摘 要

大多数企业都有由过去遗留下来的异构的系统、应用、商务流程以及数据源构成的应用环境。业务一旦发生变化就得重新修改系统，因此，把企业内部各软件平台灵活地整合在一起，就成了 B2B 时代的一个趋势。随着 EAI 以及 Web Service 技术的成熟和发展，出现了面向服务的架构（Service Oriented Architecture, SOA）。SOA 是透过业务服务的概念来提供 IT 的各项基本应用功能，让这些服务可以自由地被排列组合、融会贯通，以便在未来能随时弹性配合新的需求而调整。但是，统一接口并未改变原来点对点的集成，另外，对比 Hub 的 SOA 系统，又缺乏可靠性、效率不和扩展性。而企业服务总线作为消息传递的主干道，将担负起消息路由、事件驱动和消息格式转换等作用。

本文首先介绍了企业应用集成的发展与主流 EAI 技术的原理与应用范围。接着在简单分析传统的企业应用集成技术缺点的基础上引出了 SOA 产生的原因，详细论述了 SOA 的定义、体系结构和 SOA 的特点、价值。

接着，论文又对支撑 SOA 架构的重要组件 ESB（Enterprise Service Bus,企业服务总线）进行了全面系统地介绍，说明了 ESB 在面向服务的体系结构中所扮演的角色和主要功能。通过 ESB 对 SOA 体系结构的改进的研究与分析，归纳出了一种基于 ESB 企业应用集成模型。该模型主要包括：外部的服务请求者、外部的服务提供者、内部的服务请求者、内部的服务提供者、企业服务总线、业务服务目录、业务服务管理、ESB 网关等组成部分。

本论文的课题背景是基于中国人民银行的国库信息处理系统（Treasury Information Process System，TIPS），TIPS 是建立在财政、税务与国库各自独立的信息系统之上的信息交换和处理系统。

最后，把基于 ESB 企业应用集成模型应用到人民银行的国库信息处理系统中，有效的解决了该系统与外联系统集成过程中所遇到的难题，并总结了改模型需要进一步研究的一些方向。

关键词 EAI； SOA； 企业服务总线； 银行系统；遗留系统

Abstract

Lots of enterprises have application environment consist of many legacy isomerous systems、application and data sources now. So we'd better have all the software platforms integrated together in this B2B era. The business process will relative to more than one system. With the development of EAI technology and web service, the SOA architecture becomes the typical structure of EAI. These enterprise applications, which can't adapt the dynamic environment, will lose functions gradually, even to those business process manager systems that face to demand change frequently. It provides the basic business function by service, and the resources are available to be invoked by another service in the network. But the unified interface still can't change the pattern of point to point, and compare to the Hub structure, the hub lacks of the reliability、efficiency and expansibility. As the most important part message routing, Message Bus take charge of the message routing、mediation and event-driven.

In this thesis, a systematic survey of EAI comes to being and development research is given firstly, and then the cause of SOA was analyzed. The definition、architecture、characteristic and value of SOA was introduces detailed.

Secondly, the thesis discusses the overall of ESB, including the main function of ESB and the role which ESB acts in the SOA architecture. Then, the author proposes a new EAI model based ESB. It is includes: External/internal requestors and providers, ESB, Business Service Directory, Business Service Choreography, and ESB Gateway.

The background of the thesis is on Treasury Information Process System (TIPS), which is a core business system. of People's Bank of China. TIPS is used to exchange information among bank, exchequer and revenue. The state treasury department may avoid information isolated island by using it. For the finance, the state treasury and the tax affairs department carry on the forecast and the generalized analysis provide the foundation.

Finally, we apply the model based on ESB to the Treasury Information Process System, and obtain the certain application effect, and summarize some directions which we needs further to study.

Keywords SOA; ESB; EAI; banking system; legacy

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，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，也不包含为获得北京工业大学或其它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了谢意。

签名：郝嘉 日期：2007.6.13

关于论文使用授权的说明

本人完全了解北京工业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：郝嘉 导师签名： 日期：2007.6.13

第1章 绪论

1.1 研究背景

1.1.1 企业应用集成的产生

据 IDC 统计，在过去的 10 年中，全球企业在信息系统上一共投资 18 万亿美元。巨大的投资为企业建立了众多的信息系统，以帮助企业进行内外部业务的处理和管理工作。企业多年“自发”式的信息化建设，缺乏“自觉”的信息系统总体战略规划，致使局部需求得到满足的同时，企业内部信息系统各自为政，相互无法互通互联，形成“信息孤岛”，企业无法实现对整体业务运作和流程管理的全面掌控。根据 META Group 的统计，一家典型的大型企业平均拥有 49 个应用系统，而 33%的 IT 预算花在维护这些系统上[1]。

孤立的信息系统无法有效地提供跨部门、跨系统的综合性的信息，诸如：某个主要的订单的状况怎样？谁是我的最重要的客户？这个季度的任务能否完成？等等。孤立的信息系统也无法实现实时的信息存取和对业务流程的透视，无法实现对客户、供应商、项目、订单、资产等的全面掌控，无法实现企业价值链的全面的、彻底的透视和控制。

在二十世纪六十年代到七十年代期间，企业应用大多是用来替代重复性劳动的一些简单设计。当时并没有考虑到企业数据的集成，惟一的目标就是用计算机代替一些孤立的、体力性质的工作环节。

到了二十世纪八十年代，有些公司开始意识到应用集成的价值和必要性。这是一个巨大挑战，很多公司的技术人员都试图在企业系统整体概念的指导下对已经存在的应用进行重新设计，以便让它们集成在一起。然而这种努力收效甚微。

二十世纪九十年代，ERP 应用开始流行的时候，同时也要求它们能够支持已经存在的应用和数据，这就必须引入 EAI[2,3]。对 EAI 的需求首先来自与企业将它们的主机系统转换成 C/S 结构系统的过程中，其次是利用 ERP 建立企业骨干信息系统时。企业迫切需要一种方法，让它们少写程序，无须花费巨大的费用，就可以将各种旧的应用系统和新的系统集成起来，而其他推动 EAI 市场的因素还有供应链管理（B2B 集成）、基于流程的业务处理以及 Web 应用集成。

EAI 与电子商务的结合为企业快速实现业务的自动化提供了可靠的保证，呈现在我们面前的将是一个同时具有数据自动化和业务流程高度可塑的企业管理框架，从而进一步加快端到端的电子商务应用集成，包括供应链管理、客户关系管理和 ERP 系统相关联的门户网站、前端应用、后端应用等。

1.1.2 企业应用集成的发展过程

在企业商业需求的推动下，EAI 技术的演变经历了十多年的时间，产生了几代从不成熟到逐渐成熟的 EAI 技术，为企业带来不断增长的商业价值。下面对 EAI 技术的演变过程进行简要说明[4,5]：

（1）二十世纪八十年代，企业规模开始扩大，企业业务和数据日趋复杂，一些公司开始意识到应用集成的价值和必要性，很多公司的技术人员试图在企业系统整体概念的指导下对已经存在的应用进行重新设计，以便将它们集成在一起。此时，点到点（Point-to-Point）的集成技术开始出现，在各个应用系统之间通过各自不同的接口进行点到点的简单连接，实现信息和数据的共享。点到点的应用集成也被称为第一代 EAI 技术。该技术是要在不同系统之间的连接是硬编码实现的，维护成本高；随着企业的不断发展，需要集成的应用系统越来越多，使用这种技术会使应用系统集成的复杂度呈指数增长，如果要增加一个新的系统，则需要更改所有相关的接口，直接影响了其他系统的运行。

（2）二十世纪八十年代末和九十年代初，随着企业规模的进一步扩大，应用系统不断增加，简单的点到点连接已经很难满足不断增长的应用集成要求，企业迫切需要新的集成方法：可以少写代码，无须巨额花费，就可以将各种旧的应用系统和新的系统集成起来。第 2 代 EAI 技术的出现在一定程度上解决了这些问题，它采用 CORBA/DCOM、MOM（消息中间件）等技术，实现了对企业信息的集成，促进了企业的进一步发展。

但是，第二代 EAI 技术的缺点也是明显的：难以协调跨越不同系统，不同合作者，不同人员之间的复杂交互规则，对于交互过程的异常和错误不能有效管理；业务流程规则往往在连接器中硬编码实现，规则和数据没有实现逻辑分离，企业业务流程的变化，会需要更改连接器的代码，维护成本高；不能直观了解集成系统的运转情况，没有直观的全局性的管理和监控界面。

（3）二十世纪九十年代中后期，企业业务的迅速发展以及与电子商务的结合对应用集成解决方案提出了更高的要求，局限于信息集成的第二代 EAI 集成技术很难实现企业业务流程的自动处理、管理和监控，基于业务流程管理/集成（BPM/BPI）的第三代 EAI 集成技术成为更加合适的集成选择方案。第三代 EAI 集成技术通过实现对企业业务流程的全面分析管理，可以满足企业与客户、合作伙伴之间的业务需求，实现端到端的业务流程，顺畅企业内外的数据流、信息流和业务流。第三代 EAI 集成技术是当前集成技术发展的主流。

与第二代 EAI 构架相比，基于 BPI 的第三代 EAI 技术优势在于对业务流程的支持。BPI 集成从流程入手，是由上至下的集成方法，而第二代 EAI 主要着眼于应用层和数据层的集成，是由下至上的集成方法。面向数据和应用集成的第二

代 EAI 技术大大削减以前点到点集成的成本，并提供了功能强大的消息中间件平台，但是，它适应新的商业模型的能力有限，对于企业业务流程的支持能力决定了它在竞争能力上的局限性；而第三代 EAI 技术以流程为中心，并涵盖了第一代技术，顺畅了企业核心流程，提供实时错误检测和管理，自动化异常管理，使企业具有端到端可视能力，并能快速适应市场变化。

1.2 国内外研究现状

当前国内外企业应用系统的集成的研究，主要可以从广度和深度两个方向来研究。不同的广度和深度，集成所实现的目标是不同的。从系统内部的集成，到系统间的集成，到企业间的集成，集成的难度和能够取得的效益也是不同的。在将近 10 年的 EAI 技术的发展道路上，形成了当前 EAI 技术的 2 条发展路线[6]：

从集成的广度上来看，从易到难有以下种类的集成：

- 部门内部的信息系统集成。
- 部门之间的信息系统集成。
- 企业级的信息系统集成。
- 与随机遇到的合作伙伴之间的信息系统集成。

从集成的深度上来说，从易到难有以下种类的集成：

- 数据的集成。
- 应用系统的集成。

1.3 银行业对 EAI 技术的需求

信息化是 21 世纪国民经济和社会发展的助力器，是国家现代化的基本标志，一也是一个国家综合国力的集中体现。金融信息化是国民经济和社会发展信息化的重要组成部分，也是金融现代化的重要手段。

我国银行业的信息、化建设起步比西方发达国家晚，它经历了从无到有、从单项业务到综合业务、从机器仿人工到推出各种人性化的服务、从单机布点到数据大集中的复杂历程，不仅在技术上紧跟信息化的前沿，在认识上也从肤浅的工具论到现在的信息化战略论。现实中，银行信息化的发展正在配合、推动或者触发金融业产业结构的调整和升级，从根本上改变着传统的金融业务处理模式。

在金融信息化领域，以信息通信技术的发展和成熟为基础的数据大集中是近年来的一个热门话题，并成为包括银行、证券、保险等行业在内的整个金融信息化的发展大趋势。所谓“大集中”是一种通俗、形象的说法，其实质就是数据的集中和系统的整合。它是提高银行业核心竞争力的重要基石，一方面能够对金融业务进行即时风险控制，另一方面支持新业务的大规模、低成本扩张。数据大集

中是银行业发展的后台支撑系统，能为用户带来最直观感受的则是建立在各种信息技术基础上的电子化金融业务。

当前，国内银行信息化的重点基本上围绕着核心业务系统的建设和改造、渠道整合技术的提升，以及各种管理系统的建设而展开。此时，各银行不约而同地遇到两方面问题：一方面，随着各种业务系统的不断改进和增加，导致了必须对其所对应的渠道系统也加以改变，这大大增加了业务系统的开发和维护成本，延长了产品创新和升级的周期；另一方面，各种新兴渠道应运而生，每新增一种渠道，则要开发新的渠道系统。如何集成各应用系统，加快渠道系统与银行核心业务系统、后台管理系统的连接，如何遵从“以账户为中心”向“以客户为中心”转化的趋势，打造适应银行未来发展的信息系统，如何加快信息系统的开发节奏，统一系统的数据标准、工作流标准和报文标准这些都已成为新一轮银行信息化建设所必须面对的重点课题。

一个银行的技术架构，取决于其整体的业务架构。能够从当前与未来银行业的业务需求的角度出发，同时能够站在银行总体战略上来进行部署的技术架构，才是一个稳健、高效的架构。业内专家认为，基于 EAI 的渠道整合模式，不仅仅满足银行当前的渠道整合和核心业务系统改造所带来的需求，更重要的是，借此可以构建起面向将来的可持续发展的基础架构平台，从信息化规划的角度，来支持银行业务的创新及服务质量和管理能力的提高，从而实现更为长远的战略目标。从“紧耦合”到“松耦合”过去，渠道系统与后台业务系统的整合是通过各种“前置”系统来实现的。比如，核心业务系统有核心业务系统的前置系统，中间业务系统有中间业务系统的前置系统等。

实际上，各种前置系统只是提供了各种业务系统的接口。而且，由于没有统一的标准，各种前置系统都有其自身的技术特征。在这样的基础之上，各种渠道系统对各种前置系统进行“点到点”的连接，从而形成了紧密的网状架构。在这种“牵一发而动全身”的紧密式耦合架构下，任何一点的变化，都会带来全局的改变。也就是说，任何一次业务系统的变化，都会再次导致渠道系统的重新开发或升级。EAI 技术的核心功能之一，就是用松散耦合的方式实现系统的互连互通，同时也保证各被连接系统的独立性。这样的方向，就给银行信息化提出了全新的要求：第一，核心业务系统会逐渐成为一个核心资源系统，它必须能够向其它业务系统提供服务。第二，对银行的市场响应能力和灵活性提出了更高的要求，银行必须对现有服务进行快速重组。第三，新业务的特点是从传统的“简单交易”向跨系统与应用的“流程”方向发展。在这一点上，EAI/BPI 所独具的流程管理技术，可以说填补了传统集成方法的空白。

由此，银行信息系统的开发也必须随之而发生相应的变化这样，新业务的发展就对银行产生了全新的要求：需要银行充分利用其核心业务积累起来的客户资

源，不再仅仅是银行与客户之间的两方关系，需要与第三方企业的有效配合和密切合作，还要求银行具有对市场的快速响应能力。

综上所述，需要一个灵活的、易管理的解决方案，来解决诸如银行、电力、电信等行业的 IT 基础架构问题，这是本文的立题依据。

1.4 论文的组织

本文研究内容围绕企业服务总线的在银行系统中的应用研究展开，包括本章绪论，全文共有五章，整个论文内容结构安排如下：

第一章：绪论。主要介绍了论文的研究背景和目的，明确了论文的主要研究内容。

第二章：面向服务体系架构。在本章讲述了 SOA 产生原因以及 SOA 的定义和组成，详细论述了 SOA 的特点和优势。

第三章：企业服务总线。研究并与早期的 Hub 结构对比，分析了新的基于总线的 SOA 集成技术，从而改进了 SOA 的原有体系结构，从根本上提升了企业应用集成的灵活性。

第四章：基于 ESB 的应用集成模型。本章给出了使用 ESB 来实现 SOA 企业应用集成的典型架构。其中包括 ESB 对当前混淆的 IT 环境的支持，包括 J2EE/.Net、SAP、CICS 等系统的支持。

第五章：基于 ESB 的集成架构的应用。本章主要介绍了如何将企业服务总线应用到银行的数据大集中的系统中，详细介绍了系统的设计与实现。

最后在结论部分总结了论文的研究工作，并给出了下一步的研究方向。

第2章 面向服务的体系架构

2.1 SOA 产生的原因

当前 EAI 技术主要解决的是分布性和异构平台的问题。我们可以发现，传统的企业应用集成存在着以下一些问题：

（1）基本是面向功能的企业集成：上述的集成技术，都是从基于功能定义的设计方式，只注重技术和内容去分类，偏重于各自实现的功能，没有强调 EAI 的各类集成间的相互关系和相互的影响，以及作为整体对企业 IT 的影响。虽然这种方式相对简单和直观，但从宏观和长远角度上难以形成灵活配置和扩展的体系结构，在管理上也不能提供适应多变的业务需求的有效 IT 管控框架。

（2）忽视了企业的流程设计和集成：如果把 IT 系统的流程逻辑看作是对企业流程的模拟，我们就会发现面向功能设计的构架中存在的问题。在系统设计时，传统方法往往简单的直接套用具体业务流程的实例，从流程功能定义角度考虑，针对具体流程设计专门的紧密偶合式的集中流程控制逻辑，一旦内部流程变更增加或者捆绑，就需要重新去开发新的业务逻辑而难以重用现有的应用资源，导致流程控制改动频繁，需求一变，就重新上项目。另外一方面，基于流程功能定义角度的设计方式往往缺乏全局性的宏观模型体系，仅仅考虑眼前的应用，可重用性差，也无法形成较为统一和通用的逻辑模型视图。

（3）缺乏必要的灵活性和适应性，扩展性较差：他们都是针对特定的软硬件结构、网络环境或者特定的应用领域而进行研究和开发。面对软硬件结构等的动态变化，系统的自适应能力太差，有些甚至表现为无能为力。同时他们是针对独立的应用提供服务，系统的体系结果都是面向特定应用的，体系结构于整个企业毫无关系，使得系统毫无通用性可言，系统在企业中的可实施性比较差。

（4）缺乏合理的粒度规则：EAI 开发需要选择合适的粒度。由于上述的技术中没考虑到组件要与现有的应用程序相匹配，经常会选择了错误的粒度进行需求分析和设计，从而捣毁子了开发结果难于达到先前要求。组件的粒度很大方面影响了信息模型，如果粒度大过大，则应用程序的内部有独立的商务逻辑和商业观点，在集成过程中，很难与别的应用程序相协调，如果粒度过小，则会带来不必要的复杂性。粒度问题也决定了商业流程逻辑的隔离性，它影响了适配器和中间件组件的作用范围，也会对操作点和操作层次的设计带来挑战。

由于当前 EAI 面临着许多困难和挑战，因此 EAI 迫切需要一个新的体系结构来解决这些问题。

随着企业计算的发展，企业级应用需求要求新的软件系统不再是从底层做

起，而只是依据企业逻辑需求重新组织已有的数据存储，将现有的数据和事务通过新的渠道，比如 internet 浏览器或者手持设备呈现给用户，另外，为了提高企业计算的高效性、可用性、规模性，现有的许多操作系统都是分布式操作系统，运行在许多机器之上。这样的企业级解决方案就必须协调运行在群组硬件之上，实现这种系统的一种方法就是将该系统组织成群组服务的模式，每一个服务都提供一组定义良好的功能集合。整个系统就被设计和实现为一组相互交互的服务，而将功能以服务的形式展现出来是该系统灵活性的关键。它使得系统中的某些服务能够充分利用其它的服务同时却无需考虑其物理位置。系统通过添加新的服务来不断的升级，这样就应运而生了面向服务的体系结（Service Oriented Architecture，SOA）。SOA 定义了构成系统的服务，通过描述服务之间的交互提供特定的功能特性，并且将服务映射为具体的某种实现技术。

2.2 SOA 的定义和体系结构

2.2.1 SOA 的定义

面向服务的体系结构（SOA）建立在分布式计算技术的基础上，这种体系结构本质上是动态的，它提供对服务的登记、发现和调用的支持[7]。许多行业分析家和专家将 SOA 定义为多种不同的形式，因而对 SOA 有着不同的理解。

1996 年，Gartner 最早提出 SOA 的思想，“面向服务的架构是一种客户机/服务器软件设计方法，其中应用由软件服务和软件服务使用者组成（也称为客户机或服务请求者）”。Gartner 为 SOA 描述的远景目标是:在于让 TT 变得更有弹性，以更快地响应业务单位的需求，实现实时企业（Real-Time Enterprise）。[8]

