInfoWS?WSDL 就可以得到 Web 服务的 WSDL 文件，证明 Web 服务发布成功。

4.3 基于 BPEL 的 Web 服务合成

虽然 Web 服务技术为跨平台的应用提供了一种使用标准协议进行交互的能力，但它的规范中没有提供对业务语义的定义，各 Web 服务之间是孤立和透明的。而在复杂的商业应用中，业务模型通常都要涉及双方或多方的、长期运行的、有状态的、使用同步或异步方式通信的消息交换序列。在这种情形下，简单 Web 服务已经不能满足要求，因此需要采用专门的服务合成语言来描述，Web 服务业务流程执行语言就是在此基础上产生，其将每个 Web 服务看成一个协作者，每个协作者又提供不同的业务接口，通过对这些不同的业务接口的连接从而获得复杂的业务流程[37]。

目前 Web 服务合成是工业界和学术界研究的热点问题，最近结构化信息标准促进组织(Organization for the Advancement of Structured Information Standard ,OASIS)提出 Web 服务业务流程描述语言 WS-BPEL (Web Service Business Process Execution Language)，简称 BPEL[38]，是由 IBM 和 BEA 联合定制的，结合了 IBM 的 WSDL 和 Microsoft 的 XLANG 这两个规范的优点提供了描述商业过程的定义语言，使快速、简单的 Web 服务合成成为了可能。

4.3.1 WS-BPEL 介绍

作为服务合成可执行流程的实现语言，WS-BPEL 的作用是将一组现有的服务整合起来，从而定义一个新的 Web 服务。因此，WS-BPEL 基本上是一种实现这样的整合语言。与其他任何 Web 服务一样，整合服务的接口也被描述为 WSDL 的 portType 的集合。整合，即流程，指明了服务接口与整合的总体执行的配合情况。图 4-8 说明了从外部看到的 WS-BPEL 流程的上述情况。

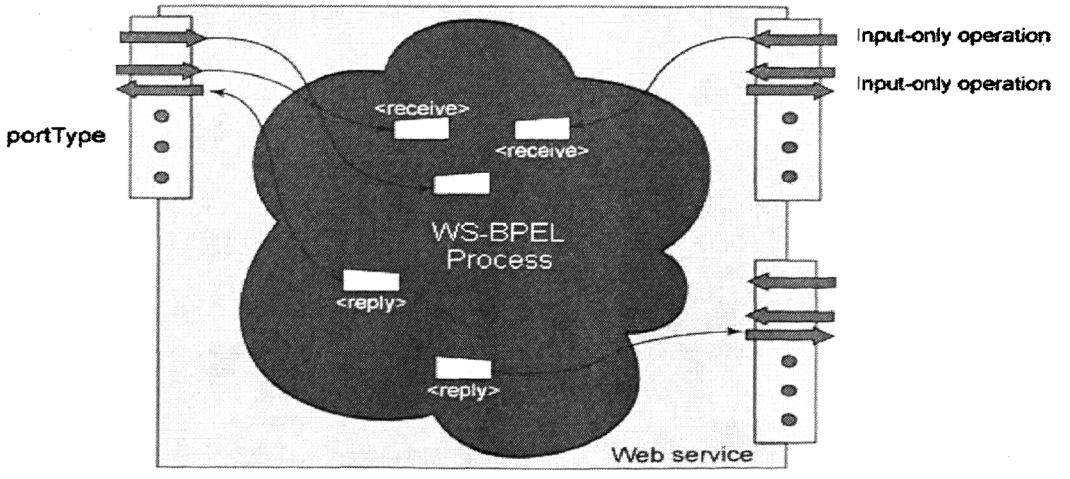

图 4- 8 WS-BPEL 流程组合的 Web 服务

下面介绍 WS-BPEL 主要的四个特点：

1)支持基本活动和结构化活动

基本活动是与外界进行交互最简单的形式，它们是无序的个别步骤，如接受外部调用请求(receive)、调用 Web 服务(in-voke)、回复请求(reply)等。 结构化活动将业务流程执行的基本活动整合到结构中，这些结构表达了业务协议中

所涉及的控制逻辑、数据流、故障和外部事件的处理等。如顺序(sequence)、并行(flow) 、循环(while)、分支（Switch)等。

2)对业务伙伴和角色的建模

BPEL 将与之交互的 Web 服务建模为伙伴，流程的每个伙伴既可以是流程要调用的服务，也可以是调用流程的服务。在流程中，每个伙伴都会被映射到它所扮演的角色上。同一伙伴可以在不同的流程中扮演不同的角色。

3)使用变量的数据持久性

变量是流程中所交换的数据，通常被映射成某个 WSDL 的消息类型。当 BPEL 流程接收到一个消息的时候，与此消息类型对应的变量便会被初始化，在流程中就可以使用此变量维持数据。从而可以实现了跨越 Web 服务的数据持久性。

4)事务和异常处理

WS-BPEL 提供一套事务处理和异常处理。在 BPEL 中一组活动可以通过嵌套在一个 scope 中形成一个事务。在这个 scope 中，可以指定故障处理器和补偿处理器，当出现故障时通过它们进行故障处理和错误恢复。

4.3.2 基于 BPEL 的业务流的构建

传统采用 BPEL 构建的基于 Web 服务的业务流大多属于静态的流程，不支持在运行阶段动态绑定和修改 Web 服务，存在灵活性差等缺点。因此，本文在服务集成层中探索性地提出了动态 Web 服务组合模型（D-WSCSM Dynamic Web Services Composition Model)，借助 UDDI 动态发现 Web 服务的能力，弥补了传统 BPEL 业务流程只能对 Web 服务进行静态绑定的缺陷，提供了动态绑定 Web 服务和异常情况下动态修改 Web 服务的功能，增强了 Web 服务的业务组合能力，提高了流程的灵活性，增强了流程的可用性。

4.3.2.1 D-WSCSM 架构

D-WSCSM 利用 UDDI 实现 Web 服务的动态发现，即利用 UDDI 统一发现服务的功能，为运行中的业务流程提供合乎需要的 Web 服务列表，供用户选择。在此基础上，可实现在业务流程运行阶段动态绑定 Web 服务和异常情况下动态修改 Web 服务的功能。

D-WSCSM 包括案例生成管理模块、BPEL 业务流程案例库、UDDI 注册中心和 BPEL 引擎四个模块，其结构如图 4-9 所示。

其中，案例生成管理模块包括了两个子模块：案例生成子模块和案例管理子模块。而 BPEL 业务流程案例库主要作用是存储 BPEL 业务流程模板，它已绑定默认的 Web 服务，在运行中可用适当的 Web 服务替换默认的 Web 服务。

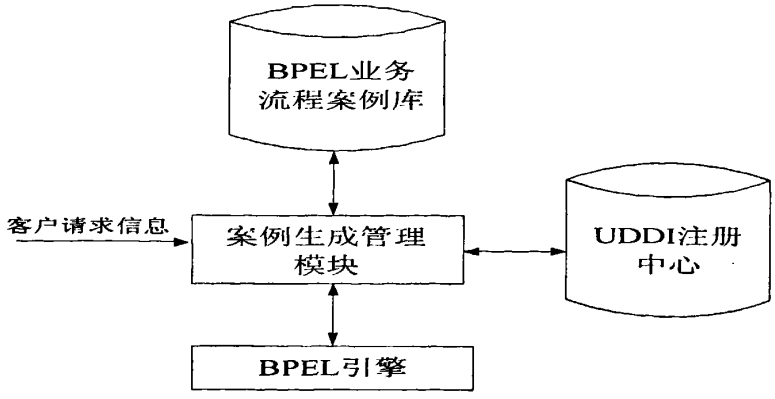

图 4-9 D-WSCSM 结构图

4.3.2.2 案例生成管理模块的设计与实现

1)案例生成子模块

(1) 处理流程

首先根据用户所输入的案例库名调用案例管理子模块，从 BPEL 业务流程案例库中进行搜索，如果库中不存在着该案例，将调用案例生成子模块。案例生成子模块中描述文档解析模块先将 BPEL 描述文档读入，在 D-WSCSM 模型中，解析器采用了 DOM 的解析方法，一次将 BPEL 描述文档读入，在读到每个节点属性时，解析模块调用解析支撑模块进行相关信息查询，获取该节点所代表的具体含义，之后将属性取值与的属性名相匹配，存入案例库中，并返回运行结果给案例管理子模块进行 BPEL 业务流程的运行。模块的处理流程如图 4-10 所示。

