摘　要

Web 服务（Web Services）作为一种新兴的中间件技术，已经被广泛应用于各种电子商务中。就目前而言，它仅仅局限于基于同步信息获取的简单应用，而对于以异步传输和高可靠性为特征的复杂应用的支持尚不足，这就大大降低了 Web 服务在诸如信贷业务等一些大型关键应用领域的竞争力。随着 Web 服务应用范围的扩展和研究的不断深入，灵活的消息传输机制和可靠的服务质量成为许多研究者关注的热点问题。但对于 Web 服务中消息传输机制和可靠性的定义目前尚未形成统一的标准和规范。

本文以基于 Internet 的应用集成为背景，以现有的 Web 服务体系结构和规范集合为基础，基于消息中间件技术，研究了 Web 服务中的异步消息传递机制和可靠性，并以此为基础开发了一个能够提供异步可靠 Web 服务的中间件系统。

本文的主要工作包括：

1．本文基于 Web 服务体系结构研究了 Web 服务的消息机制，分析了同步传输与异步传输两种不同的底层消息传输机制，以及同步操作与异步操作两种不同的消息模式，并在此基础上详细讨论了基于同步传输实现异步 Web 服务操作的四种消息模式；

2．研究了可靠 Web 服务的参考模型与特点，在此基础上分析了现有 Web 服务规范集合对于可靠性的支持，提出了基于成熟的 MOM（Message Oriented Middleware）技术保证服务器端消息的可靠投递与应用的可靠执行。

3．基于我们研制的 Web 服务部署与管理环境 StarWebAdaptor 设计并实现了支持异步可靠 Web 服务的 SOAP2JMS 中间件系统。该系统由 SJIT（SOAP-JMS Inter-Transformer）和 AM（Asynchronous Messaging）组成，SJIT 完成了 SOAP 消息和 JMS 消息之间的互译，并对整个 SOAP-JMS 系统进行管理和配置；AM 则提供了 Web 服务请求者和提供者之间的消息传递机制。

本文对 Web 服务中异步性和可靠性的探索将对提高 Web 服务的效率和服务质量、扩展 Web 服务的应用领域有着重要的实际意义。

关键词：Web 服务，异步消息，可靠性，面向消息的中间件

ABSTRACT

Being an emerging middleware, Web Services (WS) have been widely applied in all kinds of e-commerce. Presently those applications are all confined in the simple business coming from the synchronous message acquirement. However it is not good for those sophisticated business which need the asynchronous message transmission and high reliability. So it embarrasses WS competition on the large-scale and key applications domain. With Web Services' application scope expanding and further researching, the flexible message transmission and reliable quality of service have collected many attentions from lots of investigators. However there are not existing uniform standards and specifications defining what message transmission and reliability are under the specific Web Services context.

This thesis bases on the current Web Services architecture, specifications and Message Oriented Middleware, takes the credit domain as typical application scene to investigate the asynchronous message transmission and reliability. Then, on the basis of our study, we develop a middle-ware system for supplying asynchronous and reliable Web services.

The primary contributions of this thesis include the following three parts.

Firstly, based on the WS concept architecture this thesis investigates message mechanism, analyzes the underlying synchronous/asynchronous message transmission and synchronous/asynchronous message operations. Making the former research as foundation, this thesis discusses at length four message patterns based on the synchronous message transmission which could complete asynchronous WS operation.

Secondly, we advance the reliable Web Services reference model and discuss its characteristics. Then after analyzing the support from the existing WS specifications, we put forward the server side reliable message delivery and the reliable implement of applications based on the assurance from the mature Message Oriented Middleware.

Last, grounded on our developed WS deployment and configuration tool StarWebAdaptor1.0, we design and implement a supplying asynchronous reliable WS middleware system called SOAP-JMS, which consists of SJIT (SOAP-JMS Inter-Transformer) and AM (Asynchronous Messaging). SJIT completes the inter-translation between SOAP and JMS, further more provides the management and configuration of entire system; AM supplies the flexible message transmission between the Web Services providers and requestors.

The exploration to the asynchronism and reliability of WS has the significant practice meaning for improving the performance , quality of service of WS and expanding the scope of WS applications.