IBM 对 SOA 的定义为:面向服务的体系结构（service-oriented architecture）是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以以一种统一和通用的方式进行交互[9]。

SOA 架构基本的要求有三点：

- SOA 在相对较粗的粒度上对应用服务或业务模块进行封装与重用。
- 服务间保持松散耦合，基于开放的标准，服务的接口描述与具体实现关。
- 灵活的架构－服务的实现细节，服务的位置乃至服务请求的底层协议都应该透明。

SOA 是一种体系结构。在这种体系结构中，所有功能都定义为独立的服务，这些服务带有定义明确的可调用接口，可以以定义好的顺序调用这些服务来形成

业务流程。请注意这里的表述：

（1）所有的服务都是独立的。它们就像“黑匣子”一样运行：外部组件既不知道也不关心它们如何执行它们的功能，而仅仅关心它们是否返回期望的结果。

（2）所有功能都定义为服务。这仅仅包括业务功能、由底层功能组成的业务事务和系统服务功能。

（3）在其最一般的意义上来说，接口是可调用的：也就是说，在体系结构的层面上，它们究竟是本地的（在本系统内）还是远程的（在直接系统外）、是用什么互连 Scheme 或协议来调用或需要什么样的基础架构组件来连接，都是无关紧要的。服务可能是在相同的应用程序中，也可能是在公司内部网内完全不同的系统上的不对称多处理器的不同地址空间中。

2.2.2 SOA 的体系结构

面向服务的体系结构（SOA）建立在分布式计算技术的基础上，它本质上是动态的，它提供对服务的登记、发现和调用的支持。SOA 的体系结构如下图所示[10]：

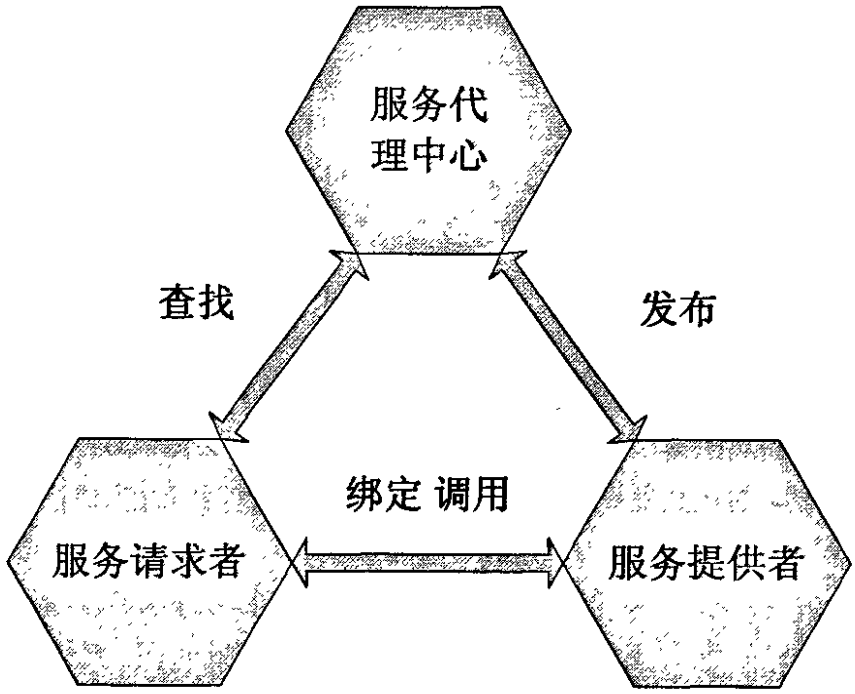


图 2-1 SOA 的体系结构

Figure 2-1 SOA architecture

SOA 是设计和构建松耦合的软件解决方案的方法，这个解决方案能够以程序化的可访问的软件服务的形式公开业务功能，以使其他应用程序可以通过已发布的和可发现的接口来使用这些服务，SOA 描述了三种角色[11]，分别承担了不

同的任务，在这三种角色上又有三个操作。

面向服务的体系结构中的角色包括：

- 服务请求者：服务请求者是一个应用程序、一个软件模块或需要一个服务的另一个服务。它发起对服务代理中心的服务的查询，通过传输绑定服务，并且执行服务功能。服务请求者根据接口契约来执行服务。
- 服务提供者：服务提供者是一个可通过网络寻址的实体，它接收和执行来自请求者的请求。它将自己的服务和接口契约发布到服务代理中心，以便服务请求者可以发现和访问该服务。
- 服务代理中心[12]：服务代理中心是服务发现的支持者。它包含一个可用服务的存储库，并允许感兴趣的服务使用者查找服务提供者接口。它集中存储服务信息，以便于服务请求者的查找。同时服务提供者可以把它们所要提供的服务在这里进行注册。对于服务请求者来讲，绑定服务信息的方式有两种：静态绑定和动态绑定。静态绑定是在开发应用程序的时候查询相关的服务信息，并得到服务的接口信息。在这种方式下，服务代理是可选的，因为服务请求者不必一定要从服务代理处获得服务提供者的访问位置，还有很多其它的方式同样可以获得服务提供者的信息，比如 FTP，URL，Email 等。动态绑定是指服务请求者在运行过程中从服务代理处获得服务信息并动态调用相关功能的过程。

面向服务的体系结构中的每个实体都扮演着服务提供者、使用者和注册中心这三种角色中的某一种（或多种）。

对应于 SOA 中的三个角色，SOA 也包括三种主要的操作：

- 发布：为了使服务可访问，需要发布服务描述以使服务请求者可以查找它。发布服务描述的位置可以根据应用程序的要求而变化。
- 发现：在发现操作中，服务请求者直接检索服务描述或在服务代理中心中查询所要求的服务类型。对于服务请求者，可能会在两个不同的生命周期阶段中牵涉到查找操作：在设计时为了程序开发而检索服务的接口描述，而在运行时为了调用而检索服务的绑定和位置描述。
- 绑定和调用：在绑定操作中，服务请求方通过分析从代理服务器中得到的服务绑定信息，包括服务的访问路径、服务调用的参数、返回结果传输协议、安全要求等，对自己的系统进行相应配置，进而远程调用服务提供者所提供的服务。

在面向服务的体系结构中，对服务有如下要求：

所有功能都定义为服务。这仅仅包括业务功能、由底层功能组成的业务事务和系统服务功能。

所有的服务都是独立的。它们就像“黑匣子”一样运行。外部组件既不知道

也不关心它们如何执行它们的功能，而仅仅关心它们是否返回期望的结果。

在其最一般的意义上来说，接口是可调用的。也就是说，在体系结构的层面上，它们究竟是本地的（在本系统内）还是远程的（在系统外）、是用什么互连协议来调用或需要什么样的基础架构组件来连接，都是无关紧要的。服务可能是在相同的应用程序中，也可能是在公司内部网内完全不同的系统上的不对称多处理器的不同地址空间中，还有可能是在用于 B2B 配置的合作伙伴的系统上的应用程序中。

通过以上论述，典型的 SOA 架构需有以下三个基本要求[13]：

（1）SOA 在相对较粗的粒度上对应用服务或业务模块进行封装与重用。

（2）服务间保持松散耦合，基于开放的标准， 服务的接口描述与具体实现无关。

（3）灵活的架构服务的实现细节，服务的位置乃至服务请求的底层协议都应该透明。

2.3 SOA 的特点和价值

2.3.1 SOA 的特点

松散祸合：SOA 允许独立的开发服务的提供者和消费者而不用考虑各自的技术组成。并且允许服务消费者在消费者和提供者所需交换信息的最低知识级别上定义和发现自己感兴趣的服务。

请求/响应：SOA 主要支持的交换为，一个特定的系统向服务提供者请求一段信息或者请求执行一段程序，服务提供者接到请求后发出响应以提供所请求的服务。

同步：SOA 主要支持同步的服务调用和执行。这意味者当消费者请求信息或执行程序时，在源和目的两个系统间必须维持一个链接直到消费者收到响应。

2.3.2 SOA 的价值

（1）业务价值，通过服务可以增加流程的颗粒度，从而带来业务的灵活性；面对市场的变化可以快速的创建业务流程并组装应用系统；无需担心 IT 的基础设施，通过使用服务就可以改进客户服务；SOA 可以增加业务变化的速度，改进业务效率和性能，保护关键信息资产的私密性和安全性，使得 IT 在安全和可管理的集成环境中以一种低成本的方式满足业务战略的要求[14]。

（2）IT 价值，使企业 IT 变成一个更具反应性、安全性和可管理的集成环

境的 IT 组织；通过使用预建的、可重用的服务构建模块，可以减少应用开发和部署的周期；通过通用服务可以降低维护的复杂性和维护成本；逐步过程现有 IT 系统，而不是完全替换他们；从 IT 的角度来看，挑战和益处包括：安全的和可管理的集成环境，通过使用预构建的、可重用的服务处理部件降低客户和部署的周期时间，以通用的服务减少复杂度和维护成本，增强现有的 IT 系统，而不是全部以新系统代替他们。

2.4 本章小结

本章主要介绍了当前EAI技术的发展和SOA体系结构的原理。首先通过EAI发展的瓶颈，引出 SOA 的体系结构，然后分析了 SOA 的原理，介绍了 SOA 的体系结构，SOA 定义了服务请求者，服务提供者和服务代理中心三个角色。最后介绍了 SOA 的特点和价值。

第3章 企业服务总线

3.1 企业服务总线的提出

上一章已阐述了什么是 SOA，按照 SOA 的思想，假设我们把企业内部和外部的所有需要集成的应用都封装成服务，公布出来，并可以相互调用服务，这样就可以在 J2EE 的环境里调用.Net，但是这点在原来没有 SOA 的时候也可以做到的。只要两个 IT 系统之间认可对方的方式，即使不存在公开/统一的服务界面也可以实现点到点的互联。因此我们不得不承认，如果我们只有服务，而服务的请求者和服务的提供者之间仍然需要这种显式的点到点的调用，那么这就不是一个典型的 SOA 架构。服务的参与双方仍然都必须建立 1 对 1 的联系，如下图所示：

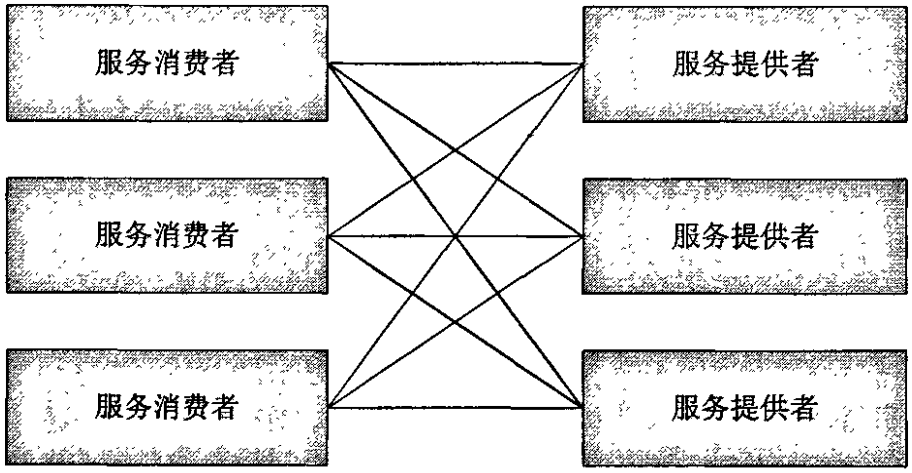


图 3-1 点到点的服务集成

Figure 3-1 Point-to-Point service integration

因此，在 SOA 中，我们还需要这样一个中间层，能够帮助实现在 SOA 架构中不同服务之间的智能化管理。最容易想到的是这样一个 Hub-Spoke 结构，在 SOA 架构中的各服务之间设置一个类似于 Hub 的中间件，由它充当整个 SOA 架构的中央管理器的作用。请看图三，现在服务的请求者和提供者之间有了一个智能的中转站，服务的请求者不再需要了解服务提供者的细节。不错！看上去是一个好的 SOA 结构。事实上，传统的 EAI 就是通过这样一种方式来试图解决企业内部的应用整合问题。

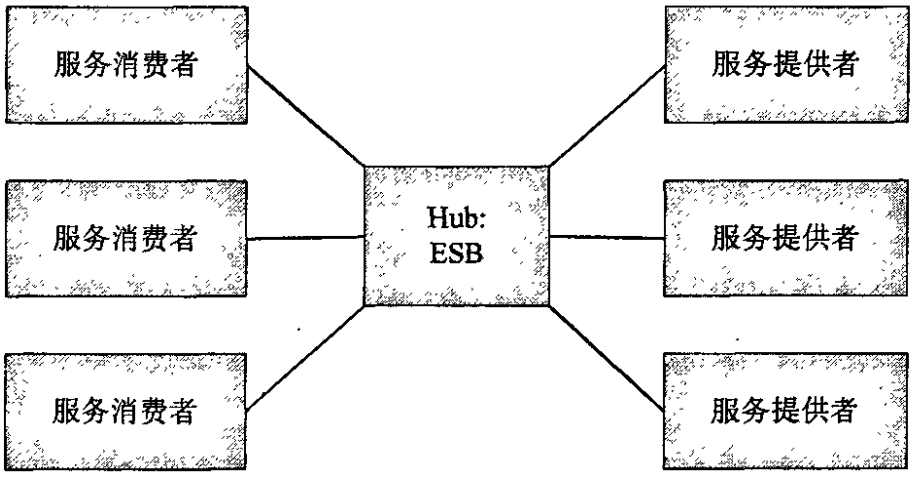


图 3-2 Hub-Spoke 结构的集成

Figure 3-2 Hub-Spoke structure integration

EAI 的目标是支持对现有 IT 系统的重新利用，通过 EAI 技术能够将不同的软件和系统串联起来，延长这些应用系统的生命周期。传统的 EAI，往往使用如 CORBA 和 COM 等的消息中间件进行分布式，跨平台的程序交互，修改企业资源规划以达到新的目标，使用中间件、XML 等方法来进行数据分配。因此，实际上传统的 EAI 是部件级的重用。很不幸的是，基于部件的架构没有统一的标准，因此，各个厂商都有各自不同的 EAI 解决方案，你会看到各种各样的中间件平台。如果 EAI 碰到了异构的 IT 环境，就必须分别考虑怎样在各个不同的中间件之间周旋，来实现合理的互联方式，你不得不考虑各种复杂的可能性。因此，你所见过的大多数传统 EAI 解决方案都比较笨重。

如果选择 Hub 的模式来构建 SOA 基础架构，从纯粹逻辑的角度，可能会出现哪些问题呢？首先，整个 SOA 架构的性能，如果每个服务的请求都经过中央 Hub 的中转，那么 Hub 的负担会很重，速度会随着参与者的增多而愈来愈慢；其次，这样的系统会很脆弱，一旦 Hub 出错，整个 SOA 架构都会瘫痪；最后，这样的架构会破坏 SOA 的开放性原则，参与者运行在一个相对封闭的环境中，扩展起来十分麻烦。因此，这也不是理想的 SOA 架构。

ESB 可以很好的解决 HUB-Spoke 模式所面临的问题。首先，它比单一 Hub 的形式更开放，总线结构有无限扩展的可能；其次，真正体现了 SOA 的理念，一切皆为服务，服务在总线（BUS）中处于平等的地位。即使我们需要一些 Hub，那么它们也是以某种服务的形式部署在总线上，相比上面的结构要灵活的多。ESB 是一种在松散耦合的服务和应用之间标准的集成方式。

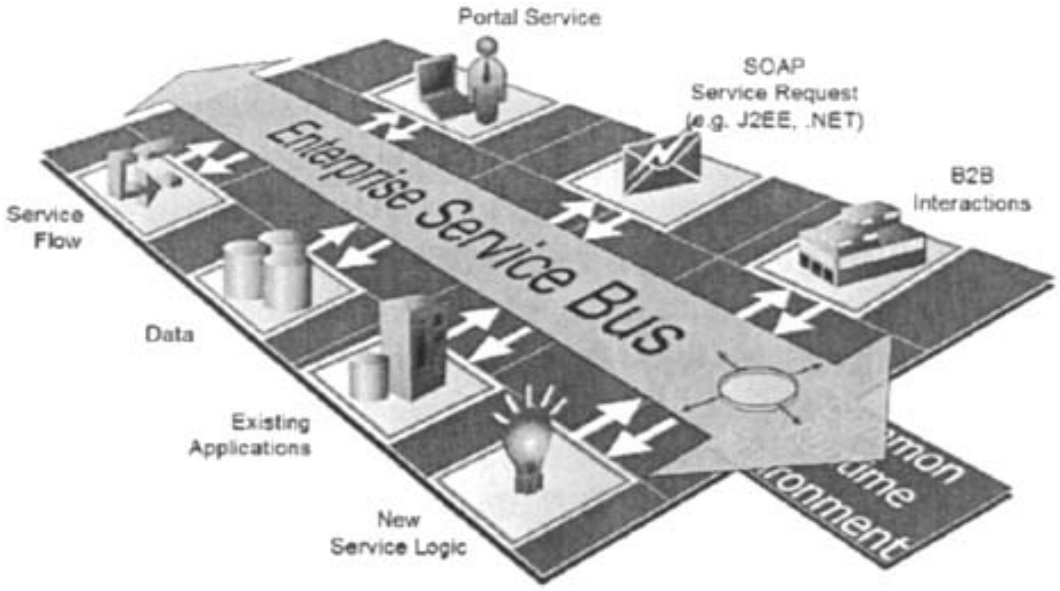


图 3-3 利用企业服务总线集成

Figure 3-3 ESB integration

3.2 企业服务总线的定义

ESB(企业服务总线)[15,16]定义通常如下：是基于中间件技术实现并支持 SOA 的一组基础架构功能，它是连接企业各种纷繁复杂应用的骨干神经系统。 支持异构环境中的服务、消息以及基于事件的交互，并且具有适当的服务级别和可管理性。

ESB 是一种使得企业应用具有被组织机构内外的其它应用重用能力的框架；是一个实现了通信、互连、转换、可移植性和安全性标准接口的企业基础软件平台；是一种能够在一个框架结构中实现统一并连接服务、应用和资源的中间件模式[17,18]。ESB 通过 Web 服务[19]、J2EE，.NET 和其它标准提供更强的系统互连功能。可以这样说，ESB 是特定环境下（SOA 架构中）实施 EAI 的方式：

首先，在 ESB 系统中，被集成的对象被明确定义为服务，而不是传统 EAI 中各种各样的中间件平台，这样就极大简化了在集成异构性上的考虑，因为不管有怎样的应用底层实现，只要是 SOA 架构中的服务，它就一定是基于标准的。

其次，ESB 明确强调消息（Message）处理在集成过程中的作用，这里的消息指的是应用环境中被集成对象之间的沟通。以往传统的 EAI 实施中碰到的最大的问题就是被集成者都有自己的方言，即各自的消息格式。作为基础架构的 EAI 系统，必须能够对系统范畴内的任何一种消息进行解析。传统的 EAI 系统中的消息处理大多是被动的，消息的处理需要各自中间件的私有方式支持，例如 API 的方式。因此尽管消息处理本身很重要，但消息的直接处理不会是传统 EAI

系统的核心。ESB 系统由于集成对象统一到服务，消息在应用服务之间传递时格式是标准的，直接面向消息的处理方式成为可能。如果 ESB 能够在底层支持现有的各种通讯协议，那么对消息的处理就完全不考虑底层的传输细节，而直接通过消息的标准格式定义来进行。这样，在 ESB 中，对消息的处理就会成为 ESB 的核心，因为通过消息处理来集成服务是最简单可行的方式。这也是 ESB 中总线（Bus）功能的体现。其实，总线的概念并不新鲜，传统的 EAI 系统中，也曾经提出过信息总线的概念，通过某种中间件平台，如 CORBA 来连接企业信息孤岛，但是，ESB 的概念不仅仅是提供消息交互的通道，更重要的是提供服务的智能化集成基础架构。