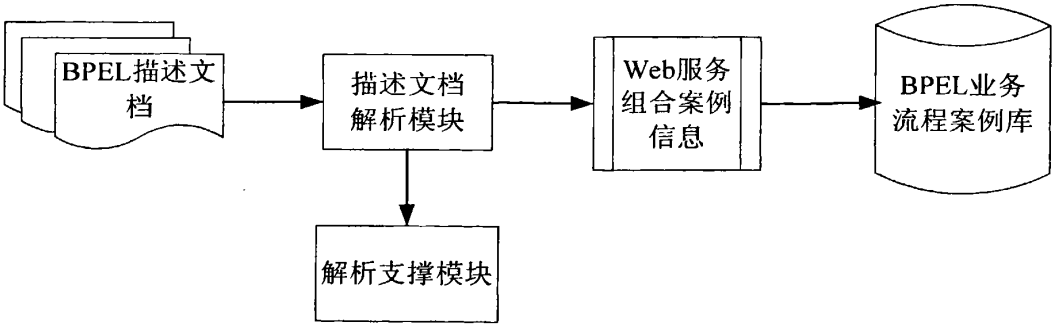

图 4-10 案例生成子模块处理流程

(2) 主要算法

案例生成的主要算法是，根据用户所输入的 Web 服务组合描述文档，依据 BPEL 标准解析并获取相关信息及其详细说明，将结果保存在案例库中。

系统首先解析用户所输入的 Web 服务组合描述文档中的流程定义文档，获取该 Web 服务组合中所调用到的所有子服务，之后递归获取每个子服

务的描述文档，验证子服务描述的完备性后进行解析获取子服务描述信息，最后如果流程定义文档和子服务的描述文档均满足完整性，则将该案例存储于案例库中，否则系统退出。如算法 4-1 所示。

算法 4-1：案例生成算法

```
输入：服务组合描述文档(BPELFiIeName )，案例库(CaseBase)
输出：结果集 result
初始化：result =Φ
实现：
  BPELdescriptionList=BPELDocumentBuilder.parse(BPELFiIeName);
                        //采用 DOM 模型解析服务组合描述 BPEL 文档；
  for (each element in BPELdescriptionList)
   {
     If(element is WSDLFiIeName)
      {//当前的 BPEL 节点为一个 WSDL 文件时，递归解析该 WSDL 节点；
            try {
             WSDLDescription=WSDLDocumentBuilder.parse(element);
                   //尝试解析 BPEL 文档中所包含的每 WSDL 文件；
             WSDLDescriptionList=Add( WSDLDescription);
                   //将解析所得的属性集存入 WSDLDescriptionList 中；
               }catch ex as Exception
             {
             result = error;
             return result;
             }
                  //对于任何一个 WSDL 的解析错误，表明该 Web 服务组
                    合流程均不可用，系统错误，终止循环。
            }
       }
    result= BPELDescriptionList∪WSDLDescriptionList;
    CaseBase.insert(result);     //案例库数据更新
    CaseBase.Update
  return result;
```

2)案例管理子模块

案例管理子模块处理流程：首先根据用户所输入的案例库名从 BPEL 业务流程案例库中进行搜索，如果库中存在着该案例，说明模板默认的 Web 服务符合请求，这时调用案例管理子模块中案例属性获取方法，读取案例库相关信息后，向 UDDI 注册中心查询符合该 BPEL 业务流程模板要求的 Web 服务，案例管理子模块将可用的 Web 服务以列表的形式发送给客户端，客户端选择相应的 Web 服务，并把结果发送给案例管理子模块；案例管理子模块根据客户端的选择，动态绑定业务流程中的 Web 服务，最后将 BPEL 业务流程送往 BPEL 引擎执行。在业务流程运行过程中，如果某个 Web 服务出现异常导致其不可用，为避免业务流程中断，D-WSCSM 中的 BPEL 引擎将启动案例管理子模块向 UDDI 注册中心查询可替代的 Web 服务对出现异常的 Web 服务进行动态替换，确保

业务流程能够继续正常执行。模块的处理流程如图 4-11 所示。

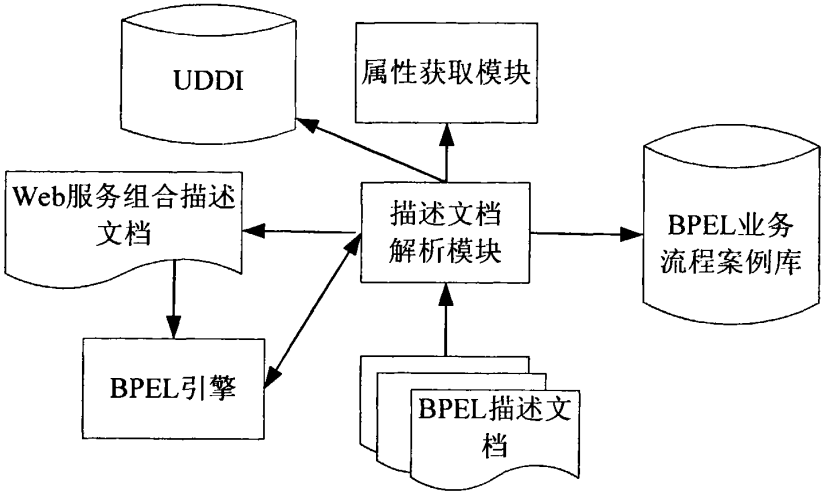

图 4-11 案例管理子模块处理流程

4.4 小结

本章主要对第三章提出的基于 Java EE 平台的应用集成模型中关键实现技术——基于 JCA 的 EAI 后端集成机制，基于 Java EE 的 Web 服务实现，基于 BPEL 业务流构建的设计技术实现进行详细的分析。

第五章 方案在实际项目中的应用

本章将通过对一个原型系统的开发，讨论如何在基于 Java EE 平台集成模型中集成企业遗留系统，并实现与其他 Web 服务系统的交互，从而验证方案的可行性。

5.1 系统分析

5.1.1 项目背景及描述

在淮北矿业集团公司信息化的进程中，各种业务系统(OA、质量、人事等)相继建立。随着业务和技术的发展，系统各自存在、各自为政的自治系统状况无法再维持下去，管理上需要整合各个系统的信息，实现跨越所有系统的整合管理，同时将各业务系统的管理信息流相结合以实现部门间高效率地协作。而且随着信息资源的增多，众多用户面对多个应用系统，每个系统占有自己的数据库，这样可能会导致信息和数据的更新不同步甚至不一致。

人事工资管理信息系统是淮北矿业集团公司多年前自主开发的一套企业内部人事管理系统，管理集团公司的部门和职工信息，是基于 Socket 协议进行数据传输，在 2008 端口监听。

淮北煤矿安全管理信息平台[39]是由合肥工业大学软件工程研究室为淮北矿业集团公司安全监察局研制开发的一套以安全信息管理和辅助安全决策为主要功能的应用软件。主要提供面向生产现场安全信息管理的集成化信息平台，以促进淮北矿业集团公司安全监察局和驻矿安监处日常安全管理工作的提高和改善。

本文结合第四章中提出的集成模型，设计并开发了一个资源适配器来实现对人事工资管理信息系统的连接与访问，供淮北煤矿安全管理信息平台重用，这样既保证业务模块的重用性，又保证了数据的一致性。并在此基础上，对人事工资管理信息系统中职工信息管理业务组件进行 Web 服务的封装，并将它做为公共服务组件，使能够适应在未来 SOA 的发展趋势中对面向服务的企业应用集成。

5.1.2 系统设计目标

整个系统设计要达到以下功能和目标：

1) 方便使用，能够尽可能地利用现有系统的职工信息管理模块，尽量保护现有的投资，减少重新的开发新模块的费用，同时避免对现有系统进行大规模的修改。
2) 具有良好的扩展性和集成性，不仅能支持现有的应用系统，当有新的应用被部署或开发的时候，这个统一职工信息管理可以作为它的职工管理模块的形式工作，也就是说，新的应用可以不自带职工信息管理，可以通过集成该业务模块形式来实现等价的功能。

3) 支持面向服务集成框架，使得在对各个应用系统实施基于 Web 服务的面向服务应用集成的时候能够使用这个职工信息管理服务进行对职工管理。其它应用系统可以根据需要使用人事系统封装的 Web 服务组件，这样可以考虑在原有的企业内部集成的基础上发展到 B2B 集成应用，以适应未来软件发展趋势。

5.2 系统设计与实现

系统的设计是以图 3-4 描述的基于 JCA 和 Web 服务相结合的应用集成模型为依据，通过 JCA 连接器架构实现对遗留的人事工资管理信息系统的业务功能封装成应用服务器中的 Java EE 组件，被淮北煤矿安全管理信息平台(采用 Java EE 平台开发)调用作为淮北煤矿安全管理信息平台的职工管理业务模块，并在此基础上进行 EJB 和 Web 服务的开发，然后在 Web 服务容器中实现人事工资业务系统与统一身份认证服务的集成。

系统开发的主要任务，一是通过对人事系统资源适配器的开发，实现人事工资管理信息系与淮北煤矿安全管理信息平台之间的集成，二是运用 4.2 节介绍的技术方法进行 Web 服务的开发，三是实现 Web 服务间的交互。