最后，事件驱动成为 ESB 的重要特征。通常服务之间传递的消息有两种形式，一种是调用（Call），即请求/回应方式，这是常见的同步模式。还有一种我们称之为单路消息（One-way），它的目的往往是触发异步的事件， 发送者不需要马上得到回复。考虑到有些应用服务是长时间运行的，因此，这种异步服务之间的消息交互也是 ESB 必须支持的。除此之外，ESB 的很多功能都可以利用这种机制来实现，例如，SOA 中服务的性能监控等基础架构功能，需要通过 ESB 来提供数据，当服务的请求通过 ESB 中转的时候，ESB 很容易通过事件驱动机制向 SOA 的基础架构服务传递信息。

3.3 企业服务总线的功能模型

3.3.1 企业服务总线的功能

表 3-1 对现有文献中确定的一些 ESB 功能进行了总结和分类[20]。虽然有一些功能非常基础，但是其他的功能（如自动化功能或智能化功能）代表着向按需操作环境转变的重要步骤。重要的是认识到，当前的大多数场景只需要部分类别中的部分功能。

表 3-1 ESB 功能模型

Table 3-1 Categorized ESB capabilities

通信	服务交互
● 路由 ● 寻址 ● 通信技术、协议和标准（例如 IBM® WebSphere® MQ、HTTP 和 HTTPS） ● 发布/订阅 ● 响应/请求	● 服务接口定义（例如，Web 服务描述语言（Web Services Description Language，WSDL）） ● 支持替代服务实现 ● 通信和集成所需的服务消息传递模型（例如 SOAP 或企业应用程序集成

● Fire-and-Forget，事件 ● 同步和异步消息传递	（EAI） 中间件模型） ● 服务目录和发现
集成	服务质量
● 数据库 ● 服务聚合 ● 遗留系统和应用程序适配器 ● EAI 中间件的连接性 ● 服务映射 ● 协议转换 ● 应用程序服务器环境（例如 J2EE 和.NET） ● 服务调用的语言接口（例如 Java 和 C/C++/C#）	● 事务（原子事务、补偿、Web 服务事务（WS-Transaction）） ● 各种确定的传递范例（例如 Web 服务可靠消息传递（WS-ReliableMessaging）或对 EAI 中间件的支持）
安全性	服务级别
● 身份验证 ● 授权 ● 不可抵赖性 ● 机密性 ● 安全标准（例如 Kerberos 和 Web 服务安全性（WS-Security））	● 性能 ● 吞吐量 ● 可用性 ● 其他可以构成契约或协定的持久评估方法
消息处理	管理和自治
● 编码的逻辑 ● 基于内容的逻辑 ● 消息和数据转换 ● 有效性 ● 中介 ● 对象标识映射	● 服务预置和注册 ● 记录、测量和监控 ● 发现 ● 系统管理和管理工具的集成 ● 自监控和自管理
建模	基础架构智能
● 对象建模 ● 通用业务对象建模 ● 数据格式库 ● B2B 集成的公共与私有模型 ● 开发和部署工具	● 业务规则 ● 策略驱动的行为，特别是对于服务级别、服务功能的安全和质量（例如 Web 服务策略（WS-Policy）） ● 模式识别

上面的许多功能既可以使用专有技术实现，也可以通过利用开放标准实现。然而，使用不同的技术来实现 ESB 可能会使它们的性能、可伸缩性和可靠性这

些特性显著不同，同时 ESB 功能和所支持的开放标准也会有所不同。由于这些原因，再加上最近制订和正在兴起的一些相关标准，当今实现 ESB 的许多关键决策都涉及到成熟的专有技术和不成熟的开放标准之间的权衡。

3.3.2 企业服务总线的最低功能

如果在前面确定的功能中只有一部分和大多数 SOA 场景相关，我们可能会问：实现 ESB 所需的一组最低功能由什么构成？为此，考虑最被普遍认同的 ESB 定义的原理：

ESB 是一种逻辑体系结构组件，它提供与 SOA 的原则保持一致的集成基础架构。SOA 原则需要使用与实现无关的接口、强调位置透明性和可互操作性的通信协议、相对粗粒度和封装可重用功能的服务定义。

ESB 可以作为分布式的异构基础架构进行实现。ESB 提供了管理服务基础架构的方法和在分布式异构环境中进行操作的功能[21]。

表 3-2 ESB 最低功能模型

Table 3-2 Minimum capabilities for the ESB

通信	集成
● 提供位置透明性的路由和寻址服务 ● 控制服务寻址和命名的管理功能 ● 至少一种形式的消息传递范型（例如，请求/响应、发布/订阅等等） ● 支持至少一种可以广泛使用的传输协议	● 支持服务提供的多种集成方式，比如 Java 2 连接器、Web 服务、异步通信、适配器等等
服务交互	
● 一个开放且与实现无关的服务消息传递与接口模型，它应该将应用程序代码从路由服务和传输协议中分离出来，并允许替代服务的实现。	

3.4 企业服务总线的应用模式

ESB 提供了交互点，服务可以在此将消息放到总线上或从总线取走。它会对动态消息应用中介，并保证这些托管交互的 QoS。从 ESB 的角度来看，所有的服务交互端点都是类似的，因为它们都发送或处理请求/事件；它们都要求特定的 QoS；它们可能都需要交互协助。ESB 模式允许集成开发人员以与处理新业务逻辑、流程编排组件或外部 Web 服务同样（面向服务）的方式对待与用户交互的请求者或提供者。

用于构建基于 ESB 的解决方案的模式分为以下几类[22]：

- 交互模式：允许服务交互点将消息发送到总线或从总线接收消息。
- 中介模式：允许对消息交换进行操作。
- 复合模式：中介模式和交互模式进行组合，以实现更为复杂的模式。

3.4.1 交互模式

ESB 允许端点通过总线以其本机交互模式进行交互。它支持各种端点协议和交互方式。交互模式的例子包括[15]：

- 请求/响应：处理端点间的请求/响应方式的交互。此 ESB 基于消息传递模型，因此由两个相关的单向消息流对请求/响应交互进行处理，一个用于请求，一个用于响应。
- 请求/多响应：上述类型的变体，可以发送多个响应。
- 事件传播：事件可以匿名分发到由 ESB 管理的相关方列表。服务可以将自身添加到该列表中。

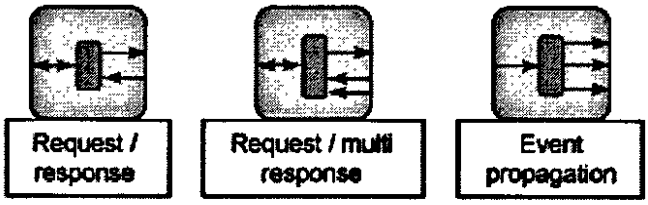


图 3-4 交互模式

Figure 3-4 The inter-operational pattern

3.4.2 中介模式

中介模式处理总线上的动态消息（请求或事件）。由请求者发出的消息会转换为稍微有些不兼容的提供者（从潜在的端点集中选择）能够理解的消息。

这些中介操作单向消息而不是请求/响应对，因为 ESB 将交互模式放在中介模式上。中介有多种基本模式；更为复杂的模式可以通过组合简单模式构建：其中包括：协议变换、转换、补偿、路由、监视和关联。

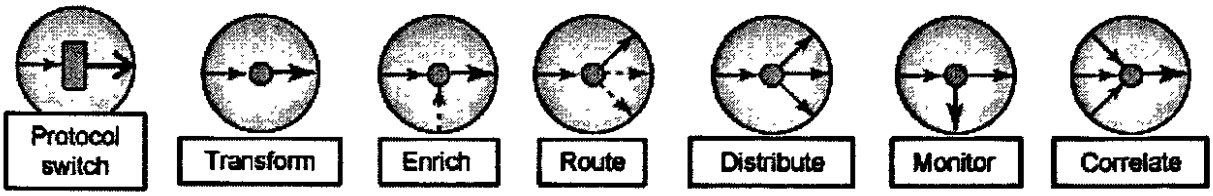


图 3-5 中介模式

Figure 3-5 The broker pattern

3.4.3 复合模式

中介模式和交互模式可以进行组合，以实现更为复杂的模式。例如，在协议变换后转换格式可以实现规范化适配器模式，在这种模式中，所有相关方使用的消息和业务对象集都标准化为规范的格式。规范化适配器模式将端点的本机总线附加协议转换为标准协议，实现有效负载规范化，并在交付时进行这些转换的反向转换。

另一种常见的复杂中介是转换、记录和路由模式。网关模式是一个复杂的协议变换变体。它可以合并转换和监视中介，以提供加密、日志记录或审核等功能。它还可以对一对多关系中的消息进行聚合和反聚合。服务门户是此类模式的代表，它为多个服务提供单一联系点，并隐藏内部服务的细节。

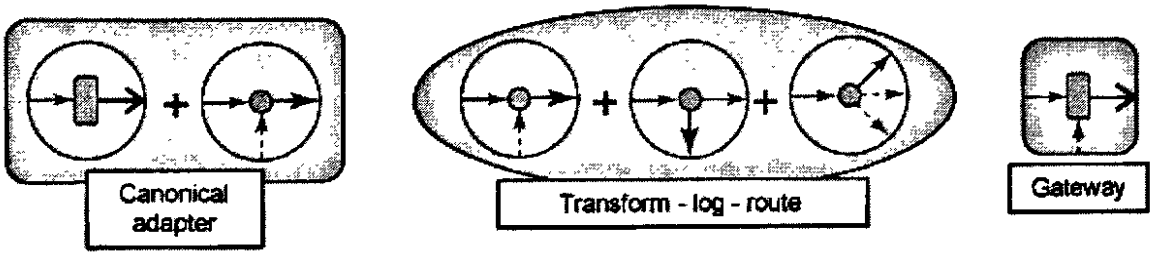


图 3-6 复合模式

Figure 3-6 The complicated pattern

3.5 本章小结

本章在前一章面向服务的体系结构的理论基础上，分析了已有企业应用集成中的业务需求，引出了企业服务总线（ESB）的概念，提出了 ESB 对传统 EAI 技术的改进和能够解决的问题，最后分析并总结了企业服务总线的功能模型和它的应用模式。

第4章 基于 ESB 的应用集成模型

4.1 数据集成

企业间的应用系统的集成可以呈现许多种形式，如用户界面集成、数据集成、函数和方法集成等。合适的集成形式依赖于许多因素，包括企业的计算机应用基础、企业的行业类别、企业应用系统的构架、企业内部应用系统间的集成度等等。通常，比较适用的是采用松散耦合的形式实现应用系统之间的数据集成。

数据集成的基础是数据交换。为了实现数据交换，对于松散耦合而言，尤其是在 Internet 环境卜，需要有一种适合 Internet 环境的消息交换协议。

在 XML[24]之前，就有一种用来描述数据的标记语言—SGML（Standard Generalized Markup Language），它可以对各种类型的数据进行描述。SGML 在 Internet 出现以前就已经有。随着 Internet 的出现，人们开始把 SGML 的运用转移到 Web 上来。但是 SGML 是一个十分庞大复杂的语言格式，用这种格式描述的数据不利于在网上传输。XML 便应运而生。XML 是 SGML 的简化版，是一种用来描述数据的语言，它略去了 SGML 中繁杂和不常用的内容，从而使得编程简单化，易理解，易于在 Web 上的传输和交互。它作为 SGML 的一个重要分支，也能像其它的 SGML 文件一样被解释和验证其有效性。

XML 解决了 HTML 不能解决的两个 Web 问题，即 Internet 发展速度快而接入速度慢的问题，以及可利用的信息多，但难以找到自己需要的那部分信息的问题。XML 能增加结构和语义信息，可使计算机和服务器即时处理多种形式的信息。因此，运用 XML 的扩展功能不仅能从 Web 服务器下载大量的信息，还能大大减少网络业务量。

XML 具有以下的主要特点[25,26]：

- 跨平台性：XML 是经过检验的国际标准，使用文本来保存数据，而不是使用二进制格式，因此对应用于跨平台的数据交换是十分方便的。
- 强大的数据表现能力：能够以简单的形式表达复杂的数据。
- 自描述能力强：便于系统和人对数据的理解，同时实现了数据与表现形式的分离。
- 对语义的表示能力强：对一种数据类型，可以用属性对其进行多方面的描述。如名称、数据类型、单位、格式等，所以 XML 可通过增加元素的属性来加强数据的语义表示能力。
- 可扩展性强，用户可以根据自己的需要设计标签集合并创建 XML 文档，对于存储和交换丰富多样的数据信息具有重要意义。

因为 XML 具有以上很多优点，所以它已经成为不同的应用和组织之间描述共享结构化数据的业界标准。在以 ESB 进行应用集成时应采用 XML 格式的消息与不同应用系统进行数据交换和信息共享。

在进行企业集成的过程中会遇到异构数据格式的问题，XML 与其他数据源之间的转换，可以分为三中情况[27]：

XML 到关系型数据库的转换：关系型数据库是 XML 的最主要数据来源和目标。就目前的主流关系型数据库来看，数据库本身已经提供了良好的 XML 接口，用户已经不必再自己开发相应的转换功能。例如，在 DB2 中，用户可以使用 SQL/XML 查询语言从关系数据直接创建 XML 文档，包括 XML 标签、属性，以及在创建文档的同时完成串联和聚合等操作。

XML 到 XML 的转换 XSLT 技术很好的解决不同 XML 之间的转换问题，这一技术也已经在多种产品中得到了实现，例如 Oracle，Micosoft.NET 等等。用户可以直接使用这些产品提供的转换功能。

XML 到其他异种数据源之间的转换[28]：除关系型数据库和 XML 以外，还有其它的数据源，例如自定义格式的文件，非主流数据库系统等等。这一类数据源需要单独开发相应的到 XML 的转换功能。今后的 XML 与其他数据源之间的转换工作，将主要集中在这一类与应用相关的异种数据源上。

4.2 中间件

为解决分布异构问题，人们提出了中间件（Middleware）的概念[29]。中间件是一类软件，而非一种软件；中间件不仅仅实现互连，还要实现应用之间的互操作；中间件是基于分布式处理的软件，定义中特别强调了其网络通讯功能[30]。中间件提供客户机与服务器之间的连接服务，这些服务具有标准的程序接口和协议。针对不同的操作系统和硬件平台，它们可以有符合接口和协议规范的多种实现。

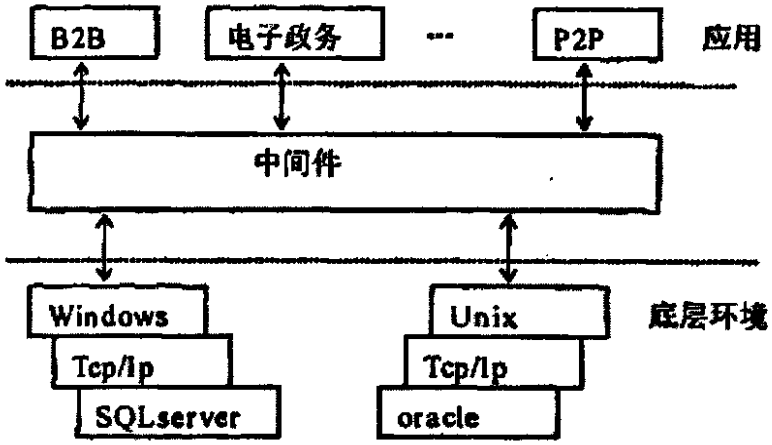


图 4-1 中间件示意图

Figure 4-1 The role of middleware

世界著名的咨询机构[31]The Standish Group 在一份研究报告中归纳了中间件的十大优越性：

（1）应用开发，The Standish Group 分析了 100 个关键应用系统中的业务逻辑程序、应用逻辑程序及基础程序所占的比例；业务逻辑程序和应用逻辑程序仅占总程序量的 30%，而基础程序占了 70%，使用传统意义上的中间件一项就可以节省 25%～60%的应用开发费用。如是以新一代的中间件系列产品来组合应用，同时配合可以复用的商务对象构件，则应用开发费用可节省至 80%。

（2）系统运行，没有使用中间件的应用系统，其初期的资金及运行费用的投入要比同规模的使用中间件的应用系统多一倍。

（3）开发周期，基础软件的开发是一件耗时的工作，若使用标准商业中间件则可缩短开发周期 50%～75%。

（4）减少项目开发风险，研究表明，没有使用标准商业中间件的关键应用系统开发项目的失败率高于 90%。企业自己开发内置的基础（中间件）软件是得不偿失的，项目总的开支至少要翻一倍，甚至会十几倍。

（5）合理运用资金，借助标准的商业中间件，企业可以很容易地在现有或遗留系统之上或之外增加新的功能模块，并将它们与原有统无缝集合。依靠标准的中间件，可以将老的系统改头换面成新潮的 Internet/Intranet 应用系统。

（6）应用集合，依靠标准的中间件可以将现有的应用、新的应用和购买的商务构件融合在一起进行应用集合。

（7）系统维护，需要一提的是，基础（中间件）软件的自我开发是要付出很高代价的，此外，每年维护自我开发的基础（中间件）软件的开支则需要当初开发费用的 15%～25%，每年应用程序的维护开支也还需要当初项目总费用的 10%～20%左右。而在一般情况下，购买标准商业中间件每年只需付出产品价格的 15%～20%的维护费，当然，中间件产品的具体价格要依据产品购买数量及哪

一家厂商而定。

（8）质量，基于企业自我建造的基础（中间件）软件平台上的应用系统，每增加一个新的模块，就要相应地在基础（中间件）软件之上进行改动。而标准的中间件在接口方面都是清晰和规范的。标准中间件的规范化模块可以有效地保证应用系统质量及减少新旧系统维护开支。

（9）技术革新：企业对自我建造的基础（中间件）软件平台的频繁革新是极不容易实现的（不实际的）。而购买标准的商业中间件，则对技术的发展与变化可以放心，中间件厂商会责无旁贷地把握技术方向和进行技术革新。

（10）增加产品吸引力：不同的商业中间件提供不同的功能模型，合理使用，可以让你的应用更容易增添新的表现形式与新的服务项目。从另一个角度看，可靠的商业中间件也使得企业的应用系统更完善，更出众。

随着计算机软件技术的发展，中间件技术也已经日渐成熟，并且出现了不同层次、不同类型的中间件产品。按照 IDC 的分类方法，中间件可分为六类[32]。

分别是数据访问中间件、远程过程调用中间件、消息中间件、交易中间件、对象中间件等。

（1）数据访问中间件，是为了建立数据应用资源互操作的模式，对异构环境下的数据库实现联接或文件系统实现联接的中间件。

（2）远程过程调用中间件，通过这种远程过程调用机制，程序员编写客户方的应用，需要时可以调用位于远端服务器上的过程；消息中间件：用来屏蔽掉各种平台及协议之间的特性，进行相互通信，实现应用程序之间的协同；

（3）交易中间件，是在分布、异构环境下提供保证交易完整性和数据完整性的一种环境平台。

（4）对象中间件，在分布、异构的网络计算环境中，可以将各种分布对象有机地结合在一起，完成系统的快速集成，实现对象重用。

4.3 服务的构建

作为 SOA 体系结构中，将企业遗留的各种 IT 资产都封装成统一平台是实现 SOA 体系集成的基石。通过对企业目前 IT 环境的调研，选择不同的技术，将企业应用打包成一个一个面向服务的组件，使企业迁移到 SOA 体系结构中。其中，构建服务包括在开发应用的时候直接开发成服务，也可以把遗留系统封装成服务。

服务构造的粒度大小根据其和其他子系统之间的业务关联来决定。可以对系统原有的旧的应用进行改造，使其封装为服务，也可以根据需要开发新的服务。服务的类型有 5 种：

- 数据访问：允许统一访问不同的数据源。
- 组件：提供对打包应用程序服务的访问，如对企业资源规划（ERP）的访问。
- 业务：提供复杂的服务，这些服务使用多个打包或定制应用程序的功能。
- 合成：使用上述三种类型类创建一个新的服务。该服务既含有新的功能，也含有现有的功能。
- 共享的或企业的基础架构服务：低级服务，例如消息日志记录，其重用可快速创建新的高级服务。

4.3.1 基于 J2EE 和.Net 的自定义应用系统与异构数据源

随着 Web 服务的出现，后端企业应用程序通过使用 WSDL 被公开为可发现并可调用的基于 EJB 和 COM 的业务服务。WSDL 为 Web 服务接口定义了服务语义，例如操作、协议绑定和消息类型等。通过将业务逻辑组件改写成业界统一标准的 WSDL 文件，供外部服务调用。

（1）封装成 web 服务，在 J2EE 和.Net 的应用系统中，需要将 J2EE 的 JavaBeans 和 C#文件等后台业务逻辑组件中的服务名称、访问协议类型、业务逻辑、在 WSDL 文件的各个元素中分别对应 J2EE 和.Net 的业务逻辑中的元素：

（2）J2EE 连接体系结构，JCA（J2EE Connector Architecture）即 Java 连接器架构，是 J2EE 规范的组成部分。JCA 提供了基于 Java 的标准架构，使 J2EE 平台能够连接到异构的 EIS（Enterprise Information Systems，即企业信息系统），从而解决企业应用集成（EAI）的问题。JCA 通过定义一组可伸缩的、安全的、支持交易的机制，实现了企业信息系统与应用服务器和企业应用之间的集成。整个 JCA 架构包括资源适配器、系统合约和通用客户接口 [33]。

基于 J2EE 连接器架构（J2EE CA）的适配器通常用于集成 ESB 和企业系统。ESB 用智能转换和路由来补充其核心的异步消息收发中心，目的是确保消息的可靠传递。参与 ESB 的服务要么使用 Web 服务消息收发标准，要么使用 JMS。ESB 把 JCA 作为适配器的一种连接方式，连接一些支持 JCA 标准的应用，使之可以被其他的应用调用。

4.3.2 连接主机交易中间件

ESB 能够连接大型的主机交易中间件[34]，包括 CICS、TUXEDO 等业界主流产品。这些不同的产品。

其连接方式主要采用适配器（Adapter）技术，在第 2 章中叙述到的应用接口服务这方面的原理上，可以开发出针对上述交易中间件的接口和适配器。

针对某些应用，如 CICS，也可以采用 ESB 提供出来的 API，直接写代码，来调用 CICS Transaction Gateway 的服务，然后将结果通过 ESB 传送到调用他的服务请求者中去。这样做的好处是效率最高，但是也有部分安全性的问题。一般情况下，选择适配器作为连接方式，实施方便，安全性也高。

4.3.3 应用程序连接

主流的 ESB 产品提供强大的连接性，既提供各种与现有商业应用连接的 Adapter，可以将企业内部各种应用系统进行无缝连接，如 SAP，Notes，Sibel，SWIFT，People Soft，I2 等，支持各种标准数据格式或应用的接口，如 XML，JDBC，对于这些应用可以不必开发新的接口，减少开发的工作量；同时提供应用程序接口，以开发客户化的连接件。

作为 ESB 的组件之一，各个应用程序适配器（包括 JCA）将 SAP、Notes、Sibel、SWIFT、PeopleSoft、I2 等打包的应用程序的数据转换为 ESB 可以识别和路由的 Business Object，ESB 将 BO 转为其他服务请求者处理，并获得响应 BO。相反，经过处理后的 BO，也是由适配器转换成应用程序自己的数据格式，完成业务处理[35]。

4.4 系统的总体架构

以企业服务总线为核心采用面向服务的体系架构，可以解决企业内部多个部门和企业外部的不同平台的多个系统进行集成遇到的问题，使系统做到随需应变具有高度的可适应性。

整个系统的逻辑架构如下图所示。

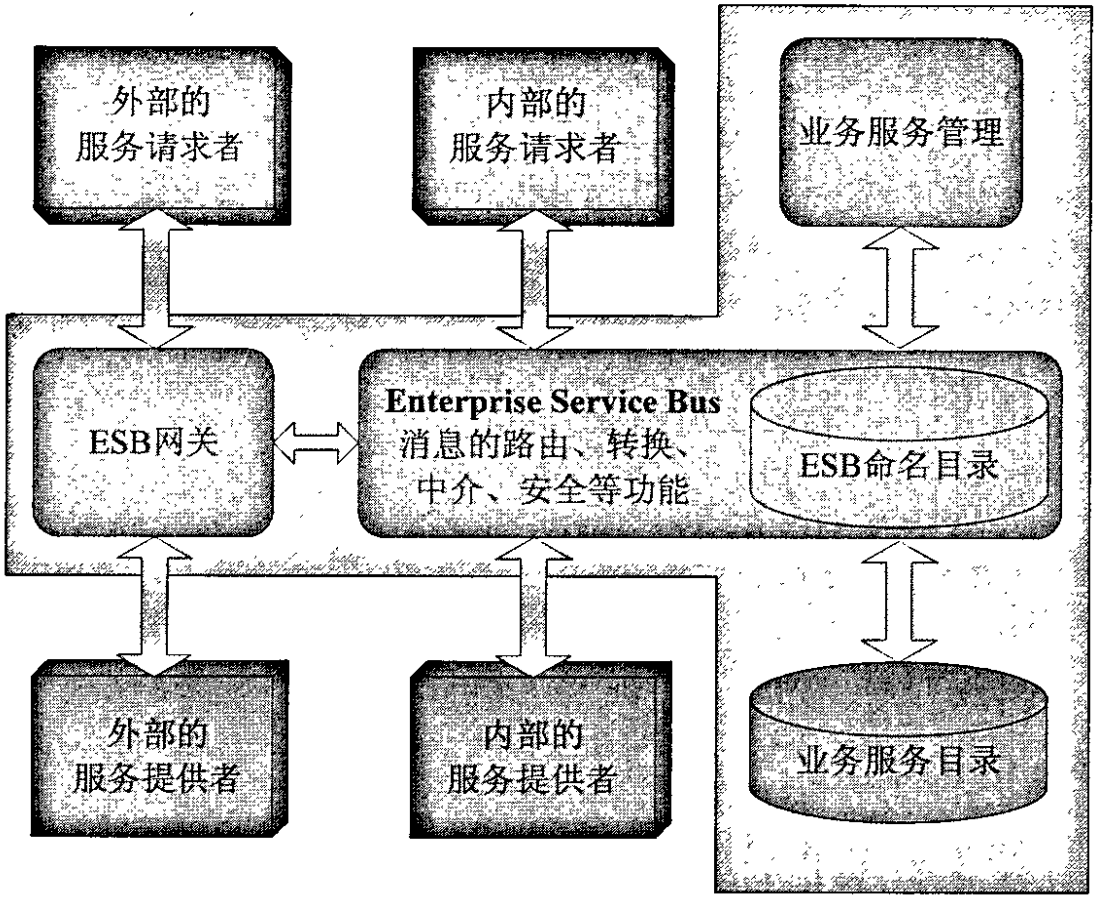


图 4-2 系统架构图

Figure 4-2 System architecture

该架构面向消息，不同的应用系统之间通过 ESB 发送和接收 XML 格式的报文（消息）进行数据交换，以实现与不同应用系统得数据交换和信息共享。

外部/内部的服务提供者/服务请求者：将企业内部和外部的应用封装成为服务，这些服务通过 ESB 进行信息的交换。

ESB：主要功能是消息的传输、路由、转换以及消息的安全性等。

业务服务目录（Business Service Directory）：ESB 需要服务的路由目录（service routing directory）对所有的服务请求进行路由。然而，SOA 可能还有单独的业务服务目录（business service directory），其最基本的形式可能是设计时服务目录，用于在组织的整个开发活动中实现服务的重用。Web 服务远景在业务服务目录和服务路由目录的角色中都放置了一个 UDDI 目录，因而使得可以动态发现和调用服务。这样的目录可以视为 ESB 的一部分，但是在这样的解决方案广泛应用之前，业务服务目录可能与 ESB 是分离的。

业务服务管理（Business Service Choreographer）：它通过若干业务服务来组合业务流程；因此，它将通过 ESB 调用服务，然后再次通过 ESB 将业务流程公开为客户端可用的其他服务。然而，Business Service Choreographer 在编排业务流程和服务中所扮演的角色确定了这种业务工作流技术是一种与基础架构技术 ESB 分离的技术。

ESB 网关（ESB Gateway ）：ESB Gateway 组件的作用是使两个或多个组织的服务在受控且安全的方式下对彼此可用。这有助于查看这些连接到 ESB 的组件，但它们并不是 ESB 的一部分。虽然有一些网关技术可以提供适合于实现 ESB Gateway 组件和 ESB 的功能，但是 ESB Gateway 组件的用途是将其与 ESB 分离。事实上，这种用途可能需要附加的功能（如合作伙伴关系管理），这些功能不是 ESB 的一部分，并且不一定受到 ESB 技术的支持。

4.5 系统的逻辑运行模型

在企业环境中出于安全性的考虑，对企业内部应用访问 Internet 时有所限制的，更不可能直接通过 Internet 访问到企业内部的应用。系统的逻辑运行模型如下图：

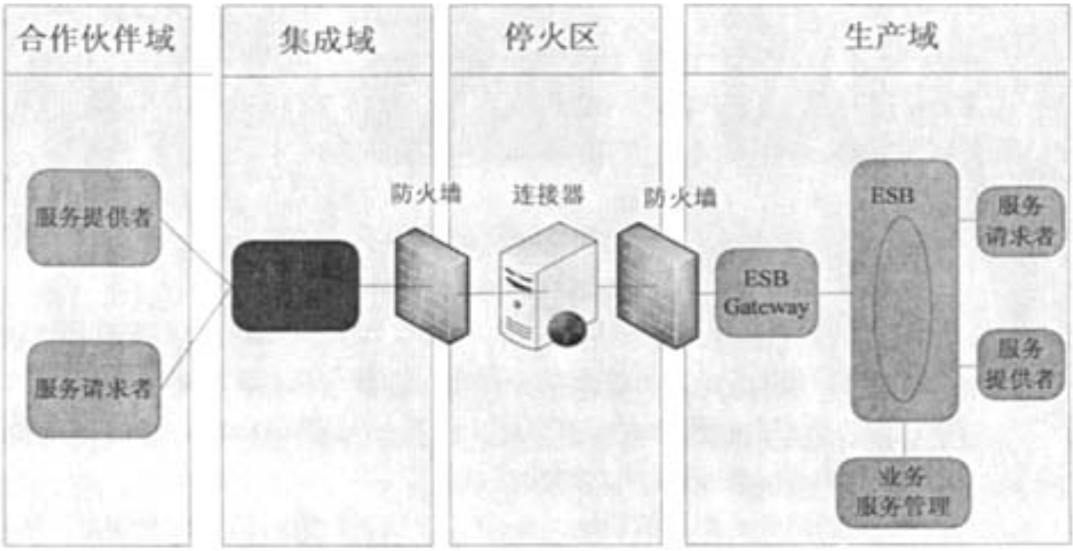


图 4-3 系统逻辑运行图

Figure 4-3 Logical operational model

利用交换机和防火墙（Firewall）将系统划分为不同的安全域，包括合作伙伴域、和集成域、停火区（DMZ）和生产域。

外部防火墙抵挡外部网络的攻击，并管理所有内部网络对 DMZ 的访问。内部防火墙管理 DMZ 对于内部网络的访问。内部防火墙是内部网络的第三道安全防线（前面有了外部防火墙和堡垒主机），当外部防火墙失效的时候，它还可以起到保护内部网络的功能。而局域网内部，对于 Internet 的访问由内部防火墙和位于 DMZ 的堡垒主机控制。在这样的结构里，一个黑客必须通过三个独立的区域（外部防火墙、内部防火墙和堡垒主机）才能够到达局域网。攻击难度大大加强，相应内部网络的安全性也就大大加强。

合作伙伴域：主要包括各个合作伙伴的应用系统。

集成域：主要包括与外部系统连接的网络基础设备。

停火区：通过防火墙在内部网络和外部网络之间构造了一个安全地带。它提供了一个区域放置公共服务器，从而又能有效地避免一些互联应用需要公开，而与内部安全策略相矛盾的情况发生。连接器是外部的应用和内部应用进行通信的桥梁。

生产域：主要包括 ESB，ESB 网关（ESB Gateway），以及企业内部各个应用系统。ESB Gateway 组件的作用是使两个或多个组织的服务在受控且安全的方式下对彼此可用。这有助于查看这些连接到 ESB 的组件，但它们并不是 ESB 的一部分。

4.6 本章小结

本章提出了一个基于 ESB 的应用集成模型，该模型以企业服务总线为核心采用面向服务的体系架构，将企业内部多个部门和企业外部的不同平台的多个系统封装成服务，不同的服务之间通过 ESB 发送和接收 XML 格式的报文（消息）进行数据交换，以实现与不同应用系统得数据交换和信息共享，这样可以解决进行集成遇到的问题，使系统做到随需应变具有高度的可适应性。

第5章 ESB 在银行系统中的应用

5.1 需求分析

5.1.1 功能性需求

国库信息处理系统（Treasury Information Process System，TIPS）是建立在财政、税务与国库各自独立的信息系统之上的信息交换和处理系统。系统的建立能够实现国家预算收支各职能部门之间信息的联网交换，加大预算执行的速度和透明度，便于各部门及时了解预算执行情况，从而加速预算资金的周转速度，提高财政资金的使用效率，防止预算执行中的腐败现象，保证国家预算的正常执行。

国库信息处理系统将实现国库信息的集中，从而在根本上改变目前国库部门在一定程度上存在的“信息孤岛”的局面，使国库内部所蕴涵的大量收支信息得以及时和充分利用，这将为下一步整合国库系统的信息资源，从而为财政、国库和税务部门进行预测和综合分析提供基础。

国库信息处理系统参与机构包括财政、国税、地税、海关、国库和各商业银行等部门。各机构业务关系如下图所示：

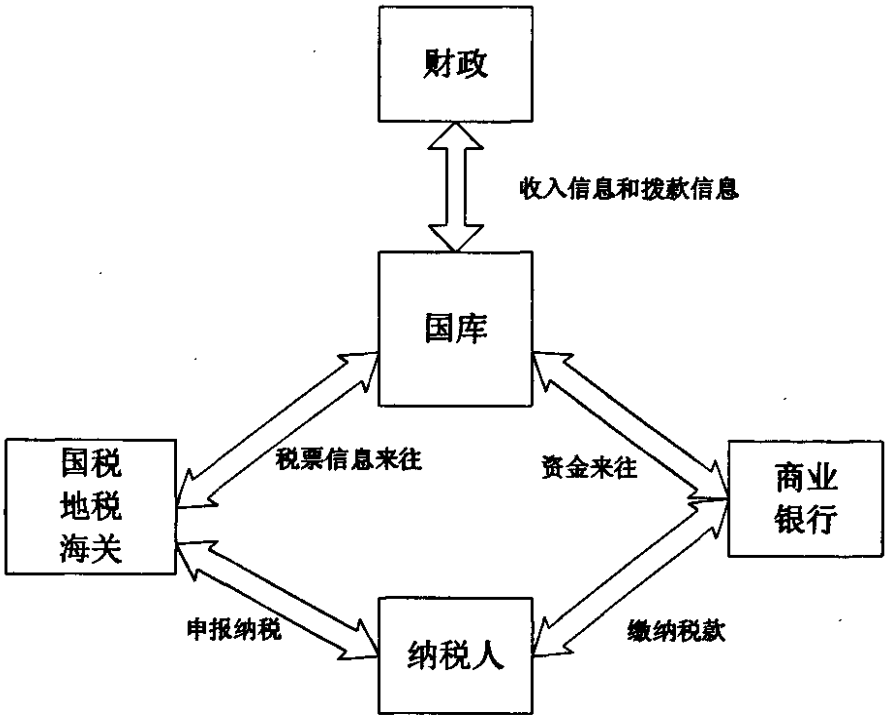


图 5-1 各机构业务关系图

Figure 5-1 System Context Diagram

国库信息处理系统的建设遵循“建立中心、数据和信息处理集中、业务操作

分散”原则，逐步实现税收收入缴库、退库、免抵调、更正等业务的电子网络一体化处理。在尽量不改变参与机构现有业务处理模式的前提下，各参与机构可根据自身的联网情况采用不同层级接入国家金库信息处理系统。国库信息数据高度集中，征收机关和商业银行可以进行不同层次的数据集中。各参与机构连入国库信息处理系统接口标准全国统一。

5.1.2 非功能性需求

- 业务量

根据2004年和2005年业务增长情况统计，2004年全国缴税业务量2.4亿笔，2005年全国缴税业务量2.9亿笔，每年增长20%左右。

- 实时处理能力

目前一年的业务量为2.9亿笔，高峰交易量为1000笔/秒，考虑每年业务量以20%的速度增长，则到2008年的预计高峰交易量为2100笔/秒。

- 业务处理和系统响应时间

系统登录时间最长3秒；从报文或文件进入系统到接收回执时间不超过5秒；报文或文件传输不成功时，在3—5秒时间内通知发送者；因某种原因，报文或文件滞留在系统中时，应在30秒时间内向发送者发出提示信息。

- 安全性

系统应保证数据的安全可靠，系统中传输的数据应可考虑采用电子签名等技术进行安全处理。应能在业务规定的工作日期和时间稳定运行。包括基于主机系统、操作系统、数据库高可用特性，以及集群（Cluster）高可用软件。（例如网络设备、前置机、数据交换平台、防火墙、通讯线路冗余备份等）

- 灵活性及可扩展性

系统在灵活性上应能适应各地财政、税务、国库的业务需要及未来发展的需要；系统在可扩展性上应能适应多种接入模式，并应预留后续接入单位的接口。

系统部署到全国数据中心以后，全国的交易处理和交易数据存储都将集中，联网中心集中承担交易的校验、存储并转发，其处理压力极大，系统中将存在大量的密集型并发交易，因此系统必须解决“三高”问题：高性能性、高可用性和高可扩展性。

5.1.3 集成产品的选择

自20世纪90年代诞生以来，中间件技术发展迅速，特别是近三年来，从全球到中国，中间件产业化速度发展很快。IBM 是中间件市场的领跑者，IBM WebSphere 可以提供基于 SOA 架构的，以企业服务总线（ESB）为基础的，以

WebSphere 应用服务器（WAS）为引擎的，完美的流程整合解决方案。IBM WBI 产品系列在国内外多个 SOA 项目中的成功实施，故选择 IBM 的 WBI 产品作为银行新业务的开发平台，这些产品构成了 IBM SOA 的基础架构，提供了 ESB 的基本功能，如服务路由、消息转换、中介、传输协议、消息传递模式、服务集成方式等，以及 ESB 的非功能性的支持，如安全性、事物、性能、可靠性、服务的监控和管理等。

在国库信息处理系统中主要用到两个产品：

（1）IBM WebSphere MQ，提供跨平台的可靠消息联接，供可靠的传输通道。可保证信息的不丢失、不重复并且做到“传一次且仅传一次的效果”[36]。消息、队列、队列管理器和通道是 MQ 中最重要的概念和对象[37]。

- 消息：在 MQ 中，把应用程序交由 MQ 传输的数据统一定义为消息，可以定义消息的内容，并对消息进行广义的理解，比如：用户的各种类型的数据文件，某个应用程序向其他应用程序发出的处理请求等都可以作为消息[38]。
- 队列：队列是消息的安全存放地，队列存放消息直到它被应用程序处理。
- 队列管理器：队列管理器是 MQ 系统中最上层的一个概念，由它为我们提供队列的消息服务。具体功能有：负责为应用系统提供队列的管理、消息的管理、应用程序使用 MQ 的接口以及和其他系统队列的管理器的通讯等。
- 通道：通道是 MQ 系统中队列管理器之间传递消息的管道，它是建立在物理的网络连接之上的一个逻辑概念，也是 MQ 产品的精华;在 MQ 中，主要有三大类通道类型，即消息通道，MQI 通道和 Cluster 通道。消息通道是用在 MQ 的服务器和服务器之间传输消息的，需要强调的是，该通道是单向的，它分为发送（send ）、接收（receive ）、请求者（requestor）、服务者（server）等不同类型，供用户在不同情况下使用。MQI 通道是 MQ Client 和 MQ Server 之间通讯和传输消息用的，与消息通道不同，它的传输是双向的。群集（Cluster）通道是位于同一个 MQ 群集内部的队列管理器之间使用的。