公司原来已经部署了基于 Java EE 的统一身份认证子系统，并能很好地满足业务要求，但为了适应未来的面向服务集成，并且保护企业现有的投资，本文以 Web 服务包装统一身份认证子系统，直接发布供人事工资管理信息系统或其他系统(如 OA、质量等)作为公共的统一身份认证管理组件进行调用。它属于服务层，可直接被集成到框架中来，因为它提供了可调用的 Web 服务，运行在 Web 服务容器中，直接通过服务客户端进行访问便可对其实现集成。人事工资管理信息系统是属于企业遗留系统资源层，可通过 API 接口访问。业务层的工作包括 JCA 连接器的开发和 EJB 组件的开发，以及对这些 EJB 接口进行 Web 服务的封装。服务集成层根据客户请求层的请求实现 Web 服务的交互。整个设计框图如图 5-1 所示。

5.2.1 开发工具

系统开发中涉及到的软件和工具如下：

- Java Standard Development Kit（JDK）版本 5.0 或版本 6.0。
- NetBeans IDE 6.0：NetBeans IDE 是由 Sun 公司推出一款优秀、免费、开源的集成开发工具，内置 Web 和应用服务器，如：Tomcat 6 和 GlassFish v2，并支持 JBoss, BEA WebLogic 10, IBM WebSphere 6 等，提供蓝图、模板和向导帮助我们创建 JAX-WS Web Services 应用。
- JBoss v5.0：开源的 Java EE 应用服务器，既可作为 EJB 容器使用，也可作为 Web 服务容器使用。
- ActiveBPEL[41]：开源的 BPEL 流程执行引擎。

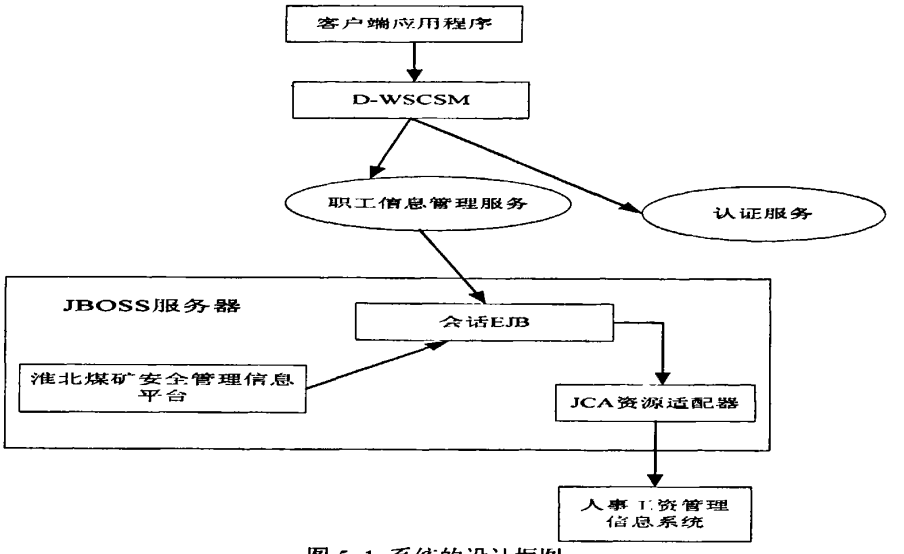

图 5-1 系统的设计框图

5.2.2 资源适配器的实现

资源适配器是一种系统级软件库，通常由 EIS 提供商根据自己的 EIS 资源进行开发。作为 EJB, JSP, Servlet 等 Java EE 平台组件访问底层资源的桥梁，资源适配器在 JCA 架构中的位置十分关键。本节针对人事系统的一个资源适配器实例的开发介绍了资源适配器构建、部署和应用。具体的内容包括：

- 资源适配器的开发；
- 资源适配器配置文件的编写；
- 资源适配器的打包及部署；
- 资源适配器的应用。

1) 资源适配器的开发

资源适配器的开发主要包括对前面所提到的系统级、应用级协议中各接口的实现。为了完全支持 Java EE 连接器体系的系统级协定，资源适配器需要实现几个 Java 接口，下表列出了这些系统级协定，并利用一个软件包分别组合在一起，并描述了每个接口的功能。如表 5-1 所示。

除了实现这些接口，资源适配器还需要提供一个连接站和一个连接的实现，在本实例开发的资源适配器支持 CCI，故它的连接站和连接类必须实现在 javax.resource.cci 软件包中定义的 ConnectionFactory 和 Connection 接口。下面以本次开发的资源适配器的连接管理为例，图 5-2 给出了在一个 Java EE 管理环境下一个 Java EE 组件请求的示意图。其操作情况如下：

① Java EE 组件使用 JNDI 查找一个连接站，连接站已经由资源适配器实现。

② Java EE 组件调用连接站上的 getConnection 方法。

③ 连接站向 ConnectionManager 授权请求。在管理环境下，ConnectionManager 由应用服务器实现。

④ ConnectionManager 从连接池取回一个现有的连接以满足请求，或者调用 ManagedConnectionFactory 来创建一个新的物理连接，它由 ManagedConnection 对象表示。

⑤ 应用服务器可以利用 ManagedConnection 注册一个或多个监听器，以便接收有关连接事件的通知。关闭连接就是连接事件的一个例子。

⑥ 在必要时，应用服务器完成事务划分，它从 ManagedConnection 获取 LocalTransaction 和 XAResource 对象。

⑦ 应用服务器通过调用 ManagerConnection 的 getConnection 方法请求连接句柄。

⑧ 应用服务器将连接句柄返回给连接站。

⑨ 连接站将连接句柄返回到 Java EE 组件。

表 5-1 由资源适配器实施的系统协定

接 口 名	说 明
Package javax.resource.spi	
ConnectionManager	由资源适配器实现，用以提供在非管理环境下的连接管理支持；
ConnectionRequestInfo	封装安全和客户专用信息，连接站创建实现这个接口的对象，资源适配器将其不变地传送到一个管理的连接站；
LocalTransaction	为本地事务划分提供方法，LocalTransaction 和 XATransaction 级别的资源适配器必须实现这个方法；
ManagedConnection	表示到一个 EIS 的物理连接；
ManagedConnectionFactory	表示一个管理连接的站；
ManagedConnectionMetaData	提供关于管理连接的信息；
Package javax.transaction.xa	
XAResource	为分布式 XA 事务划分提供方法，为两阶段提交提供支持；