（2）IBM WebSphere Business Integration Message Broker，Message Broker 是 WBI 家族的一个重要组成部分[38]，基于 WebSphere MQ 开发。Message Broker 采用 WebSphere MQ 提供的可靠消息服务（不丢失，不复传）在应用系统之间通过基于消息的异步方式集成各应用系统。针对不同系统所处理的消息格式各不相同的特点，MESSAGE BROKER 提供了专门的格式代码转换器（Formatter）在不同的消息格式之间按照预先定义好的转换规则进行自动的格式转换，然后将结果自动路由到目标应用系统。在消息转换的过程中 MESSAGE BROKER 能够识

别 XML，C 结构，JMS 等多种消息格式；对消息的各种操作包括消息的来源、消息的目标应用、所期望的消息格式等通过定义各种操作规则（Rules）进行。

使用 Message Broker，您可以实现：

系统结构由网状结构到星型结构的改变，大大简化 MQ 的配置和管理；

智能路由和格式转换，根据消息的内容进行灵活而高性能的决策，从而将消息发送到相关的目的地；

由各种消息处理节点（Message Processing Node）组成的消息流（Message Flow），可对消息进行各种处理操作，每一个消息处理节点示对消息的一种处理，节点 与节点相连，便组成了一个消息流；在消息从数据中心经过时便可以被进行相应地计算，从而发往目的应用系统；

与 Database 紧密集成，提供了与 DB 操作相关的各种节点，如 INSERT，UPDATE，DELETE 等节点，用户可以之间通过 ESQL 进行和数据库的操作，如把数据存入数据库，从数据库中取数据等；

提供各种消息解析和处理服务，可以之间识别 XML，C 和 COBOL 中的结构以及用户通过 MRM 工具定制的符合自己需求的消息类型；可通过消息字典定义消息格式——消息字典可以是随产品提供的、与 Message Broker Version 1.x 相兼容的字典，也可以是由厂家授权的第三方字典。例如：它能够实现自定制格式与 XML 格式之间的灵活转换；

动态分发和订阅功能，能实现基于主题和内容的发布和订阅功能，该发布/预订服务系统与 MQ 基本的发布/预订功能相兼容，并包括如下的显著增强功能：

- 消息的主题和内容发送给感兴趣的订户
- 基于多级主题名进行授权
- 支持更灵活的发布/预订代理器的拓扑结构
- 定义发布/预订代理器拓扑结构的图形工具

友好的图形界面，强大的控制中心，在 GUI 下可以设计对消息的各种处理和路由，利用方便的图形工具使定义如何处理业务事件和关键数据变得容易，处理函数序列动态地处理和发送消息，并使它们与从公用数据库、数据仓库实时得到的信息数据相结合，对其进行审计和分析，并有效地将信息交付预订应用程序。

5.2 国库信息处理系统的总体设计

5.2.1 TIPS 的总体结构

以企业服务总线为核心采用面向服务的体系架构，可以解决 TIPS 与多个部门的不同平台的多个系统进行集成遇到的问题，使系统做到随需应变具有高度的

可适应性[39]。

整个系统的逻辑架构如图所示。

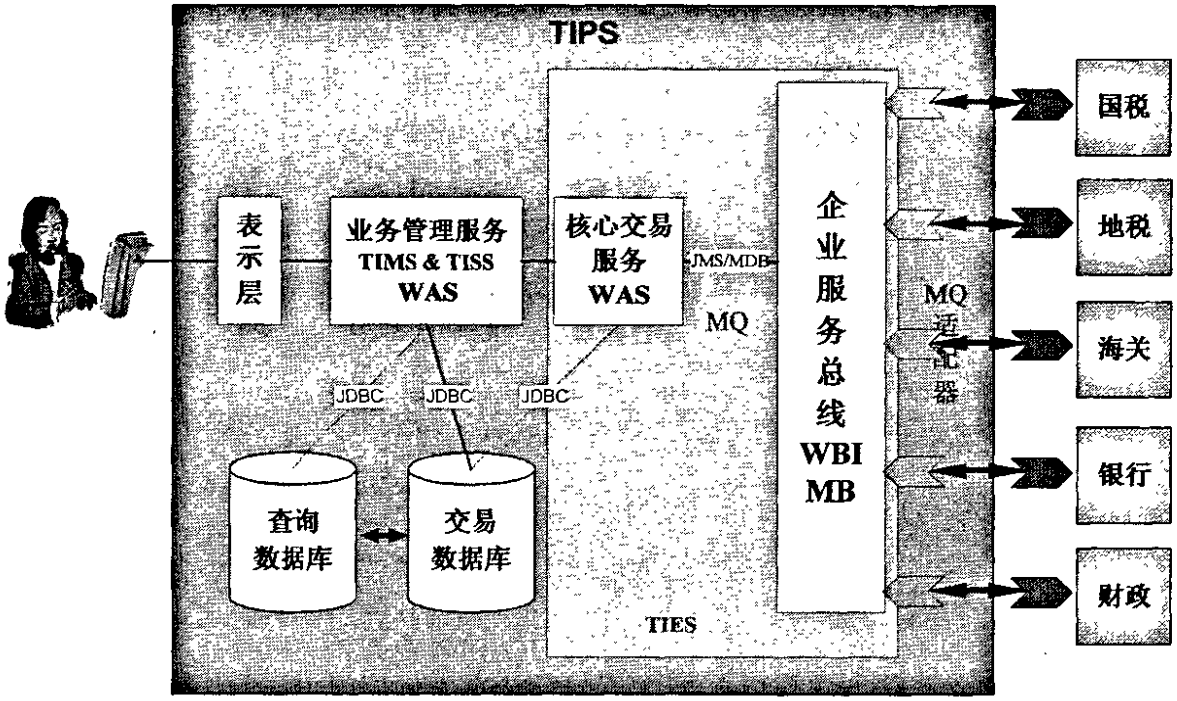


图 5-2 系统逻辑架构图

Figure 5-3 Architecture Overview Diagram

TIPS 采用 IBM WebSphere Application Server（简称 WAS）作为系统的应用服务器。TIPS 基于面向消息的架构，不同的应用系统之间通过 ESB 发送和接收报文（消息）进行数据交换。

其中，TIPS 由表示层、管理查询服务、核心交易服务、企业服务总线以及与相关系统集成的适配器 （Adapter）组成，各个部分功能如下。

（1）表示层支持用户通过 Browser 或定制的 Java 客户端访问系统。

（2）管理查询服务（TIMS&TISS）提供系统的配置管理和从业务角度的监控能力，同时提供数据分析、查询的处理能力。

（3）核心交易服务提供国库信息处理（如实时扣税）工作中各个环节的流程和业务完整性控制，并将相关信息保存到数据库中。

（4）数据库提供数据的存储，主要用来记录交易业务的日志和跟踪其状态信息。

（5）企业服务总线（ESB）主要提供相关系统的接入，消息（Message）的转换（Transformation）和路由（Routing）功能。通过 ESB 能够大大增强系统的适应性（Flexibility）。

（6）MQ 适配器提供联网机构和联网中心间稳定、可靠、安全的数据传输服务，能够保证数据的可靠传输——只传一次，保证传到。

本系统架构中各部分均采用 J2EE 三层（多层）架构进行构建，以确保系统

的开放性和扩展性。

5.2.2 TIPS 的运行模型

利用交换机和防火墙（Firewall）将系统划分为不同的安全域，包括接入环境（中国人民银行内联网）、停火区（DMZ）、生产域和集成域（Extranet）。利用工作负载均衡节点来完成工作负载在多个 Web 服务器间的均衡，以提升系统的吞吐率。在生产域中，将应用服务器分为显示应用服务器和交易应用服务器两类节点。通过企业服务总线与外部相关系统集成。

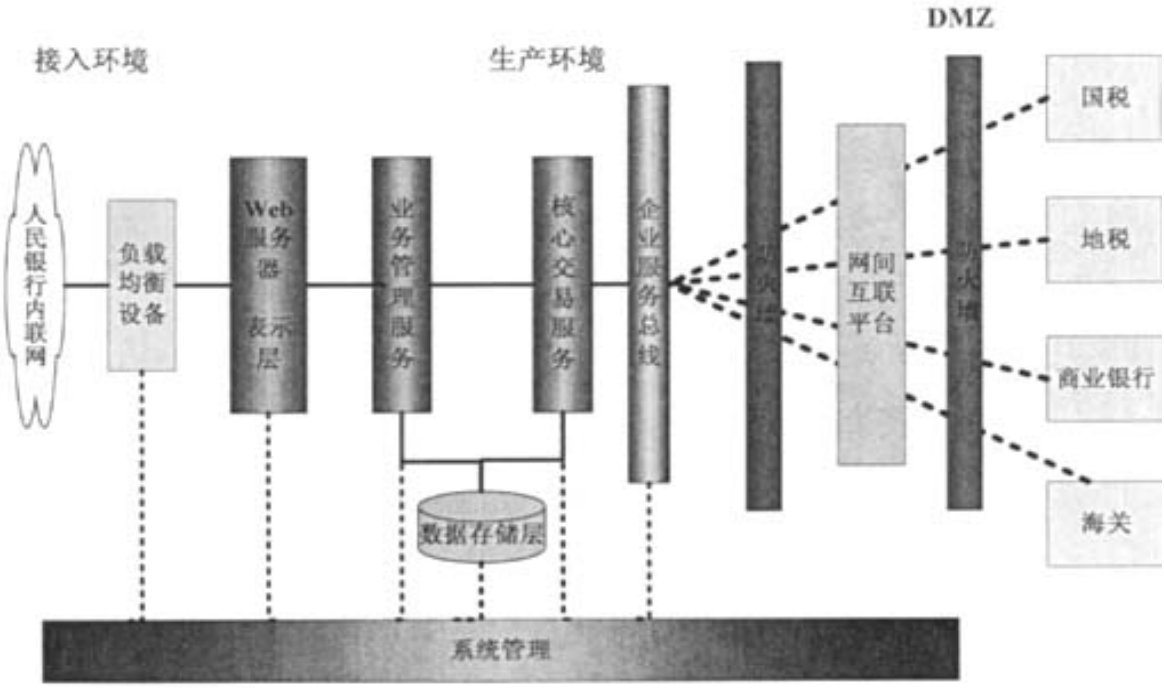


图 5-3 TIPS 系统逻辑运行模型

Figure 5-3 Logical Operational Model

5.2.3 TIPS 的报文设计

TIPS 采用了 XML 作为报文的统一描述语言，这是因为 XML 作为开放性数据描述语言，XML 广泛应用于应用程序间的数据交换，在数据交换中承担了一个重要角色。作为一种与平台无关的标准文本，XML 能够被所有程序语言读写，使用文档类型声明（DTD）[40]或 Schema 技术，XML 的解释程序就能对文件内容进行验证并处理[41]。一个可以对任何数据源进行抽象的、统一的数据访问的基础架构对于 SOA 的实现是很有价值的。XML 数据服务（XDS）提供了对多种数据资源的访问，数据建模功能，物理数据向逻辑数据的翻译和转换，以及对逻辑

数据的基于 XML 的访问。

（1）报文的基本结构，国库信息处理系统定义了一组基于 XML 格式的报文，以实现与不同应用系统得数据交换和信息共享[42]。

报文基本结构如下图所示：

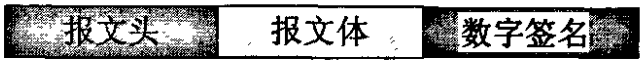


图 5-4 报文的基本结构图

Figure 5-4 The structure of the message

说明：总体上，报文全部内容封装在一个 XML 报文中，报文分为三大部分：报文头部分、报文体部分及数字签名部分。

- 报文头部分

报文头部分用于标识 XML 报文的基本属性，包括当前应用版本、发起节点代码、接收节点代码、应用名称、报文的编号、版本号、标识号、参考号及 TIPS 的工作日期。

- 报文体部分

报文体部分用于存放具体的交易报文，其内容由具体应用的交易种类决定。

- 数字签名部分

数字签名部分用于存放报文的数字签名信息，用于交易参与方的身份认证[43]。算法对报文开始（即从<?xml version=="1.0" encoding="GBK"?>行开始）至</CFX>之间的全部内容（不包括</CFX>之后的任何字符）进行签名，并以 XML 注释的形式存储于原 XML 报文的尾部。

注：标准 XML 报文首位字符不允许出现空格等字符，应以<?xml version 开始。具体格式示例如下：

```
<?xml version="1.0" encoding="GBK"?>
<CFX>
   <HEAD>报文头内容</HEAD>
   <MSG>报文体内容</MSG>
</CFX>
<!—数字签名 —>
```

TIPS 使用的报文按报文所含交易笔数分为两种：单笔报文和批量报文。

（2）单笔报文结构，单笔报文指报文所含内容为单笔交易信息，而对于某些特殊报文甚至可以没有报文体（比如连接测试报文）。该类报文的结构如下图所示：

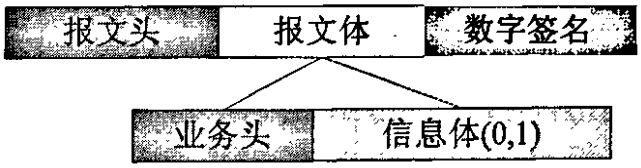


图 5-5 单笔报文结构图

Figure 5-5 The structure of the single message

该类报文主要包括征收机关实时扣税请求、TIPS 转发实时扣税请求、实时扣税回执、实时冲正请求、实时冲正回执、与银行税票明细对账回执、止付请求、止付应答、撤销请求、撤销应答、交易状态查询请求、交易状态查询应答、自由格式报文、连接测试请求、通用应答、通用确认应答、通用处理结果通知、银行申报请求、银行申报回执、登录请求、登录应答、退出请求、退出应答、三方协议验证请求、三方协议验证应答、包明细重发请求和对账信息下载请求等；

（3）批量报文结构，批量头必选，并且将明细信息根据信息内容的不同划分为不同的信息段。某些信息段下又包含各个明细，其他信息段内容则属于各个明细的公共部分，并具有特殊的作用（比如定时批量扣税中的转账信息段还具有辅助路由的功能）。该类报文的结构如下图所示：

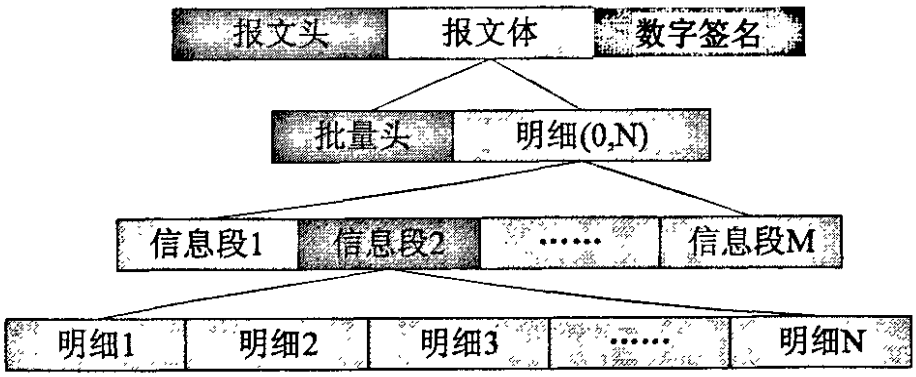


图 5-6 批量报文结构图

Figure 5-6 The structure of composite message

如征收机关批量扣税请求、TIPS 转发批量扣税请求、批量扣税回执、征收机关自缴核销请求、退库请求、更正请求、免抵调请求、与银行税票明细对账通知、与银行信息包核对通知、银行端缴款信息包核对通知、银行端缴款核对回执、与征收机关已扣税税票核对通知、与征收机关已入库税票核对通知、与征收机关退库核对通知、与征收机关更正核对通知、与征收机关免抵调核对通知、公共数据更新通知等。

5.2.4 TIPS 解决“三高”问题的策略

在本系统中，对性能的要求非常高。在全国集中的部署方式下，要求系统支持 5000 个以上系统的联结，具备 1600 笔/秒的业务处理能力。同时业务量每年

以 20%的速度递增，所以系统也应具有高可扩展性。

TIPS 是一个财政预算收入和支持的交换系统，将对社会公共服务如纳税提供基础平台的关键信息系统，除了要求系统具备足够的吞吐能力外，也要求系统具有高可用性，来支持业务的连续运行。

（1）高性能和高可扩展的解决策略，WBI Message Broker 是一个高性能的产品，在国内外许多大型的银行系统中有成功应用的案例。同时 WBIMessage Broker 提供了 Cluster 群集解决方案，通过横向扩展提高系统的吞吐能力。

经过多年的发展，WAS 本身具有相当高的处理能力。同时，WAS 可以通过多个 Server 的方式进行横向扩展。由于多个 Server 间相互独立，WAS 能够提供近似线形的扩展能力。

与此同时，从整体架构的角度来看，需要进行认真的设计，使得整个系统具有足够的扩展能力。

下图说明了在全国集中部署时实现系统扩展能力的架构：

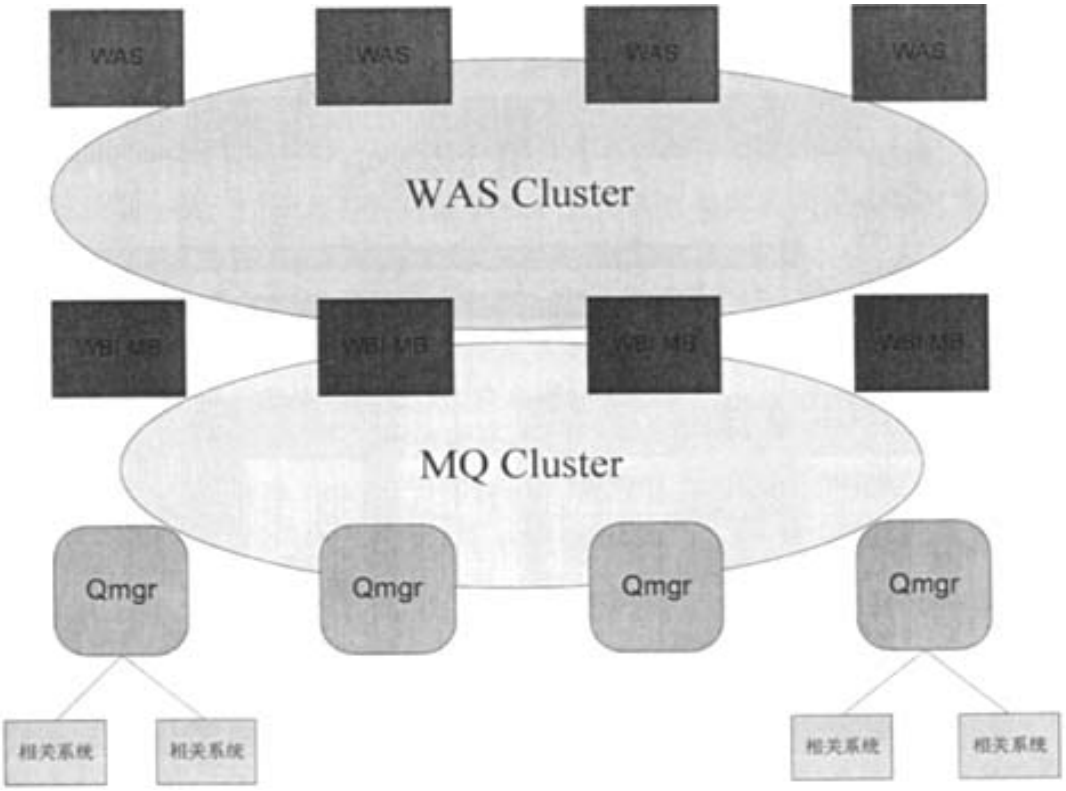


图 5-7 系统扩展能力的实现图

Figure 5-7 The scalability of TIPS

在 WBI MB 层次，通过同时部署多个 MB 提高系统的处理能力。同时，多个 Q Manager 与 MB 共同组成一个 Cluster，实现 Q Manager 之间，以及 MB 之间的全面负载均衡。

在 WAS 层次，同样通过同时部署多个 WAS 提高系统的处理能力。同时，

多个 WAS 与 MB 共同组成一个 Cluster，实现 多个 WAS 之间的全面负载均衡。

在此架构下，当需要进一步扩充系统的处理能力时，可以通过相应增加 Adapter、Q Manager、WBI MB、WAS 的数量，再结合增加系统的资源实现。

（2）高可用性的解决策略，高可用性的关键在于避免系统的单点故障（Single Point Of Failure），结合 IBM eServer 逻辑分区以及 HACMP 的功能，本系统支持高可用性（High Availability）的架构如下图所示：

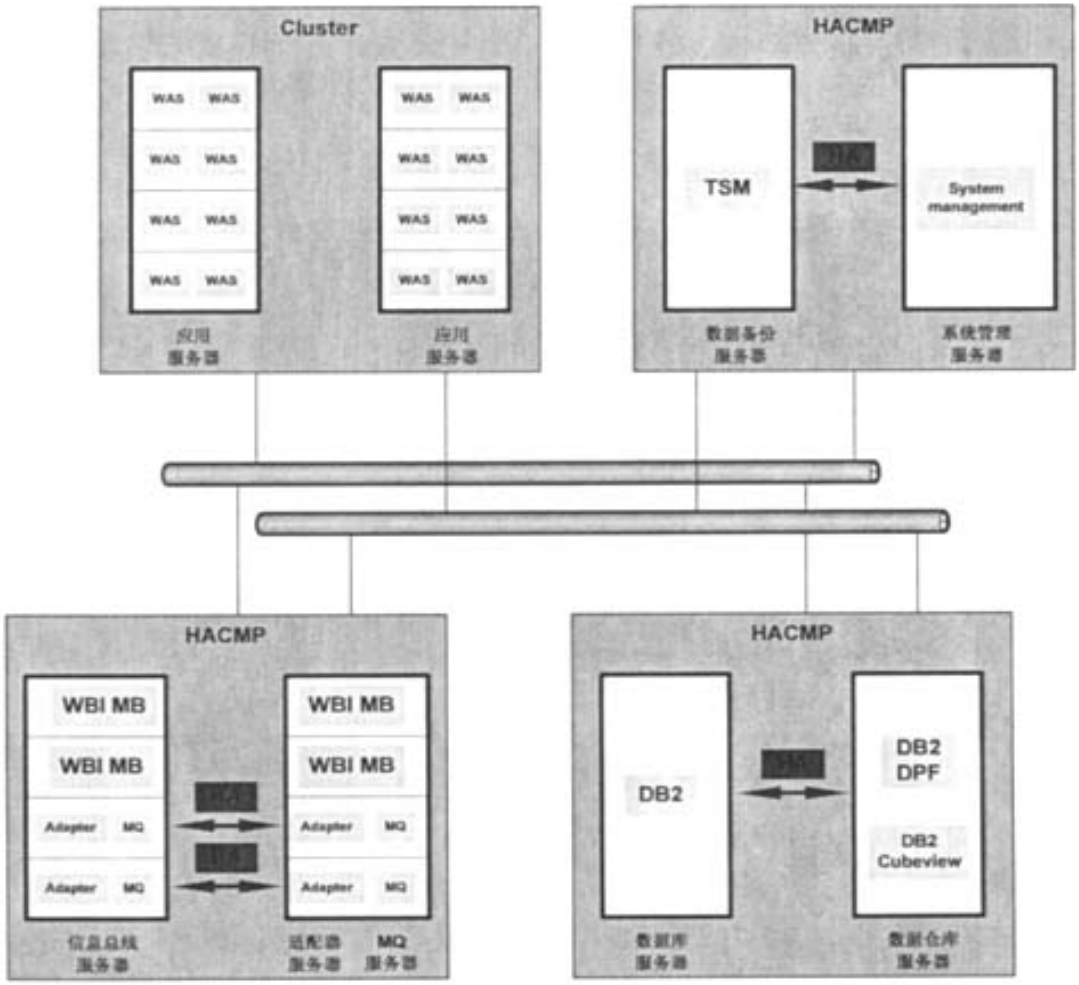


图 5-8 系统高可用性的实现图

Figure 5-8 The high availability of TIPS

其中 WAS 不使用 HACMP，而是通过不同层次的多机集群最大化地实现系统的高可用性。WAS 完全提供了独立于硬件系统的集群功能，实现了整个系统能够有效支持近似 7*24 小时不间断运行。当系统需要升级或安装补丁时，可对集群中的成员进行逐一更新，支持系统在升级期间不间断运行。

与 WAS 类似，多个 WBI MB 共同组成一个 Cluster 环境，在不依赖系统 HACMP 的情况下，实现系统的高可用性。

对 MQ 来说，为了支持 MQ Cluster，需要 Q Manager 的支持。该 Q Manager 用来消除一个单点故障。另外，Adapter 需要服务于固定的系统连接。他们都需要利用系统的 HACMP 实现高可用性。

对于 TSM（Tivoli Storage Manager） 和系统管理（System Management） 功能，需要两台独立的服务器，并利用系统的 HACMP 实现高可用性。

同样，对于数据库，也需要利用系统的 HACMP 实现高可用性。

考虑到系统的可管理性以及应用间的隔离，对运行 WAS 的两台服务器各划分 4 个分区（LPAR），在每个分区内可以同时运行多个 WAS 实例。

对信息总线服务器和 Adapter 服务器/MQ 服务器，同样各划分 4 个分区。分别运行两个 WBI MB 和 Adapter/MQ 的实例。其中，Adapter/MQ 分区两两进行 HA 配置。WBI MB 与 Adapter/MQ 分区之间的资源可以基于实际的运行情况，利用 pSeries 的动态 LPAR 功能进行动态调整。

数据库不进行系统级分区划分。其中，数据库本身支持大量用户请求的并行处理。

对于采用软件 Cluster 方式的部分，当其中的一个部分需要升级或安装补丁时，可以让其中的一个成员暂时停止服务，对其进行相应的操作后重新加入 Cluster 中。按照此方法可以对每个成员逐一进行处理。

当采用双机互备的方式时，如其中的一个部分需要升级或安装补丁，可以先在备份节点上进行相应的操作，再将系统从主节点切换到备份节点，之后对主节点进行操作，最后将系统从备份节点切换到主节点。

需要说明的是，以上两方面都需要对系统、中间件和应用进行仔细的规划，并且在进行充分的测试后，再在生产系统中使用。

此外，当系统需要变更接入机构时，仅需要修改 MB 中的相关设置，并逐一重新编译部署两个 MB 上流程，即可支持在不中断系统运行的条件下，完成变更接入机构。

综上所述，通过系统、中间件以及应用架构的良好设计，配合相应的管理流程，在开放系统中实现 24*7 的业务连续性，保持业务的不间断运行是可以实现的。

5.2.5 TIPS 的设计特点

松耦合性：TIPS 系统的一个典型特点就是实现与联网机构的有机集成。系统设计将应用程序定义为不同组件（或称为服务），通过这些服务之间定义良好

的接口和契约联系起来。接口是采用中立的方式进行定义的，它独立于实现服务的硬件平台、操作系统和编程语言[44]。这使得构建在各种这样的系统中的服务可以以松耦合的方式集成，并采用一种统一和通用的方法进行交互。基于上述思想，将横跨多个部门的一个业务流程（Process）分解为多个相对独立的活动（Activity），再通过一个集中控制的服务进行业务完整性控制。一旦某个环节出现异常或超时，则进行相应的冲正处理。这也要求与 TIPS 相联的系统提供对冲正处理的支持。

适应性：由于需要集成的系统相当多并且复杂，系统设计必须能够方便地适应当前相关系统的不同情况以及未来变化，包括一定范围内支撑技术、产品版本以及业务需求等方面的变化，同时也能通过描述的方式适应集成环境的改变。本系统应尽可能减少对现有系统的改变[45]。遵循关注分离的原则，TIPS 通过将显示服务、交易和集成服务、企业服务总线、数据存储服务和数据仓库/分析服务等相分离。同时服务之间通过标准的接口进行集成。如果某一个服务需要修改时，其他服务可以保持不变。同时，利用企业服务总线，可以将与本系统集成的其他系统的改变 （如系统 IP、报文格式等） 进行屏蔽，使得系统的其他部分保持不变。

成熟性：使用的产品都经过了市场考验，并且在全球范围内有广泛的用户。尽量避免采用一些小的厂商开发的、或者自己开发的中间件产品。

先进性：设计方案中采用市场领先并且成熟的技术，使项目居于国内同业领先的地位。

5.3 ESB 的总体设计

5.3.1 ESB 的总体结构

方案中我们采用 IBM WBI Message Broker 和 MQ 作为系统的信息总线。它在本系统中起到以下方面的作用：

（1）所有相关单位的接入服务，支持多种接入方式，并且进行各种接入协议之间的转换。

（2）数据路由服务，根据数据体标识，把数据路由到正确的目的地，当接入单位增加、删除、修改时，灵活地维护系统的路由表。

与自行开发的或其他类似产品比较，它的优势在于：

（1）具有优异的处理性能，该产品在业界同类产品中的性能是无可比拟的，它内部用于数据处理的消息流是以多线程方式工作的，同一个消息流还可以分配到不同的执行组，从而提高整个系统的运行效率；

（2）具有交易完整性保证，该产品支持不同层次的交易完整性要求，例如：可以设定整个消息流为一个完整的交易，当某一环节发生错误时，整个消息流回滚，保证数据一致性，这一点在该系统中是非常重要的；

（3）具有高可靠性，该产品支持 HA，在本方案中，通过四层交换机（硬件）或 IBM 的 MQ Cluster，可以实现多个 Message Broker 同时工作；

（4）具有方案的可扩展性，该产品功能完善，随着今后系统新需求的出现，可以不断发挥其强大功能，如消息格式转换的功能。

ESB 基于消息中间件 MQ 与外联机构（征收机关或商业银行）进行信息交换；在 TIPS 系统内部，消息中间件 MQ 作为 ESB 和 WAS Server 之间的桥梁，将 ESB 和 WAS Server 有机集成起来。ESB 系统总体逻辑图如下所示：

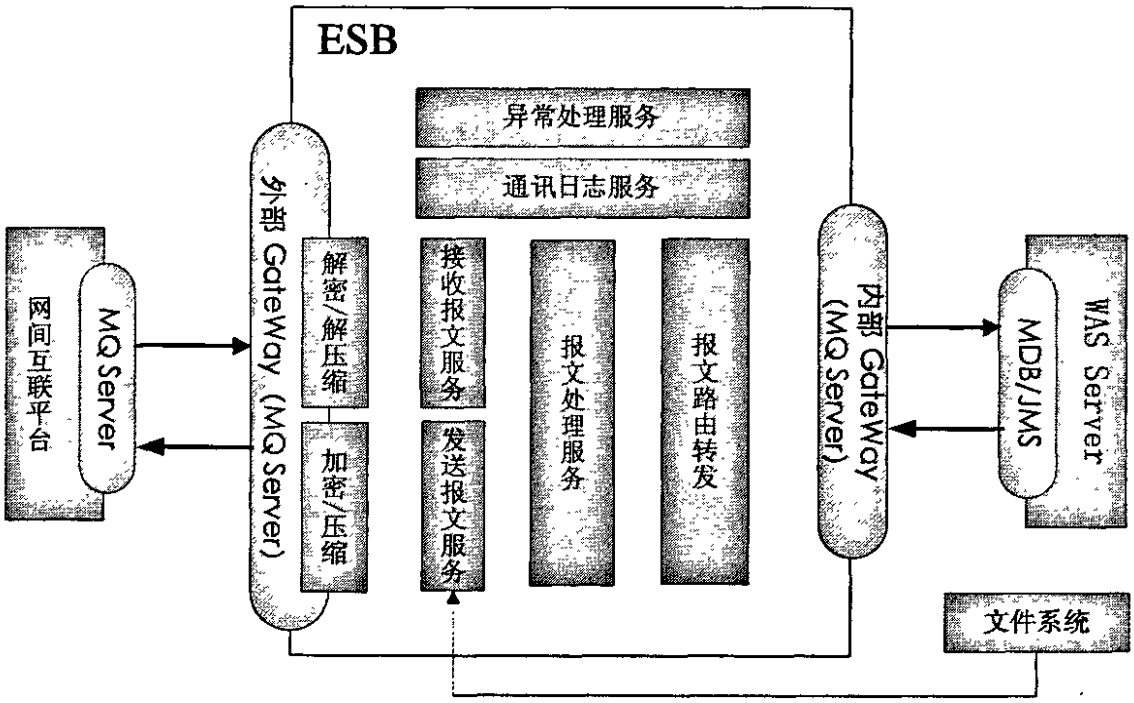


图 5-9 ESB 系统总体逻辑图

Figure 5-9 The architecture of ESB

ESB 作为 TIPS 的交易转接系统，由接收报文服务、发送报文服务、报文处理服务、报文路由转发服务、通讯日志服务、异常处理服务共计六部分组成。各部分的主要功能如下：

- 接收报文服务：本部件主要负责接收外联机构（征收机关或商业银行）发送给 TIPS 系统的各类交易报文，并实时交由报文处理服务进行处理；
- 发送报文服务：本部件主要负责发送报文给外联机构（征收机关或商业银行）；
- 报文处理服务：本部件主要负责报文的初步解析和必要的格式转换；在接收外联机构发起的报文时，本部件负责解析报文头信息，并转换成 WAS Server 容易处理的报文格式；在发送报文到外联机构时，本部件负

责转发报文，对于文件类报文，还负责报文由文件到报文格式的转换；报文处理完毕后，统一交由报文路由转发服务进行处理；

- 报文路由转发服务：将收妥的报文转发给 TIPS 系统的 WAS Server；将需要发送的报文路由到正确的接收者；
- 通讯日志服务：负责记录接收或发送的报文日志；
- 异常处理服务：在上述部件的任何一点出现异常时，本部件将自动接管报文的后续处理，在系统中主要起到跟踪作用。

另外，ESB 基于安全和通讯效率的考虑，通过 MQ Server 实现了加密和压缩功能。

5.3.2 ESB 与外部系统的连接

TIPS 与外联机构的业务系统之间的应用连接结构如下图所示：

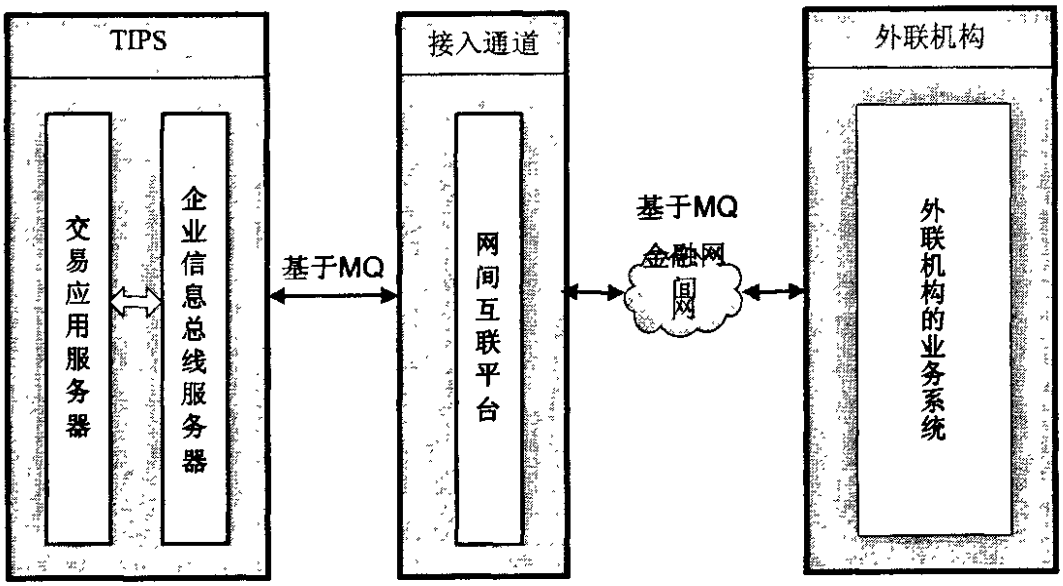


图 5-10 外部应用连接图

Figure 5-10 The connection between ESB and external applications

TIPS 与外联机构的业务系统的通信连接采用 MQ 中间件实现，外联机构的业务系统可选择的部署 MQ Server 或 MQ Client，如果选择部署 MQ Server，则选择对等模式进行通信连接，本模式可以保障性能需求；如果选择部署 MQ Client，则选择中心节点模式进行通信连接，本模式可以节约成本，但在高性能需求方面存在局限性。外联机构采用上述何种通信连接模式，建议遵循“以满足本入网机构性能需求为前提”这一准则。

在接口队列设计方面，为保障实时类交易的高可靠性和高实时响应性，做到实时类交易和非实时类交易相互影响降至最低，我们对实时类交易和非实时类交易分别提供一组通信接口队列。两组接口队列从业务角度上看，分别处理不同类

别的交易；但在通信流程上，没有本质差别。

因此，现以实时类交易为例，对中心节点模式和对等模式两种通信连接模式的通信流程分别加以说明：

- 中心节点模式（C-S 模式）

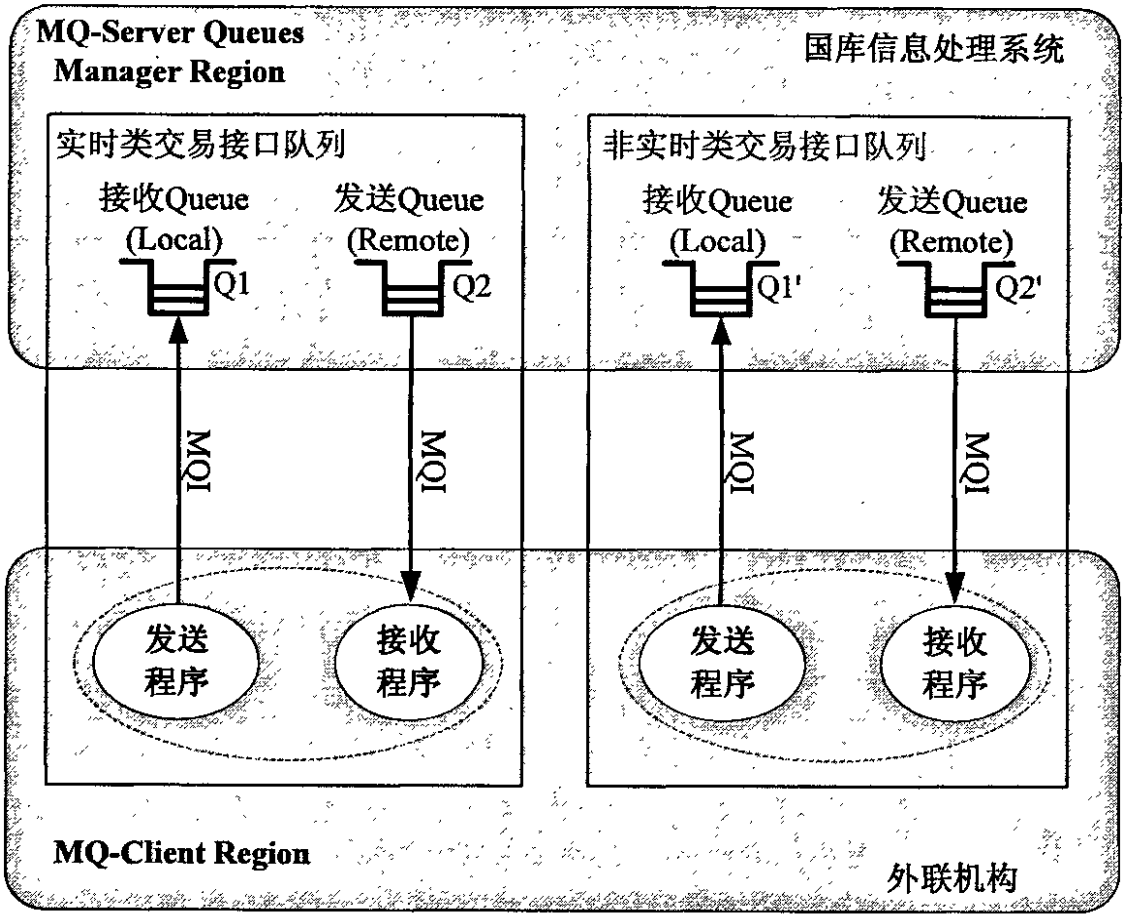


图 5-11 中心节点模式中间件数据传输图

Figure 5-11 The data transmission of C-S Model

TIPS 和外联机构的系统间的通信连接采用中间件 MQ Client TO MQ Server 的方式（即中心节点模式）。

（1）联网中心配置 MQ Server。通过 MQ manager 为每个接入机构分别创建接收队列（Q1）和发送队列（Q2）。接收队列（Q1）用于接收外联机构业务系统向 TIPS 输入的数据，发送队列用于发送 TIPS 向外联机构业务系统输出的数据。