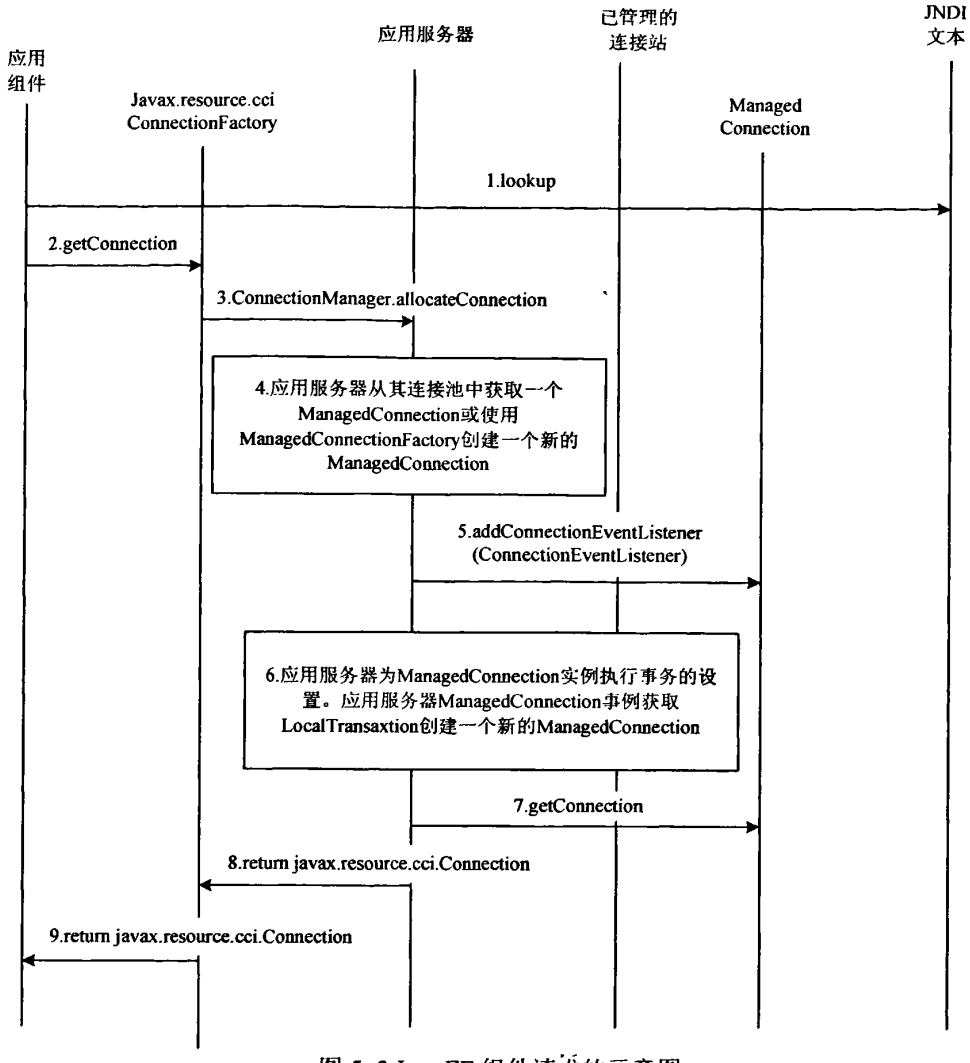

图 5- 2 Java EE 组件请求的示意图

根据列表 5-1 和图 5-2 的连接交互过程，本文设计的 JCA 资源适配器类图如下：

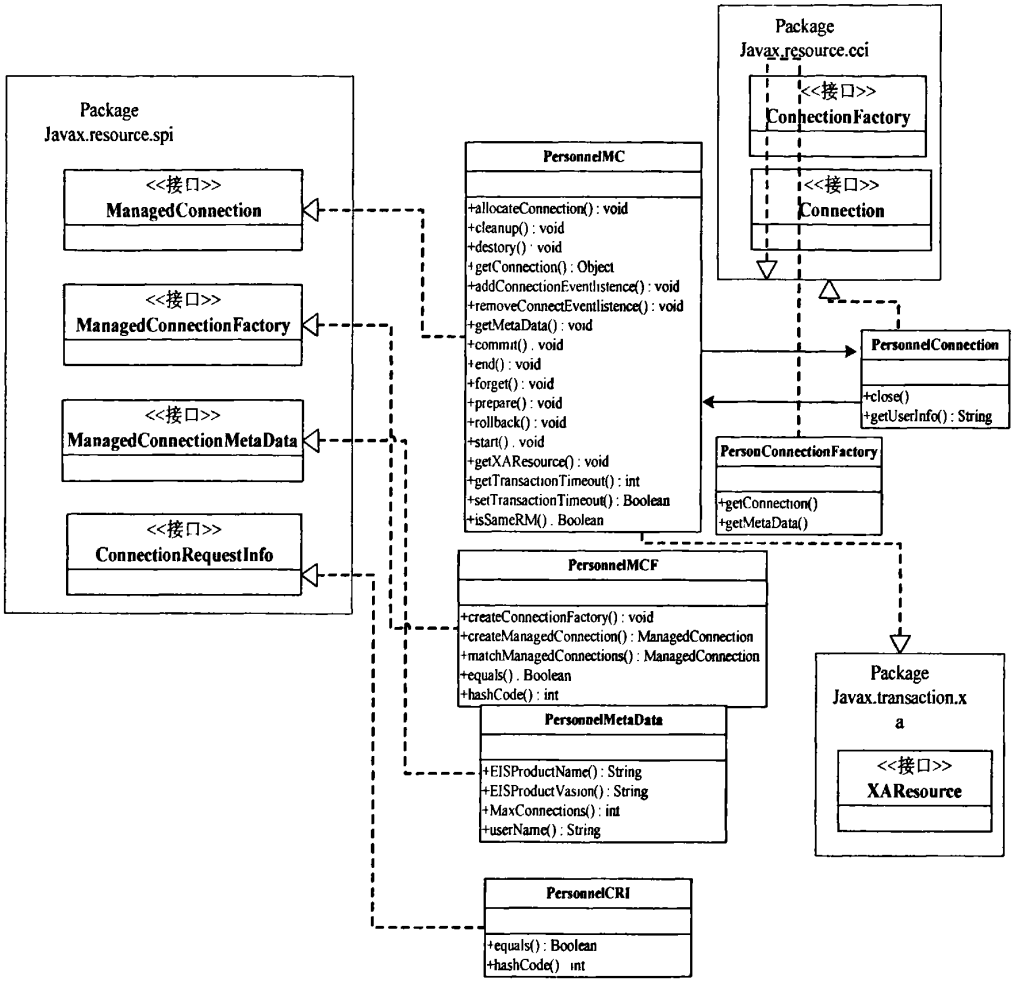

图 5-3 资源适配器类图

下面给出几个重要类的代码及其说明。

PersonnelConnection 扩展了 CCI 的 Connection 接口，它由客户端程序使用，代表了到 EIS 的“虚拟”连接，通过这个“虚拟”连接，客户端可以调用 EIS。需要注意的是，虚拟连接关闭时，物理连接不一定关闭。

PersonnelConnection 通过 PersonnelMC 来完成具体的业务方法。PersonnelMC 是代表到 EIS 的物理连接。

```
Package com.edu.sei.jca;
Import .....
Public class PersonnelConnection implements Connection
 {    //连接实现类，它通过 PersonnelMC 来完成具体的任务
       Protected PrintWriter out;
       Protected PersonnelMC   personnelMC;
          public void close  ( )
            { //关闭连接，释放资源
            if (personnelMC ==null)   return;
            personnelMC.removeConnection(this);
            personnelMC.connectionClosedEvent();
            personnelMC =null;
            }
          Public PersonnelMC   getManager ( )
           {      //返回和这个连接关联的被管理连接
           return personnelMC;
           }
          Public void setManager (PersonnelMC manager)
            {//设置和这个连接关联的被管理连接
            this. personnelMC =manager;
             }
          Public String getUserInfo (String ID) throws ResourceException
           {       //业务方法，它通过调用被管理的连接来实现。
             return personnelMC.getUserInfo (ID);
            }
         Public void invalidate()
           {  //使连接无效
             personnelMC = null;
            }
    ......
}
```

PersonnelConnectionFactory 也是和 CCI 相关的类，它实现了 ConnectionFactory 接口，它主要用于创建客户端要使用的虚拟连接(PersonnelConnection)。由于 PersonnelConnectionFactory 类需要在 JNDI 名字空间中注册，故它需实现 Serializable 和 Referenceable 接口。客户端查找 PersonelConnectionFactory 类，然后使用这个类来获得到 EIS 的连接。PersonnelConnectionFactory 代码如下。

```
Package com.edu.sei.jca;
Import ……;
Public class PersonnelConnectionFactory implements ConnectionFactory
  {
    Protected PersonnelMCF manager;
    Protected ConnectionManager connectionManager;
    Protected PrintWriter out;
  Public PersonnelConnectionFactory (PersonnelMCF manager,
          ConnectionManager connectionManager){   //构造方法
          this.manager=manager;
    If (connectionManager==null)
     { //如果连接管理器为空，那么创建一个新的连接管理器
       connectionManager=new ConnectionManager ( );
((PersonnelConnectionManager) connectionManager).setLogWriter (out);
 }
    else
     {
     this.connectionManager=connectionManager;
     }
  Public Connection getConnection ( ) throws ResourceException
     {  //获得一个连接。Java EE 应用使用 getConnection 方法的这种
        形式时，应用服务器负责将客户安全信息送到资源适配器。
   return (PersonnelConnection) connectionManager.allocateConnection
     (manager, null);
     }
Public ResourceAdapterMetaData getMetaData ()
  {     Return null;
  }
}
```

PersonnelConnectionManager 是连接的管理器，它为资源适配器把连接请求传递给应用服务器提供了一个切入点。应用服务器实现 ConnectionManager 接口，这个实现不针对具体的资源适配器和连接工厂接口。PersonnelConnectionManager 的任务就是分配连接。对于一些高级的应用，PersonnelConnectionManager 通过和连接池交互来分配连接，这个类主要实现 allocateConnection 创建或从连接池中取得一个空闲的连接。

/* 分配一个连接，并不一定每次都是创建一个新的PersonnelManagedConnection，却由应用服务器决定是创建还是匹配，可以认为此方法调用前应用服务器拦截了方法调用，先进行匹配，当没有匹配的PersonnelManagedConnection 时才创建 */

```
Package com.edu.sei.jca;
Import ……
Public class PersonnelConnectionManager implements
    ConnectionManager, Serializable
{
    Protected PrintWriter out;
    Public Object allocateConnection (ManagedConnectionFactory
        managedconnectionfactory, ConnectionRequestInfo
        connectionrequestInfo) throws ResourceException
    {                                          //分配一个连接
      PersonnelMC managedConnection=
        managedconnectionfactory.createManagedConnection(null,
      connectionRequestInfo);
return managedConnection.getConnection(null, connectionRequestInfo);
    }
     Public void setLogWriter (java.io.PrintWriter out)
      {
            this. out =out;
       }
}
```

PersonnelMC 是资源适配器的关键所在，它代表了和 EIS 的物理连接，前面介绍的 PersonnelConnection 是由客户端使用的虚拟连接，虚拟连接要通过物理连接才能使用 EIS。一个物理连接可以被多个虚拟连接使用，可以通过associateConnection 方法来把虚拟连接和物理连接进行关联。PersonnelMC 也提供了产生虚拟连接实例的方法。PersonnelMC 的代码如下：

```
Package com.edu.sei.jca;
Import ……
Public class PersonnelMC implements ManagedConnection
{           //PersonnelMC 代表了到 EIS 的物理连接
    Protected Socket socket;    //和 server 连接的 Socket
    Protected PrintStream serverStream;
    Protected BufferedReader serverBufferedReader;
    Protected Setconnections=new HashSet ();   //被管理的连接
    Protected SetconnectionListeners=new HashSet (); //连接监听器
    PersonnelMCF   factory;//连接工厂
```

```
    Public PersonnelMC (PersonnelMCF factory)
      {
       this.factory=factory;
      }
    //为连接增加事件监听器
     Public void addConnectionEventListener(ConnectionEventListener   l)
       {
          connectionListeners.add (l);
        }
     //清除连接监听器
   Public void removeConnectionEventListener (ConnectionEventListener l)
      {
       connectionListeners.remove (l);
      }
     //返回连接工厂
     Public PersonnelMCF getFactory ()
      {
        Return factory;
      }
      //增加一个连接，并且返回它
  Public Object getConnection (Subject subject, ConnectionRequestInfo
cxRequestInfo)
      {
            PersonnelConnection connection=new PersonnelConnection ();
             Connection. setManager(this);
             connection.setLogWriter (out);
             addConnection(connection);
             return connection;
       }
   public void cleanup ( )   throws ResourceException
   {   //清除占用的资源
        destroyedError ( );
        Iterator it=connections.iterator ();
        While (it.hasNext ())
         {
       PersonnelConnection PersonnelConnection= (PersonnelConnection)
                          it.next ();
        PersonnelConnection.invalidate ( );
         }
        Connections. clear ( );
    }
```

```
public void destroy ( )
{    //销毁所有的物理连接
      Iterator it=connections.iterator ( );
       While (it.hasNext ())
        {
      PersonnelConnection PersonnelConnection= (PersonnelConnection)
            it.next ( );
          PersonnelConnection.invalidate ( );
        }
       connections.clear ( );
       If (socket! = null)
         try {socket.close ( ); }
         Catch (Exception e) {}
  }
 void connectionClosedEvent ( )
    {//关闭连接的事件，释放资源，由连接监听器来处理
       Iterator it=connectionListeners.iterator ( );
        While (it.hasNext ())
{ConnectionEventListener listener=(ConnectionEventListener) it.next ( );
            listener.connectionClosed ( );
(new ConnectionEvent(this, ConnectionEvent.CONNECTION_CLOSED));
}
    }
   //打开物理连接，物理连接是和 EIS 的当前连接。通过 Socket 来进
            行通信
Public void openPhysicalConnection (String serverName, int portNumber)
    Throws UnknownHostException, IOException {
      socket=new Socket (serverName, portNumber);
      serverStream=new PrintStream (socket.getOutputStream ( ) );
        ServerBufferedReader=new BufferedReader
        (new InputStreamReader (socket.getInputStream ( ) );
  }
//业务方法，它是同步的，同时只能一个和 Server 进行通信
  Public synchronized String getUserInfo (String ID)
       throws ResourceException
      {
       serverStream.println (ID);
       try
         {
          String in = serverBufferedReader.readLine ();
```

```
            return in;
              }
            Catch (Exception e)
              {
                Throw new ResourceException
                ("调用 getUserInfo()发生错误: " + e.toString());
              }
        }
```

PersonnelMCF 是 PersonnelMC 的工厂，它的主要任务是创建和匹配 PersonnelMC 实例。在创建 PersonnelMC 实例时，PersonnelMC 实例同时打开到 EIS 的物理连接。PersonnelMCF 的代码如下所示。

```
Public class PersonnelMCF implements ManagedConnectionFactory,
Serializable
  {
   protected PrintWriter out=new PrintWriter(System.out);
   private int port;          //连接 EIS 的端口
   private String server;   //数据库服务器的 url
  //创建连接工厂，指定连接工厂的连接管理者为 connectionManager。
Public Object createConnectionFactory (ConnectionManager
connectionManager)
{
  PersonnelConnectionFactory personnelConnectionFactory= New
   PersonnelConnectionFactory (this, connectionManager);
          personnelConnectionFactory. SetLogWriter (out);
            Return personnelConnectionFactory;
 }
 //创建被管理的连接，被管理的连接是和 EIS 的真实连接，它是物理连接。
      Public ManagedConnection createManagedConnection (Subject subject,
        ConnectionRequestInfo cri) throws ResourceException {
          PersonnelMC personnelMC =new ManagedConnection (this);
            personnelMC. SetLogWriter (out);
            try
              {
                System.out.println(“打开物理连接.....”),
                personnelMC.openPhysicalConnection (server, port);
                 return personnelMC;
```

```
        } Catch (IOException e)
         {throw new ResourceException (e.toString ( ); }
      }

  //匹配被管理的连接，如果匹配到连接，则返回，否则返回 null
    Public ManagedConnection matchManagedConnections
        (Set connections, Subject subject, ConnectionRequestInfo cri)
      { Iterator it=connections.iterator ( );
        While (it.hasNext ()) {
              Object obj=it.next ( );
            If (obj instanceof PersonnelMC)
              {
   PersonnelMC personnelMC=(PersonnelMC) obj;
   PersonnelMCF personnelMCF =personnelMC.getFactory ( );
              If (personnelMCF.equals (this))
                  return personnelMC;
              }
               }
          return null;
      }
 Public int hashCode ()
  {
      If (server == null) return 0;

return server.hashCode ();
  }
 //判断两个被管理的连接工厂是否相等
 Public boolean equals (Object O)
   {
      If (O == null) return false;
      If (! (O instanceof   PersonnelMCF))
        Return false;
      PersonnelMCF other = (PersonnelMCF) o;
      If (server.equalsIgnoreCase (other.server) &&
        Port == other.port) return true;
      Return false;
   }
  ......
```

在 PersonnelMCF 类中，设置了两个属性 Port 和 Server，Port 表示连接 EIS

使用的端口，Server 表示连接 EIS 时使用的 URL。这两个属性需要在资源适配器的部署描述符里指定具体的值。

2) 编写配置文件

每个资源适配器都包括一个 XML 部署描述符文件。部署描述符文件描述了资源适配器的功能，并为配置者提供足够的信息，以便在基于应用服务器的环境中正确地设置资源管理器。应用服务器还依赖部署描述符中的信息，以了解如何与资源适配器较好地进行互操作。在本实例中，部署描述符里指定资源适配器的相关接口和实现类具体如下：

ManagedConnectionFactory，这里是 PersonnelMCF；

Connectionfactory 的接口，这里是 javax.resource.cci.ConnectionFactory；

Connectionfactory 的实现类，这里是 PersonnelConnectionFactory；

连接的接口，这里是 javax.resource.cci.Connection；

连接的实现类，这里是 PersonnelConnection。

另外，还需要指定以下的属性：

是否支持事务，这里是 NoTransaction；

安全认证的支持，这里是 false；

PersonnelMCF 中使用的属性，其中 Server 的值为 localhost，Port 的值为 2008。

资源适配器的具体内容要保存在一个名为 ra.xml 文件中，并把该文件放在资源适配器打包文件 RAR 的 META-INF 目录下。资源适配器的具体内容如下：

```
<? Xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE connector PUBLIC '-//Sun Microsystems, Inc.//DTD
  Connector 1.0//EN'  'http: //java.sun.com/dtd/connector_1_0.dtd'>
<connector>
   <display-name>PersonnelRA</display-name>
   <vendor-name>HBCOAL</vendor-name>
   <spec-version>1.0</spec-version>
   <eis-type>NO TRANS</eis-type>
   <version>1.0</version>
<resourceadapter>
 <managedconnectionfactory-class>
  com.edu.sei.jca. PersonnelMCF
</managedconnectionfactory-class>
 <connectionfactory-interface>javax.resource.cci.ConnectionFactory</co
nnectionfactory-interface>
<connectionfactory-impl-class>com.edu.sei.jca.PersonnelConnectionFactory
</connectionfactory-impl-class>
```

```
<connection-interface>
javax.resource.cci.Connection
</connection-interface>
 <connection-impl-class>
  com.edu.sei.jca.PersonnelConnection
  </connection-impl-class>
      <transaction-support>NoTransaction</transaction-support>
      <Config-property>
        <config-property-name>Server</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>localhost</config-property-value>
      </config-property>
      <Config-property>
        <config-property-name>Port</config-property-name>
        <config-property-type>java.lang.Integer</config-property-type>
        <config-property-value>2008</config-property-value>
      </config-property>
      <reauthentication-support>false</reauthentication-support>
   </resourceadapter>
 </connector>
```

3) 资源适配器的打包及部署

资源适配器的打包文件由实现系统级/应用级协议和资源适配器功能的类/接口、资源适配器的工具类、平台相关的本地类库和描述文件等几个部分组成。资源适配器必须由 Java Archive (JAR)打包为以 RAR 为扩展名的压缩包。

资源适配器的描述文件必须以 xml 格式存放在 META-INF 目录下，并命名为 ra.xml。所有的类/接口应该被打包为一个或多个以 jar 为扩展名的压缩包。打包后的资源适配器名为 PersonnelRAR.rar。其中 ra.xml 为资源适配器部署文件，PersonnelRAR.rar 是实现了系统级协议和资源适配器功能的类包。