（2）外联机构业务系统与 TIPS 建立通信连接后，外联机构业务系统调用 MQ Client API（即 MQI）将请求消息写入远程的 TIPS 分配的接收队列（Q1）中。

（3）TIPS 接收请求消息并处理完成后，将返回结果写到对应外联机构的发送队列（Q2）中，联网机构调用 MQ Client API（即 MQI）从对应的发送队列（Q2）中轮询接收处理结果。

- 对等模式（S To S 模式）

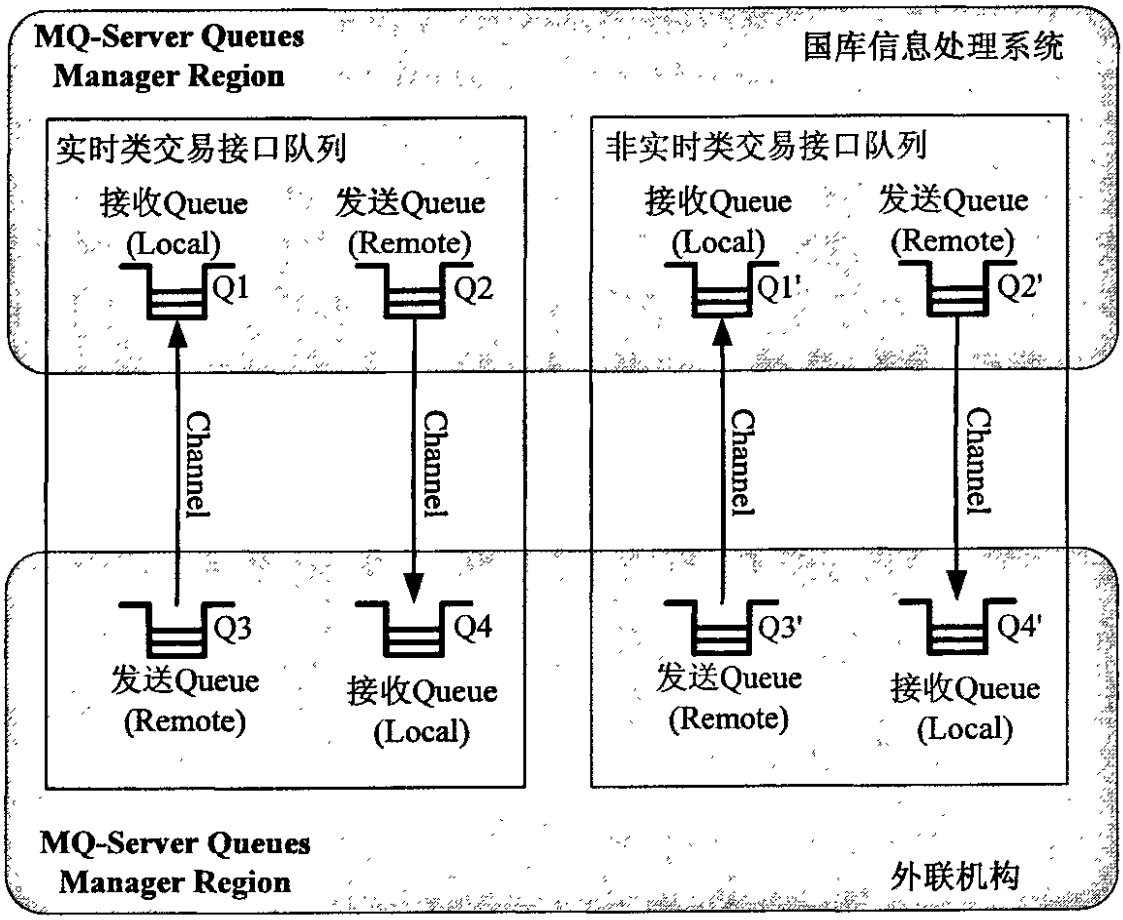


图 5-12 对等模式中间件数据传输图

Figure 5-12 The data transmission of S-S model

TIPS 和外联机构的业务系统间的通信连接采用中间件 MQ Server TO MQ Server 的方式（即对等模式）。

（1）TIPS 通过自身的 MQ manager 为接入的外联机构业务系统分别创建接收队列（Q1）和发送队列（Q2），接收队列用于接收外联机构业务系统向 TIPS 输入的数据，发送队列用于发送 TIPS 向外联机构业务系统输出的数据。

（2）外联机构业务系统通过自身的 MQ manager 创建自己的发送队列（Q3）和接收队列（Q4）。接收队列用于接收 TIPS 向外联机构业务系统输入的数据，发送队列用于发送外联机构业务系统向 TIPS 输出的数据。当外联机构业务系统与 TIPS 建立连接以后，会创建从外联机构业务系统到 TIPS 之间的通道（channel），通道两端队列的对应关系见上图。

（3）外联机构业务系统通过调用 MQ API 将请求消息写入本地 MQ Server 的发送队列(Q3)中，消息经通道传输至 TIPS 为该业务系统分配的接收队列(Q1)中，TIPS 处理完成后，将返回结果写到对应为该业务系统分配的发送队列（Q2）中，消息经通道传输至外联机构业务系统。

5.3.3 ESB 与内部系统的连接

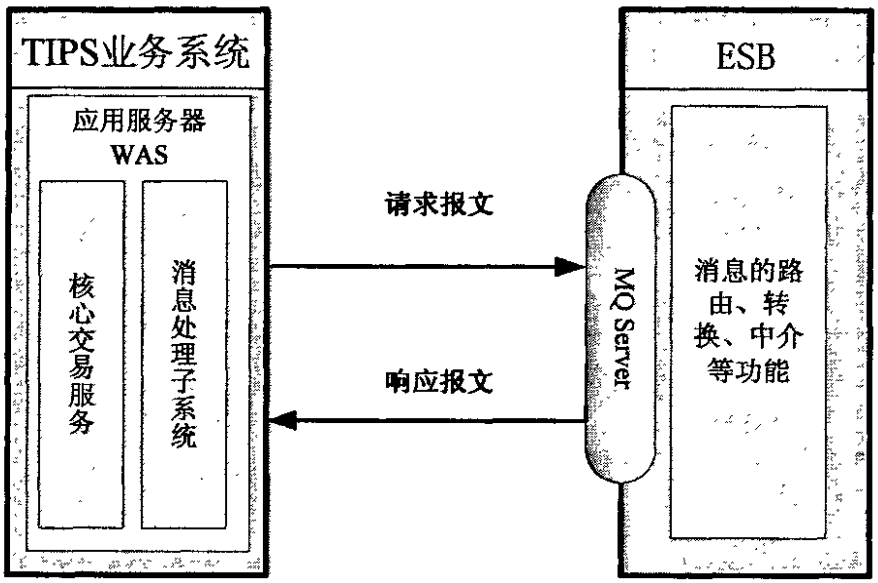


图 5-13 ESB 与内部系统链接图

Figure 5-13 The connection between ESB and internal applications

TIPS 业务系统运行在 WebSphere Application Server（WAS）应用服务器中，消息中间件 MQ 作为 ESB 和 WAS Server 之间的桥梁，将 ESB 和 WAS Server 有机集成起来。

WAS 与 MQ 可以很好地结合，它们是 WebSphere 产品家族的重要成员，都支持 J2EE 标准，两者可以通过 JMS 实现有机的结合，其中 MQ 可以作为 WAS 的 JMS provider，MQ 的消息可以直接通过 WAS 的 Message Driven Bean 驱动 WAS 的应用。另一方面，WAS 应用可以直接通过 JMS 访问 MQ 消息队列。由此实现 WAS 与 MQ 的高度集成，使应用的开发和维护更加方便。

TIPS 业务系统通过消息处理子系统与企业服务总线进行消息的交互。消息处理子系统主要负责监听 MQ 队列，对传入的消息进行处理（格式验证、验签名），并根据报文的类型调用 TIES 中的服务。同时负责将服务处理完的结果打包、签名、发送到 MQ 队列中。

5.4 ESB 接收报文模块

5.4.1 接收报文逻辑流程

ESB 接收报文服务主要负责接收外联机构（征收机关或商业银行）发送给 TIPS 系统的各类交易报文，并实时交由报文处理服务进行处理。ESB 接收报文的逻辑流程如下图所示：

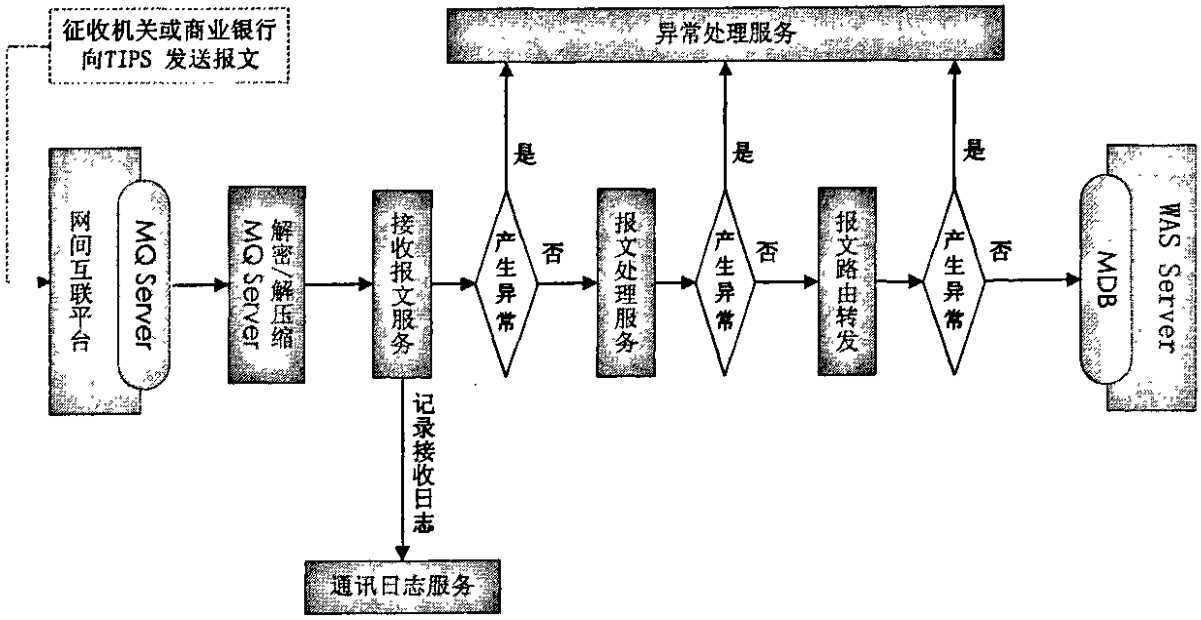


图 5-14 接收报文处理逻辑流程图

Figure 5-14 Flow chat of the message receiving model

为了提高报文在网络上的传输效率，各商业银行接入节点发送的报文须进行压缩处理，接收的报文要进行解压缩处理。TIPS 是人民银行的一个重要应用系统，TIPS 与外联结构的信息交换涉及到大量的保密信息，所以 TIPS 与外联结构的数据交换需要安全的通信机制。

对于 ESB 接收报文的处理逻辑，主要分为以下六个步骤：

（1）征收机关或商业银行将发送给 TIPS 系统的压缩和加密后报文首先发送到网间互联平台。

（2）网间互联系统通过 MQ Server 将压缩和加密后的报文发送到 TIPS 系统。

（3）ESB 通过 MQ Server 对接收到的报文进行解压缩和合法性的验证。

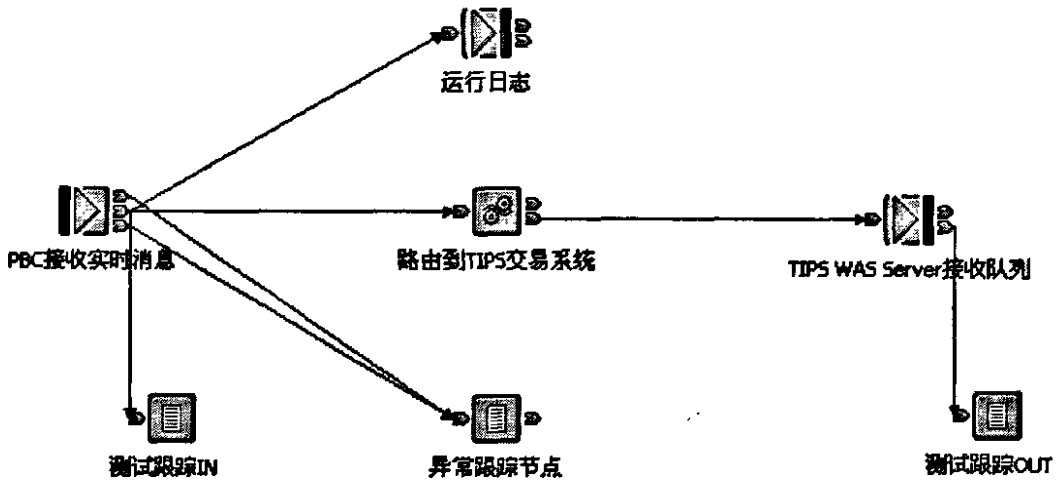
（4）调用接收报文服务对接收到的报文进行记录，并写入到接收日志。

（5）调用报文处理服务对报文格式转换预处理，以便于 WAS Server 处理，并通过报文路由转发服务，转发到 WAS Server。

（6）在上述处理任意环节出现异常，报文都将交由异常处理服务进行后续处理。

5.4.2 接收报文的消息流

- 接收报文的消息图



5-15 接收外联机构消息的消息流图

Figure 5-15 Flow chart of TIPS receive external message

对于外联机构发送给 TIPS 系统的实时报文，由实时报文接收队列（PBC.EXT.ONLINE.IN）负责统一接收，MB 负责将消息从队列 PBC.EXT.ONLINE.IN 转发到 TIPS WAS Server 接收队列（PBC.TIPS.ONLINE.OUT），交由 WAS 进行业务逻辑处理。

同时，MB 将接收到的实时交易报文记录到运行日志队列（PBC.TIPS.ONLINE.LOG）中；如果出现异常，将异常信息通过异常跟踪节点记录到日志文件。通信日志的记录，为保证报文路由转发的高效性，采用异步记载日志的方式，通信日志将通过单独的消息流写到日志文件中。

- 消息计算节点的 ESQL 代码

```
BROKER SCHEMA TIPS
CREATE COMPUTE MODULE ToTips_OnLineMsgFlow_Compute
    CREATE FUNCTION Main（） RETURNS BOOLEAN
    BEGIN
        -- 拷贝 MQ 自带的消息头信息
        CALL CopyMessageHeaders（）;
        -- 创建 JMS 消息头
        CREATE FIELD OutputRoot.MQRFH2;
        -- 拷贝 XML 消息体
        SET OutputRoot."XML" = InputRoot."XML";
        IF CheckMsgHead（OutputRoot） THEN
            -- 增加 JMS 消息头
            CALL AddJMSHead（OutputRoot）;
            -- 将消息路由到目标队列（WAS 接收队列）
```

```
            SET OutputLocalEnvironment.Destination.MQ.DestinationData[1]
             .queueName = OUT_ESB_ONLINE_QUEUE;
                RETURN TRUE;
            ELSE
                THROW USER EXCEPTION CATALOG 'HeadError'
            MESSAGE 2900 VALUES('报文头格式错误，或关键要素存在空值');
                RETURN FALSE;
            END IF;
        END;
        CREATE PROCEDURE CopyMessageHeaders() BEGIN
            DECLARE I INTEGER 1;
            DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
            WHILE I < J DO
                SET OutputRoot.*[I] = InputRoot.*[I];
                SET I = I + 1;
            END WHILE;
        END;
        CREATE PROCEDURE CopyEntireMessage() BEGIN
            SET OutputRoot = InputRoot;
        END;
    END MODULE;
```

5.5 ESB 发送报文模块

5.5.1 发送报文逻辑流程

发送报文服务主要负责将 TIPS 的报文发送给外联机构（征收机关或商业银行）ESB 发送报文的逻辑流程如下图所示：

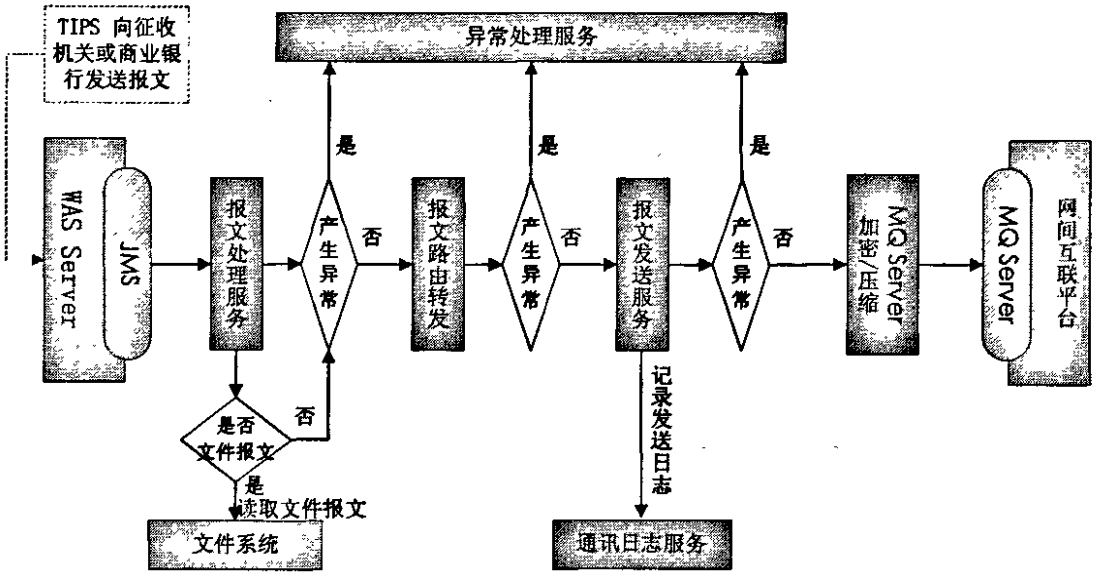


图 5-16 发送报文处理逻辑流程图

Figure 5-16 Flow chat of the message sending model

TIPS 系统发送报文给征收机关或商业银行时，对于 ESB 发送报文的处理逻辑，主要分为以下六个步骤：

（1）TIPS 业务系统通过基于 JMS 的消息驱动 Bean（Message Drive Bean）将报文发送到 ESB。

（2）ESB 通过报文处理服务对报文进行解析，并对于文件类报文，从文件系统读取文件，转换成报文格式。与非文件类报文以统一的方式，交由报文路由转发服务进行路由交由报文路由转发服务进行路由，由报文发送服务负责发送给征收机关或商业银行。