资源适配器模块类似于其他 Java EE 模块，必须先进行部署和配置以便为其他的 Java EE 应用组件使用。部署资源适配器有两种方法可供选择：

- 独立部署—将资源适配器部署为一个独立模块。
- 捆绑部署—首先将资源适配器装配到一个 Java EE 应用中，然后部署为这个 Java EE 应用的一部分。

本文中是将资源适配器部署为 JBoss 应用服务器的一个独立单元。按照这种部署，Java EE 平台能保证资源适配器适用于在 JBoss 应用服务器上运行的所

有 Java EE 应用，好处是允许多个 Java EE 应用共享惟一的资源适配器。

4) 资源适配器的应用

一旦资源适配器被打包部署到应用服务器中，其他业务组件就可以通过这个适配器来访问人事工资管理信息系统提供的服务。业务组件当然可以直接访问资源适配器，但这并不是最好的办法。资源适配器是后端 EIS 的代表，按照多层思想及 Java EE 设计模式之 Facade 模式的要求，最好的办法是用会话 Bean 将适配器封装，通过 Bean 业务方法来调用这些资源层的逻辑。下面将通过一个名为 PersonnelEJB 的 stateless Session Bean 的例子说明如何在 EJB 应用组件中通过连接器的方式对资源适配器(PersonnelConnectionFactory)进行访问。

（1）EJB 组件的开发

EJB 组件通过资源适配器来访问 EIS，在 EJB 组件中，定义了最终客户端要使用的业务方法，一般实现为会话 Bean，它一般通过和多个 Java EE 组件或者资源适配器交互来完成具体的业务逻辑。本次开发的 EJB 组件实现的功能非常简单，就是接收最终客户端的请求，然后通过资源适配器调用 EIS 并获得结果，最终把结果返回给客户端。EJB 组件定义了一个业务方法，它的远程接口的代码如下：

EJB 组件的远程接口：

```
Package com.edu.jca.ejb;
Import ..............;
Public interface Personnel extends EJBObject
{
    Public String getUserInfo (String id) throws RemoteException;
}
```

在这个接口里，定义了一个业务方法 getUserInfo(String id)。下面我们看 EJB 组件的实现类部分代码。

EJB 实现类的部分代码:

```
Package com.edu.jca.ejb;
Import ....
Public class PersonnelEJB implements SessionBean
  {    Private SessionContext sessionCtx;
  public String getUserInfo(String id)throws Exception //业务方法
  String result="";
   try {
    InitialContext iniCtx = new InitialContext ();
    Context enc = (Context) iniCtx.lookup ("java: comp/env");
```

```
Object ref = enc.lookup("eis/PersonnelEISFactory");//获得连接工厂
ConnectionFactory dcf = (ConnectionFactory) ref;
PersonnelConnection dc =(PersonnelConnection) dcf.getConnection();//从连接
                                                        工厂创建连接
    result=dc.getUserInfo(id);//调用虚拟连接的业务方法
    dc.close();    //使用完，关闭虚拟的连接。
        }
        Catch (NamingException e)
        {
            System.err.println("在查找 JNDI 名字时遇到错误:"+ e);
        }
        return result;//返回调用的结果
    }
...
}
```

从上面代码可知，首先通过 JNDI 来查找连接工厂实例（这个实例由应用服务器自动绑定），然后从这个连接工厂获得连接实例，接下来从连接实例调用业务方法，最后关闭连接。

（2）EJB 组件交互

淮北煤矿安全信息平台的业务模块通过这个 EJB 组件可以实现对人事系统的访问。具体访问代码如下：

```
InitialContext context=new InitialContext ();
Object ref = context.lookup ("PersonnelEJB");
PersonnelHome home=
(PersonnelHome) javax.rmi.PortableRemoteObject.narrow (context.lookup
    ("ejb/personnel"), PersonnelHome.class);
Personnel personnel=home.create ();
personnel.getUserInfo (id);
```

最后根据应用服务器的要求，EJB 的部署需要 ejb-jar.xml 以及 Jboss.xml 文件的共同描述。这里不再给出文件代码。

至此，通过 JCA 连接器架构就可以实现了对遗留的人事工资管理系统与淮北煤矿安全管理信息平台之间的集成，完成了业务流程的整合，实现了业务集成的基础上的数据集成，最后把资源适配器和淮北煤矿安全管理信息平台部署到 JBoss 应用服务器中，启动服务器，进入到安全管理信息平台的职工信息管理模块，就看到人事系统中职工信息管理模块被安全管理信息平台调用，如图 5-4 所示。

图 5-4 安全信息平台调用人事工资管理信息系统的职工信息管理模块

图 5-5 显示安全信息平台调用人事工资管理信息系统的职工和部门信息，完成奖惩管理模块奖励人员信息添加，实现数据资源的共享。

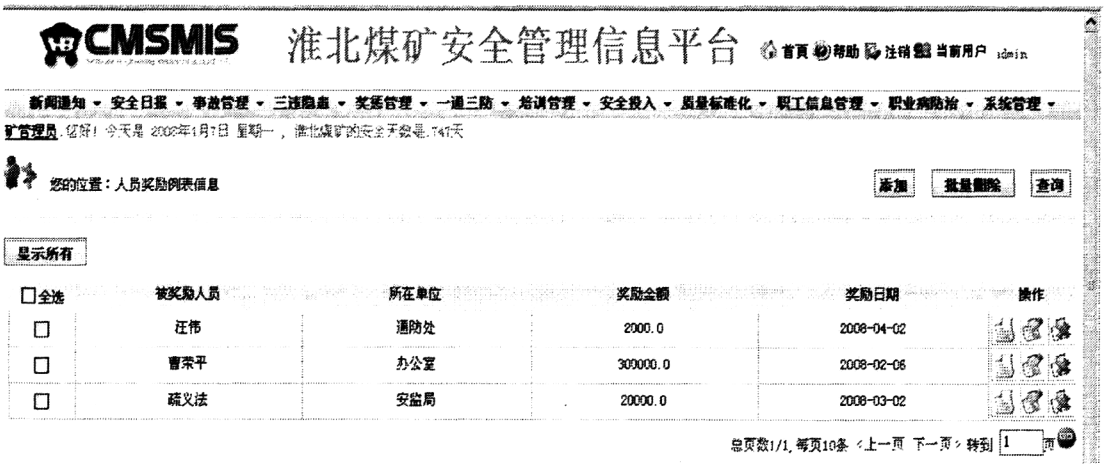

图 5-5 人员奖励管理页面

5.2.3 Web 服务的开发

为了将人事系统中职工信息管理业务模块做为一个公共服务组件，并适应未来面向业务流程集成，将对其进行 Web 服务的封装。而在 Java EE 平台上可以发布为 Web 服务的模块有三种：Servlet、EJB、POJO（Plain Old Java Object），本文就是采用基于 JAX-WS 的“自底向上”开发模式对上一节中开发访问资源适配器的 PersonnelEJB 组件暴露成 Web 服务。

将 EJB 组件声明为 Web 服务很简单，不用开发人员编写任何代码就能完成这一工作，只需使用标注—Annotation，就可以将其暴露成为 Web 服务。如下：

```
Package com.edu.sei.services;
Import .....
@WebService (serviceName = "Employee",
        PortName = "EmployeePort",
        targetNamespace="http://com.edu.sei/jaxws/Employee")
Public class PersonnelEJB implements SessionBean
{
Private SessionContext sessionCtx;
  @WebMethod
   public String getUserInfo(String id)throws Exception   //业务方法
      {
    String result="";
    try {
       InitialContext iniCtx = new InitialContext ();
       Context    enc = (Context) iniCtx.lookup ("java: comp/env");
       Object     ref = enc.lookup("eis/PersonnelEISFactory");//获得连接工厂
       ConnectionFactory dcf = (ConnectionFactory) ref;
       PersonnelConnection dc =(PersonnelConnection) dcf.getConnection();
                                                    //从连接工厂创建连接
       result=dc.getUserInfo(id);//调用虚拟连接的业务方法
       dc.close();     //使用完，关闭虚拟的连接。
            }
            Catch (NamingException e)
            {
                System.err.println("在查找 JNDI 名字时遇到错误:"+ e);
            }
            return result;//返回调用的结果
        }
     ...
     }
```

实现 Web 服务后，需要生成部署服务所需的所有构件，然后将 Web 服务打包为部署构件（通常为 WAR 文件），并将 WAR 文件部署到任何支持 JAX-WS 2.0 规范的兼容服务器上（本文部署到 JBoss）。运行 wsgen 工具，以职工信息管理 Web 服务的 JAX-WS 可移植构件，如下：wsgen -cp com.edu.sei.services.PersonnelEJB -wsdl，这样就生成了服务的 WSDL 文件，具

体代码在这里就不叙述了。

至此，职工信息管理 Web 服务业务组件开发工作就完成了，下一步工作就是实现 Web 服务间的协同和交互。

5.2.4 Web 服务的交互

为了保证集成系统的动态性，采用 4.3 小节的提出的动态 Web 服务组合系统模型（D-WSCSM）实现 Web 服务的动态调用，其调用模型如图 5-6 所示。

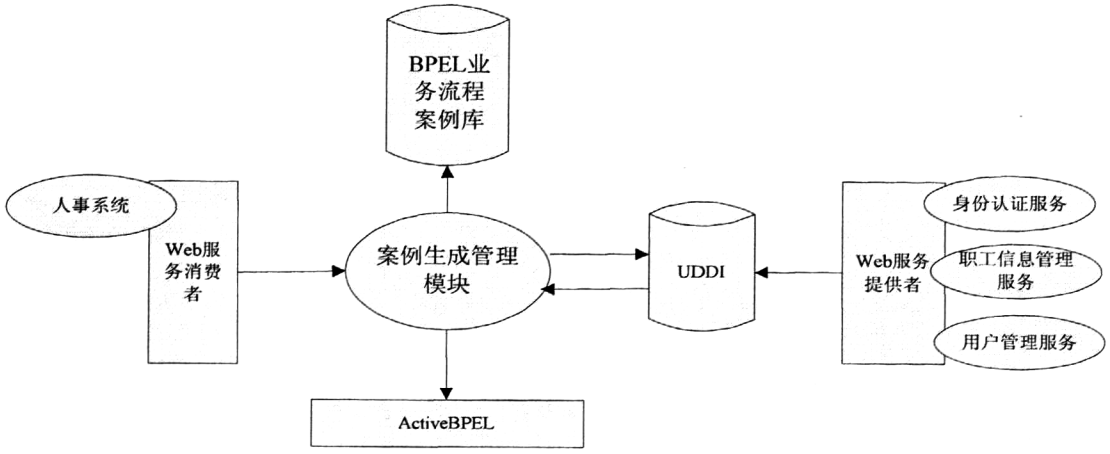

图 5-6　Web 服务交互的调用模型

人事业务系统作为服务的消费者，首先向案例生成管理模块发送描述身份认证服务与用户管理服务组合的 BPEL 文件，然后案例生成管理模块解析 BPEL 文件，从 UDDI 注册中心查询获取统一身份认证服务的 WSDL 文件，通过对统一身份认证服务的 WSDL 文档进行解析得到的接口参数，然后利用接口参数对身份认证服务进行动态调用。

用户输入用户名和密码，统一身份认证服务进行身份认证，如图 5-7 所示。

图 5-7　身份认证界面

认证成功后，自动进入人事工资管理信息系统主界面，用户可以根据自己权限对人事工资系统操作。如图 5-8 所示。

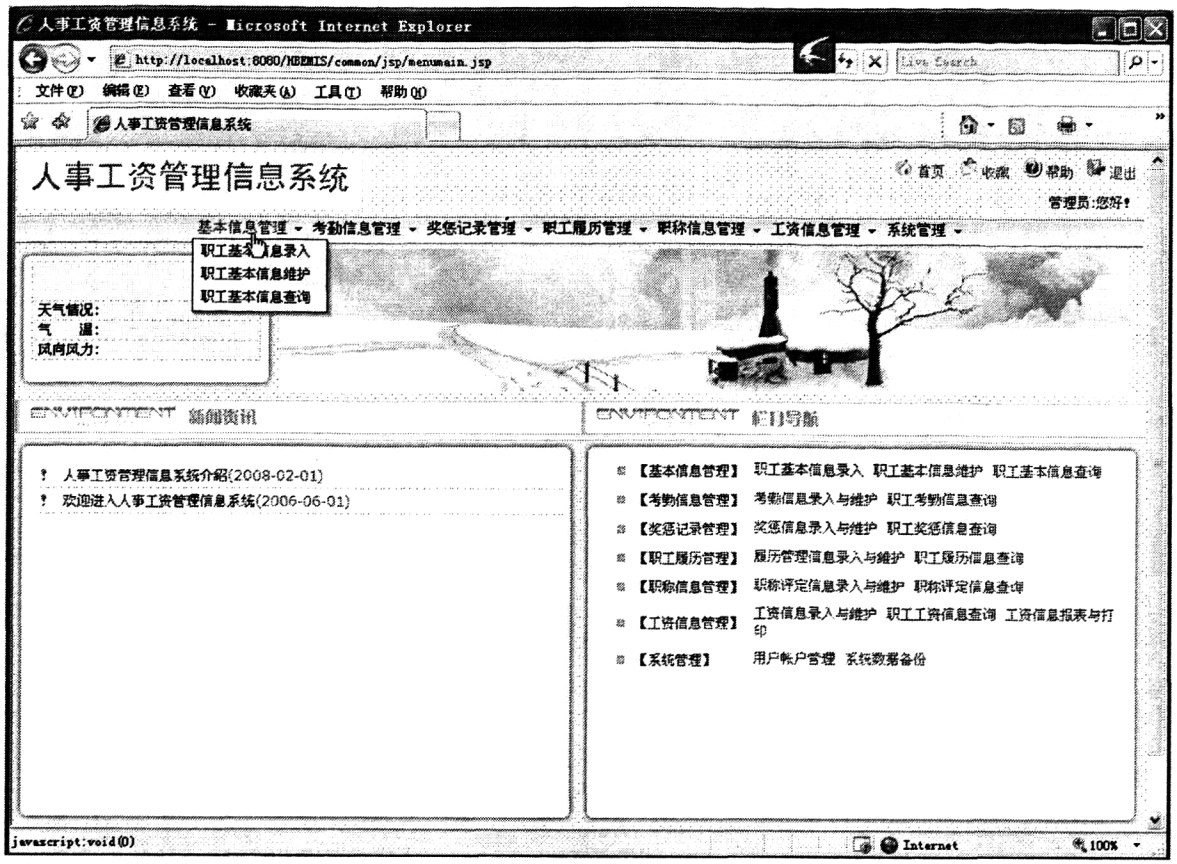

图 5-8 人事工资系统主界面

5.3 小结

本章介绍了项目背景，以本文提出的集成模型为依据，完成了人事系统的资源适配器的设计、开发和部署，实现了人事系统与淮北煤矿安全管理信息平台之间的整合，并此基础上进行 Web 服务的封装，实现了 Web 服务之间的合成，以适应企业 B2B 集成。

第六章 总结与展望

6.1 论文工作总结

针对目前企业应用集成的问题主要在于对遗留系统的处理上，本文通过对基于 JCA 技术和基于 Web 服务技术的企业应用集成方案研究分析，得出了它们在技术优势上具有很好的互补性，于是将两种技术相结合，提出了一种基于 Java EE 平台的企业遗留系统应用集成方案，并进行了具体的应用研究，这样不仅可以通过较低的成本解决企业中的信息孤岛问题，以及企业中系统更新方面的异构性问题，而且保护了早期企业在信息化建设中的投资，并能够适应在未来 SOA 的发展趋势中对面向服务的企业应用的集成，因此这种新的应用集成方案对企业的发展具有很好的承上启下的作用。

本文所做的工作主要包括以下几点：

1) 分析了 EAI 相关概念、层次构成、一个完整 EAI 解决方案各层技术实现和 EAI 实施的特点以及实现平台的选择。
2) 对 JCA 和 Web 服务两种技术在 EAI 方案中应用进行了分析，并比较两种技术在集成企业遗留系统中应用，得出两种技术相结合的优势，提出一种基于 Java EE 平台的企业遗留系统集成模型，并给出了设计思路以及分层技术实现，其研究成果主要表现在以下几个方面：

①运用了多种设计模式的思想，使得开发出的资源适配器只要修改简单的配置同样也可以插入到遵循了 JCA 规范的其他 Java EE 应用服务器当中，实现了软件的可配置性及可重用性。

②研究了利用现有 Java EE 开发平台的特性进行 Web 服务的开发过程，并使得 Web 服务封装、发布以及部署过程得以简便。

③针对集成模型服务集成层，提出了一种动态 Web 服务组合模型（D-WSCSM Dynamic Web Services Composition Model），弥补了传统 BPEL 业务流程只能对 Web 服务进行静态绑定的缺陷，并提供了动态绑定 Web 服务和异常情况下动态修改 Web 服务的功能，增强了 Web 服务的业务组合能力。

3) 开发了一个实验系统，描述了基于该模型的应用集成系统的开发实现过程，并展示了系统运行效果。

6.2 工作展望

企业应用集成是一项庞大而复杂的系统工程，在本文的范围内不可能涉及到集成的方方面面。要把新的集成模型应用在实际工程当中，还需要进一步完成以下几个方面的工作。

1) 完善 JCA 资源适配器。本文完成了资源适配器的总体设计，实现了资源的连接和管理，进一步工作中还需要实现其他功能模块，如事务管理、安全性管理等等。

2) 虽然 JCA 资源适配器可以实现对现有业务逻辑的封装，但这是同步方式的封装，如何实现异步的 EIS 访问，在本文中并没进行相关的研究。将 JMS 加入到集成模型中，利用 JMS 消息传递机制实现 EIS 之间的解耦，也是下一步要进行的工作。

3) 本文提出的动态 Web 服务组合模型仍然是一种“静态”的服务合成方法，是通过服务注册中心浏览器进行服务选择，但这种服务选择还比较初步，只是人工进行选择，并没有进行相关服务发现和选择的具体设计，要想达到完全动态的 WEB 服务合成(即能够根据需求，自动在网上搜索合适的 Web 服务，并进行合成)，如何做到这一点，还需要大量的研究工作。

参考文献

[1] The Legacy System Dilemma: Making the Right Choices, H.Alex Aminian, Insurity Inc. 2003

[2] Legacy Wrapping In a Component Architecture, MTW Corporation, http:// www. edgeusergroup.org/cbd/docs/legacy%20wrapper.pdf

[3] JOHANNESSON P. Design principles for process modeling in enterprise application integration, InformationSystems.2003.3

[4] KOBAYASHIT. Business process integration as a solution to the implementation of supply chain management systems.2003.