（3）调用报文路由转发服务确定报文发送到的目的队列。

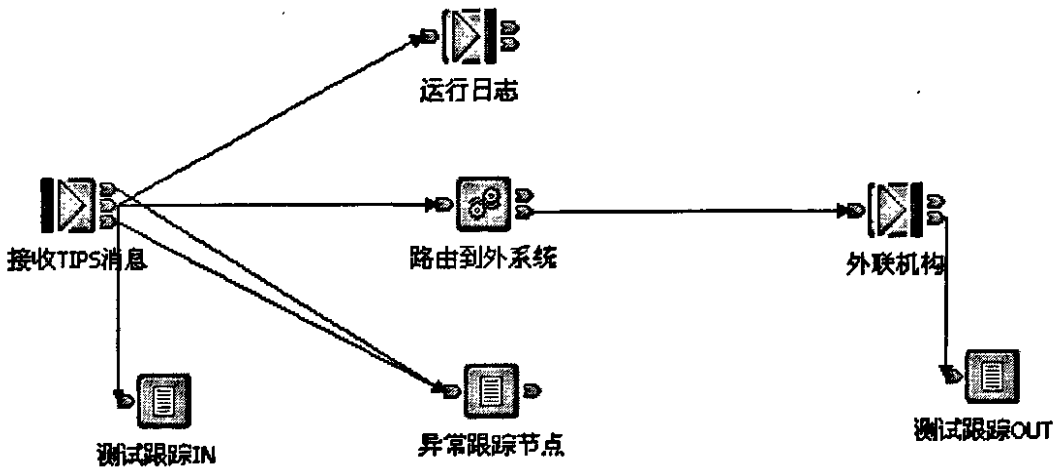
（4）由报文发送服务负责将报文发送给征收机关或商业银行，途经 MQ Server 时，进行加密和压缩。在报文通过发送报文服务时，记录发送日志。

（5）网间互联系统通过 MQ Server 将压缩和加密后的报文发送到外联机构的系统。

（6）在上述处理任意环节出现异常，报文都将交由异常处理服务进行后续处理。

5.5.2 发送报文的消息流

- 发送报文的消息流图



5-17 发送到外联机构消息的消息流图

Figure 5-15 Flow chart of TIPS send message

对于 TIPS 系统发送给外联机构的实时报文，由实时报文转发接收队列（PBC.TIPS.ONLINE.IN）负责统一接收，MB 负责将消息从队列 PBC.TIPS.ONLINE.IN 取出，通过路由后，转发到外联机构接收队列（PBC.节点代码.ONLINE.OUT）。

同时，MB 将接收到的实时交易报文记录到运行日志队列（PBC.EXT.ONLINE.LOG）中；如果出现异常，将异常信息通过异常跟踪节点记录到日志文件。通信日志的记录，为保证报文路由转发的高效性，采用异步记载日志的方式，通信日志将通过单独的消息流写到日志文件中。

- 消息计算节点的 ESQL 代码

```
BROKER SCHEMA TIPS
CREATE COMPUTE MODULE ToExtSys_OnlineMsgFlow_Compute
    CREATE FUNCTION Main（） RETURNS BOOLEAN
    BEGIN
        CALL CopyEntireMessage（）;
        -- 删除 JMS 消息头
        SET OutputRoot.MQMD.CodedCharSetId
= OutputRoot.MQRFH2.（MQRFH2.Field）CodedCharSetId;
        DELETE FIELD OutputRoot.MQRFH2;
        -- 设置消息有效期
        --CALL setMsgTimeout（OutputRoot）;

        --DECLARE Router CHARACTER;
        DECLARE Router CHARACTER InputRoot.XML.CFX.HEAD.DES;
        SET
```

```
OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName
= GetOutputOnlineQueue（Router）;
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders（） BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER CARDINALITY（InputRoot.*[]）;
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage（） BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

5.6 ESB 加密和压缩报文模块

5.6.1 报文的加密

TIPS 是人民银行的一个重要应用系统，TIPS 与外联结构的信息交换涉及到大量的保密信息，所以 TIPS 与外联结构的数据交换需要安全的通信机制。

近年来，随着 Internet 的普及，网络安全越来越受到人们的普遍重视，安全套接字（Security Socket Layer—SSL）协议已经逐渐成为一种安全标准被人们广泛采用。SSL 协议使用密钥加密和公钥加密、数字签名以及数字证书的概念在各方之间建立一个安全连接，并且用这些概念使得在不安全网络上传输安全数据更容易。

WebSphere MQ 5.3 支持 SSL，MQ 中的 SSL 体现为通道的属性。这里的通道即可以是 MQ Server 之间的消息通道，也可以是 MQ Client 与 MQ Server 之间的调用界面通道。

无论是 Server 端还是 Client 端的配置，都需要经历准备证书、配置证书（添加和指定证书）、配置通道三步。

5.6.2 报文的压缩和解压缩

为了提高报文在网络上的传输效率，各商业银行接入节点发送的报文须进行压缩处理，接收的报文要进行解压缩处理。

用户出口（User Exit）是 WebSphere MQ 中的高级功能，它实际上是一个函数接口，它可以帮用户将自己编写的程序嵌入 WebSphere MQ 运行环境中，并在适当的时候被自动调用。

利用出口机制，客户可将自己的加密、压缩等计算处理模块平滑地嵌入到 MQ 的消息传输处理流程中，从而使这些计算处理的算法部分与进行队列读写的应用程序相分离，这样根据灵活性和扩展性的需要，可以经常改变出口的计算处理算法，而不需修改应用程序。

一般而言，出口程序是成对使用的，譬如在发送端使用了压缩出口程序，则在接收端必须使用解压缩出口程序。这些出口程序的作用分别在于：

- 安全出口：主要用于两个 MQSeries 系统之间通道启动时的双方的身份认证。
- 发送和接收出口：可以用来进行数据的加密/解密以及数据的压缩/解压缩。
- 消息出口：可以用来在消息级实现用户的特定功能，如数据转换，加密/解密等。

压缩解压程序使用开源的 ZLIB 程序。压缩使用 zlib 提供的压缩函数 compress2（Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen, int level)。解压改造 zlib 的解压函数 uncompress（Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen)，由于预先不知道解压长度，所以为了保险起见在程序中动态分配解压缓存，编写自定义函数 decompress （uLongf *destLen, const Bytef *source, uLong sourceLen, int *error)。

解压函数为 char* decompress （uLongf *destLen, const Bytef *source, uLong sourceLen, int *error)。

destLen 解压后的长度。source 压缩的字符串。sourceLen 压缩的字符串长度。Error 解压结果：Z_OK 成功，Z_MEM_ERROR 内存不足，Z_BUF_ERROR 输入输出缓存空间不足，Z_DATA_ERROR 压缩数据错，999 分配内存错。返回：解压后的字符串。注意：使用完这个函数的返回值后，要将其内存释放掉。

函数源代码如下：

```
char* decompress （uLongf *destLen, const Bytef *source,
uLong sourceLen, int *error）
{
```

```
//初始解压长度，暂定为压缩长度的 2 倍
uInt initLen = 2*sourceLen;
z_stream stream;
char *dest;
int i = 1;
Byte *uncomprTemp;
if ((dest = (char*) malloc (initLen)) == NULL)
{
    *error = 999;
    return dest;
}
strcpy (dest, "") ;
stream.next_in = (Bytef*) source;
 stream.avail_in = (uInt) sourceLen;
/* Check for source > 64K on 16-bit machine: */
if ((uLong) stream.avail_in != sourceLen)
{
    *error = Z_BUF_ERROR;
    return dest;
}
stream.zalloc = (alloc_func) 0;
 stream.zfree = (free_func) 0;
 *error = inflateInit (&stream) ;
if (*error != Z_OK)      return dest;

//临时分配缓存
if ((uncomprTemp = (Byte*) calloc (initLen, 1)) == NULL)
{
    *error = 999;
    inflateEnd (&stream) ;
    return dest;
}

//循环解压，动态分配解压内存
for (;;) {
```

```
            stream.next_out = uncomprTemp;
            stream.avail_out = initLen-1;
        if （stream.avail_out != initLen-1）
        {
            *error = Z_BUF_ERROR;
            break;
        }
          *error = inflate（&stream, Z_NO_FLUSH）;
        if（*error != Z_OK && *error != Z_STREAM_END）
        {
            break;
        }
        //每次把解压好的接到解压字符串的后面
        strcat（dest, （const char *）uncomprTemp）;
        //遇到内存流的末尾退出循环
        if （*error == Z_STREAM_END） break;
        i++;
        //解压长度不够，再分配
        if（(dest = （char*）realloc（dest, （initLen-1）*i+1)） == NULL）
        {
            *error = 999;
            break;
        }
}
//释放临时缓存
    free（uncomprTemp）;
    *destLen = stream.total_out;
     inflateEnd（&stream）;
    //*error = inflateEnd（&stream）;
    //返回解压后的字符串
    return dest;
}
```

5.7 本章小结

本章介绍了国库信息处理系在解决“信息孤岛”和在应用集成所面临的问题，并将基于 ESB 的应用集成模型应用在国库信息处理系统，详细的介绍了系统的整体架构、ESB 总体设计、ESB 与外部系统的连接、ESB 与内部系统的连接和 ESB 各个功能模块设计和实现。

结论

企业应用集成技术是利用通用的中间件，融合企业已有的应用软件、商业封装式应用软件以及新代码三方面的功能。现在，企业应用集成在 IT 行业里随处可见，由于 Web 的出现，企业对于加强与客户和合作伙伴的联系、优化内部业务流程，缩短应用程序的市场化周期的需要等这些商业发展的驱动，EAI 越来越成为人们关注的焦点。

面向服务的体系结构的出现将减少集成那些完全不同系统所需的时间，并通过快速开发和组件重用来快速部署服务。基于 SOA 体系结构的 EAI 技术目前已经成为当前企业应用整合项目中的主流框架，其基于业界标准的 WebService、SOAP、JMS、企业服务总线（ESB）等技术能对企业的应用系统高效集成和对市场快速响应。业务流程的优化使开发、部署和改进业务的循环闭合起来，使企业能够以最少的开销来改善其服务。

本文分析了 EAI 技术的产生和发展，研究 SOA 产生的原因，重点介绍了 SOA 的定义、体系结构、特点和价值。然后详细阐述了 ESB 的概念、功能模型和应用模式，提出了基于企业服务总线的应用集成模型，并将该模型成功的应用到中国人民银行的国库信息处理系统。

本文主要完成了以下几个方面的工作：

（1）对国内外企业应用集成技术产生和发展进行了综述，分析了国内外在应用集成的现状同时指出了银行业对 EAI 技术的迫切需求。在此基础上，确定了本文的研究范围和内容。

（2）通过分析当前的应用集成技术存在的问题，在当前企业应用的要求下，了解了传统技术存在的不足和新挑战。剖析了 SOA 产生原因，详细论述了 SOA 的定义、组成、特点和优势。

（3）研究并与早期的 Hub 结构对比，分析了新的基于企业服务总线的 SOA 集成技术，从而改进了 SOA 的原有体系结构，从根本上提升了企业应用集成的灵活性。并总结了企业服务总线来的功能模型和主要应用模式。

（4）通过 ESB 对 SOA 体系结构的改进的研究与分析，归纳出了一种基于 ESB 企业应用集成模型。

（5）把基于 ESB 企业应用集成模型应用到人民银行的国库信息处理系统中，有效的解决了该系统与外联系统集成过程中所遇到的难题。

利用 SOA 的体系结构来进行企业应用集成在国内外都得到广泛的应用，但是其支撑技术却得到不断的发展还更新，但是在国内，仅停留在 EAI 的一些简单技术的研究，许多关键技术的研究还停留在探索阶段。同时，基于 ESB 的

SOA 实现技术越来越受到重视，其本身也处在发展和成长中，但是目前，各大厂商虽然投入大量资源去开发基于 SOA 的企业系统架构，但多数都不够成熟，在实际应用中还不够理想。而且，相当多的问题还没有得到解决，还需要进一步研究：

作为一种基础化企业 IT 架构平台，涉及到性能、合理引擎设计、事务处理和任务监控等。本文涉及此方面的内容较少。

对于企业应用中遗留系统的封装，异构系统的集成，本文只给出了基本的数据模型和适配器模型，而且也只提出了主流组件的封装技术，对于企业应用中的其它组件的封装包括特定应用系统的集成，还没涉及到。

灵活、随需应变的跨系统工作流引擎也是 SOA 体系结构中的另外一个重要部分，它要求不同系统之间的工作流能够通过整合，协同工作，因此一种名为 BPEL（业务流程执行语言）的新标准的出现为解决上述问题迈出了关键一步，该标准能够可以与运行在任意平台(例如 J2EE 和.Net)上的 Web 服务进行通信，从而创造出跨系统的，灵活的工作流。在本文中，只涉及到不同系统间的数据通信，而未对工作流进行研究，需要在以后的工作中进一步研究。

参考文献

1 Francisca Losavio, Dinarle Ortega, Maria Perez. Modeling EAI[A]. Proceedings of the XXII International Conference of the Chilean Computer Science Society（SCCC'02）[C]，2002:22-27

2 Sharif, A.M. Elliman, T. Love, P.E.D. & Badii, A. "Integrating the IS with the enterprise: key EAI research challenges", The Journal of Enterprise Information Management, Vol. 17, No. 2, 2004, pp.164-170.

3 Lee, J, Siau, K., and Hong, S. "Enterprise Integration with ERP and EAI", Communications of the ACM, Vol. 46, No. 2, 2003, 54-60.

4 Jeff Sutherland, Willem-jan van den Heuvel. Enterprise Application Integration Encounters Complex Adaptive Systems: A Business Object Perspective [A]. Proceedings of the 35th Hawaii International Conference on System Sciences[C], 2002, IEEE.

5 Linthicum, D. Enterprise Application Integration. Addison-Wesley, MA, USA, 1999.

6 周晖.基于 Web 服务的企业应用集成技术研究[D] 北京：北京理工大学，2003.3

7 F.A.Rabhi, H.Yu, F.Dabous and S.Wu, A Service-Oriented Architecture for Financial Business Processes, FiananceCom05 international conference, 2005

8 Wendy.浅析：SOA 核心理念的应用发展.[EB/OL]. http://www.enet.com.cn/article/2006/0213/A20060213500360.shtml.

9 IBM.SOA-面向服务的体系结构.[EB/OL].http://www-128.ibm.com /developerworks/cnlwebservices/ws-themelws-soa.htm1.2004-07.

10 Tom B. and Luc C. UDDI Version 3.0. UDDI.org Published Specification. http://uddi.org/pubs/uddiv3.htm, July 2002.

11 Eric Newcomer,Greg Lomow.《Understanding SOA with Web Services 中文版》：电子工业出版社，2006-7

12 Mark E.Atkins. Closing the EAI Gap with Data Integration [M]. February 2003

13 Martin G. and Marc H. SOAP Version 1.2 Part] Messaging Framework. W3C Candidate Recommendation. http://www.w3.org/TR/soapI2-part-l,Dec 2002.

14 Paul Patrick, "Impact of SOA on enterprise information architectures", Proceedings the 2005 ACM SIGMOD international conference on Management of Data, Baltimore, Maryland, USA, June, 2005, pp. 844-848.

15 Michelson B M. Enterprise Service Bus Q&A[EB/OL]. http://ebizq.net/bot topics/esb/feahmes/6117.htm1

16 Cbappell D. Enterprise Service Bus [M]. O'Reilly Publisher, 2004

17 Frank Leymann, "The （Service） Bus: Services Penetrate Everyday Life", ICSOC 2005, Amsterdam, Netherlands,December, 2005, pp. 12-20.

18 Min Luo, Benjamin Goldshlager, and Liang-Jie Zhang, "Designing and implementing Enterprise Service Bus（ESB）and SOA solutions", Tutorial, IEEE International Conference on Services Computing, SCC 2005, Orland, FL, USA, July, 2005, p. xiv.

19 柴晓路，梁宇奇 WebServices 技术、架构、应用[M].北京:电子工业出版社，2003

20 Keen MAcharya A, Bishop S. Patterns: Implementing an SOA Using an Enterprise Service Bus [EB/OL]. http://www.redbooks.ibm.com/ redbooks /pdfs/sg246346.pdf.

21 Martin Keen, Jonathan Bond. Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture [EB/OL]. http://www.redbooks.ibm.com/ redbooks /pdfs/sg246773.pdf.

22 谢小轩. 典型的企业应用集成方法介绍[J]. AMT 企业资源管理研究 2003.11:43-47

23 陈亚华.企业应用集成架构研究与实现[D] 上海: 复旦大学 2004.5

24 Harold E R, Means W S. XML 技术手册[M].北京:中国电力出版社, 2001.

25 吴永英,吕继云,班鹏新.基于 JMS 和 XML 的数据集成研究[J].计算机应用研究,2004,（7）:43-45.

26 李军怀,等.基 XML 的企业异构数据集成方法研究[J] .计算机工程,2002,28（9）:63-64.

27 李晓东，杨 扬，郭文彩.基于企业服务总线的数据共享与交换平台[J].计算机工程,2006,Vol.32,No.21:218-219

28 刘迎春,兰雨晴,于乐乐.ESB 中的数据交换技术[J].计算机系统应用，2005,（7）:42-43

29 Chris Britton. IT 体系结构与中间件[M]. 刁联旺，李彬译. 北京：人民邮电出版社，2003

30 Vinoski, S. “Where is middleware?”, IEEE Internet Computing, Vol. 6, No. 2, 2002, pp. 83-85.

31 Chris Britton 著。刁联旺、李彬译. IT 体系结构与中间件：建设大型集成系统的策略.人民邮电出版社。2003.7

32 范国闯，陈宁江，钟华，2004，Web 应用服务器，新一代中间件，计算机科学，Vol.31，1-4

33 耿建光，赵钢，章翔峰,等. 企业信息集成系统中的过程管理[J]，计算机集成制造系统.2001.12 Vol7, No12.

34 刘发贵,王宇军. IBMS/390 事务处理-CICS[M]. 杭州：浙江大学出版社 2001

35 陈宇翔. 精通 WebSphere MQ[M]. 合肥：安徽科学技术出版社，2003：23-28

36 于慧龙.MQSeries 队列管理器的远程管理[J].计算机安全:2003.1:671-672

37 Saida Davies,Peter Broadhurst. WebSphere MQ V6 Fundamentals

[EB/OL]. http://www.redbooks.ibm.com/ redbooks/pdfs/sg247128.pdf.

38 娄丽军,WebSphere Business Integration Message Broker 循序渐进[EB/OL].http://www.ibm.com/developerworks/cn/websphere/library/techarticles/loulijun/0404_WBIMB/part1.htm

39 刘晶炜，闫健卓.《IBM 信息集成技术原理及应用》：电子工业出版社，2004-5

40 Priscilla Wahnsley. XML 模式权威教程[M].北京:清华大学出版社，2003.1

41 Weihong Qi, Sharon Crook. Tools for neuroinfonnatic exchange: an XML application for neuronal morphology data Neurocomputing, 58-60（2004）, 1091-1095

42 Zeng Xiaochun, Zeng Jiaoyan, Guo Qiang. Research of trust on B2C electronic commerce.ICEC2005 Conference Proceedings,2005:221-225

43 L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana,and G. Rackham, Impact of service orientation at the business level ,IBM System Journal, Volume 44, Number 4,2005

44 Lam, W. & Shankararaman, V. "An Enterprise Integration Methodology", IT professional, Vol. 6, No.1, 2004, pp. 40-48.

45 甘荃，娄丽军.《IBM WEBSPHERE MQ 基础教程》：电子工业出版社，2004-1

攻读硕士学位期间所发表的学术论文

1 张书杰,鄯嘉. 企业服务总线在银行系统中的应用, 计算机与信息技术, 2007（4）

致谢

本文的研究工作是在我的导师张书杰教授的精心指导和悉心关怀下完成的，在我的学业和论文的研究工作中无不倾注着导师辛勤的汗水和心血。导师的严谨治学态度、渊博的知识、无私的奉献精神使我深受的启迪。从尊敬的导师身上，我不仅学到了扎实、宽广的专业知识，也学到了做人的道理。在此我要向我的导师致以最衷心的感谢和深深的敬意，并真诚地祝愿他身体健康、工作顺利、家庭幸福！

感谢同在中国人民银行软件开发中心实习时我的项目经理陈光权、同事丌秀燕、林志荣和我的同学曾永远、杨利征、赵燕平等给予了我很大帮助。与他们在人民银行项目上的讨论，给了我很大启发和新想法，祝愿他们在今后的人生之路上一帆风顺、飞黄腾达。

感谢我的父母和亲人，虽然他们不在我身边，但是他们的关心、爱护、鼓励和支持永远陪伴我，使我得以顺利地完成今天的学业.在此，我要深深地祝福他们。

在此，向所有关心和帮助过我的领导、老师、同学和朋友表示由衷的谢意！