[5] HALEVY A Y. Theory of answering queries using views [J].SIGMOD Record, 2000, 29(4):40-47

[6] HULL R.Managing semantic heterogeneity in databases: a theoretical Perspective[C].Proc of the ACM SIGACT-SIGMOD-SIGART Sym-posium on Principles of Database Systems. Arizona: 1997:51-61.

[7] BPMI.org. Business Process Modeling Language [EB/OL] http://bpmi.org/hpml-spec.esp.

[8] LINTHICUM D S. Enterprise application integration [M].Massechusetts: Addison-Wesley, 1999.

[9] PUSCHMANN T, ALT R. Enterprise applicationintegration-the case of the mbert Bosch group[C].Pros of the 34th Hawaii International Conference on System Sciences.Hawaii.2001.9047-9056.

[10] LEE J, SIAU K, HONG S. Enterprise integration with ERP and EAI [J].Communications of the ACM, 2003, 46(2):54-60.

[11] Sup J G, MEWES M.An architecture proposal for enterprise message brokers [M].Heideberg: Springer Berlin, 2001.

[12] CALI A, CALVANESE D, GIACOMO De D, et al. Accessing data integration systems through conceptual schemas[C].Proc of the 20th In Conf on Conceptual Modeling. 2001:270-284.

[13] LENZERINI M. Data integration needs reasoning[C].Proc of the 6th International Conference on Logic Programming and Nonmonolonic Reasoning. Vienna. 2001: 54-61.

[14] ULLMAN J D.Information integration using logical views [J].Theoretical Computer Science. 2000, 239(2):18-21.

[15] Thomas Schael.Workflow Management Systems for Process Organisations [M].New

York: Springer-Verlag, 1996.

[16] GEORGAKOPOULOS D, HORNICK M. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure [J]. Distributed and Parallel Databases, 1995, 3 (2):119-153.

[17] 娄丽军.EAI 概念与技术分析 [EB/OL]http://www.ibm.com/developerworks/cn/websphere/library/techarticles/loulijun/0412/eai.html.

[18] Web Services Description Language (WSDL) Version1.2 [EB/OL]http://www3.org/TR/2003/WD-wsdl12-20030611, June, 2003.

[19] Simon Johnston.Modeling service-oriented solutions [EB/OL], http://www.ibm.com/developerworks/rational/library/jul05/johnston/index.html?S_TACT=105AGX52&S_CMP=cn-a-r.

[20] EAI 专题.http://www.ibm.com/developerworks/cn/ondemand/eai/level.html,Apirl, 2007.

[21] Web Services and Service-Oriented Architectures [EB/OL], http:// www.service-architecture.com/.

[22] UDDI Version 3.0.2 Specification, http://uddi.org/pubs/uddi_v3.htm.October, 2004.

[23] W3C Simple Object Acess Protocol Specification Version 1.2, http://www.w3.org/TR/soap/, June 2003.

[24] Rahul Sharma, Beth Steams, Tony Ng 著.杨晓红，杨莉萍，李健译.J2EE 连接器体系与企业应用集成[M].北京：电子工业出版社，2003.

[25] Java EE Connector Architecture [EB/OL]. http://java.sun.com/Java EE/connector/ , May, 2005.

[26] Alan Shslloway,James R.Trott 著.熊节译.设计模式精解[M].北京.清华大学出版社.2004

[27] W.Pree. Design patterns for Object-Oriented Software Development [M]. Ninth Edition.Addison Wesly/ACM Press. 2002.

[28] R.Wirfs-Brock, R.Johnson.Surveying current research in object-oriented design [J] Communications of the ACM. 2000.33(9):55-57.

[29] Jay Foster,Mick Porter 等著.李华鹰，黎晓冬等译.应用 Java API 开发 Web 服务 -Developing Web 服务 With Java AIPs for XML Using WSDP[M]. 第一版.北京.中国水利水电出版社.2003.7.

[30] Joshy Joseph. 开发者关于 JAX-RPC 的介绍[EB/OL]. http://www-900.ibm.com/developerWorks/cn/webservices/ws-jaxrpc/parll/index.shtml

[31] Vlada Matena,Sanjeev Krishnan 等著.施平安，施惠琼，罗德良译.EJB 应用指南—基于组件的 J2EE 平台开发[M].第 2 版.北京.清华大学出版社.2004.

[32] Richard Monson-Haefel 著.崔洪斌，王爱民译.JAVA EE Web 服务高级编程[M].北京.清华大学出版社.2004.

[33] Kevin Mukhar, James Weaver, Jim Crume, Chris Zelenak 著.窦巍，顾玲译.Java EE 5 开发指南[M].北京.机械工业出版社.2006.

[34] John Stearns, Roberto Chinnici, and Sahoo.An Introduction to the Java EE 5 Platform. http://java.sun.com/developer/technicalArticles/JAVA EE/intro_ee5/ .May, 2006.

[35] Java API for XML-Based Web 服务 (JAX-WS) Documentation http://java.sun.com/webservices/jaxws/docs.html. Jan, 2006.

[36] Naveen Balani,Rajeev Hathi.设计与开发 JAX-WS2.0 Web 服务，http://www.ibm.com/developerworks/cn/edu/ws-dw-ws-jax.html. Nov, 2007.

[37] 高尚.基于 Web 服务动态合成框架的服务组装引擎研究及实现[D].合肥工业大学硕士学位论文.合肥工业大学，2007,6.

[38] OASIS. Web Services Business Process Execution Language Version 2.0.January, 2007.

[39] HBCMSMIS 项目组. 淮北矿业（集团）有限公司煤矿安全管理信息平台研制工作报告 V2.0. 合肥工业大学软件工程教研室, 2006,6.

[40] Axis Documention. http://ws.apache.org/axis/java/user-guide.html,May, 2005.

[41] ActiveBPEL Documention. http://www.active-endpoints.com/open-source-documentation.htm,2006.

[42] 牛新征. 基于 J2EE 的企业应用集成的关键技术研究与应用设计[D].电子科技大学.硕士学位论文. 2004. 3.

[43] 蒋效宇，周志逸.用 JCA 实现企业应用集成.网络信息技术[J]，2004. 121(2)：55-59.

[44] Harry, M.S.Planning the Reengineering of Legacy System [J].IEEE Software, 1995:12(1):24-34.

[45] Vijay K. Madisetti, Moinul H .Khan, et al. Reengineering Legacy Embedded Systems [J].IEEE Design and Test of C omputers, 1999, 16(2):38-47.

[46] Ettore Merlo, Jean-Francois Giratd, et al.Reengineering User Interfaces [J].IEEE Software, 1995, 12(1):64-73.

[47] Ian Warren, Jane Ransom.Renaissance: A Method to Support Software System Evolution [J].26th Annual International Computer Software and Applications C onference, IEEE Computer Society, 2002:415-421.

[48] Jesus Bisbal,Deirdre Lawless,et al.An Overview of Legacy Information System Migration[C].Fourth Asia-Pacific Software Engineering and International Computer Science Conference, IEEE computer Society.1997:529-530.

[49] Lei Wu, Houari Sahraoui, Petko Valtchev.Coping with Legacy System Migration Complexity[C].Proceedings of the 10th IEEE International Conference on engineering f Complex Computer Systems,IEEE Computer Society.2005:600-609.
[50] 张友生.遗留系统的评价方法和进化策略[J].计算机工程与应用.2003(13):29-35.
[51] 姜展，刘军.基于 CORBA 的遗留系集成模型的研究与应用[J].计算机工程与科学，2003,25(6):46-49.

附录　研究生期间参加的主要科研工作及成果

攻读硕士学位期间发表的论文

(1) 汪正海，袁兆山，李兴勇。基于 J2EE 平台的 Web 应用框架整合的研究与应用。计算机技术与应用进展。2007.8

(2) 汪正海，袁兆山，黄晓玲。Web Services 动态合成方案及其向 BPEL4WS 自动转换的框架。计算机应用研究。(已录用)。

(3) 李兴勇，袁兆山，汪正海。复杂报表生成系统实现技术研究。计算机应用，2007.7

攻读硕士学位期间参与的项目

(1) 淮北矿业集团煤矿安全管理信息平台，淮北矿业集团公司科技项目，项目编号：104-432332

(2) 环境管理信息平台，淮北矿业集团公司科技项目，项目编号：104-432371

(3) 煤质管理信息平台，淮北矿业集团公司科技项目，项目编号：104-432372