Keywords: Web Services, asynchronous message, reliability, Message Oriented Middleware

独 创 性 声 明

本人声明所呈交的学位论文是我本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和撰写过的研究成果，也不包含为获得国防科学技术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谢意。

学位论文题目：异步可靠 Web 服务关键技术的研究与实现

学位论文作者签名：左克　　日期：2003 年 12 月 20 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；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。

（保密学位论文在解密后适用本授权书。）

学位论文题目：异步可靠 Web 服务关键技术的研究与实现

学位论文作者签名：左克　　日期：2003 年 12 月 20 日

作者指导教师签名：[illegible]　　日期：2003 年 12 月 25 日

图目录

第一章 绪　　论

§1.1 研究背景

对于什么是 Web 服务，目前存在着很多的定义，我们这里引用 W3C 组织公布的 Web 服务体系结构（Web Services Architecture）中对 Web 服务的定义力图阐述其具有的特点：

Web 服务是描述一些操作（利用标准化的 XML 消息传递机制可以通过网络访问这些操作）的接口。Web 服务是用标准的、规范的 XML 概念描述的，称为 Web 服务的服务描述。这一描述囊括了与服务交互需要的全部细节，包括消息格式（详细描述操作）、传输协议和位置。该接口隐藏了实现服务的细节，允许独立于实现服务基于的硬件或软件平台和编写服务所用的编程语言使用服务。这允许并支持基于 Web 服务的应用程序成为松散耦合、面向组件和跨技术实现。Web 服务履行一项特定的任务或一组任务。Web 服务可以单独或同其它 Web 服务一起用于实现复杂的聚集或商业交易。

Web 服务体系结构基于三种角色（服务提供者、服务注册中心和服务请求者）之间的交互。交互涉及发布、查找和绑定操作。这些角色和操作一起作用于 Web 服务构件：Web 服务软件模块及其描述。在典型情况下，服务提供者托管可通过网络访问的软件模块（Web 服务的一个实现）。服务提供者定义 Web 服务的服务描述并把它发布到服务请求者或服务注册中心。服务请求者使用查找操作来从本地或服务注册中心检索服务描述，然后使用服务描述与服务提供者进行绑定并调用 Web 服务实现或同它交互。服务提供者和服务请求者角色是逻辑结构，因而服务可以表现两种特性。图 1.1 显示了这些操作、提供这些操作的组件及它们之间的交互。

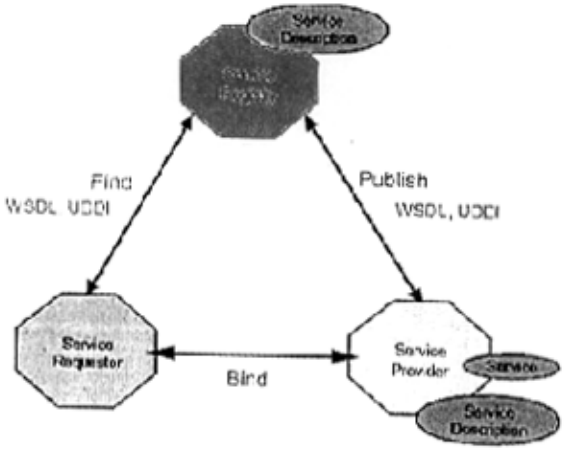

图 1.1 Web 服务角色、操作和构件

作为一个分布式应用的开发环境，Web 服务必须有一个包含每一层标准的 Web 服务

协议栈。图 1.2 展示了一个概念性 Web 服务协议栈。上面的几层建立在下面几层提供的功能之上。

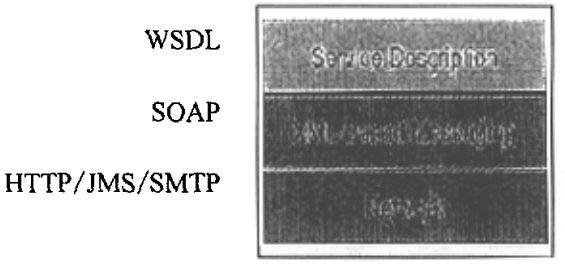

图 1.2 Web 服务协议栈

Web 服务协议栈的基础是网络层。Web 服务要被服务请求者调用，就必须是可以通过网络访问的。因特网上可以公用的 Web 服务使用普遍部署的网络协议。HTTP 凭借其普遍性，成为了因特网可用的 Web 服务真正的标准网络协议。Web 服务还可以支持其它因特网协议，包括 SMTP 和 FTP。

从 Web 服务的体系结构中可以看出，Web 服务并未限制其使用的应用模式，也并未将其与底层的传输协议进行绑定，所以，在 Web 服务体系结构和标准规范之内，为应用提供异步性与可靠性支持是可行的。

§1.2 应用背景

今天，处理 XML 编码 SOAP 消息、通过 HTTP 作为首选的底层传输协议、使用 WSDL 描述的 Web 服务已经广泛应用在互联网上。它的使用范围主要集中在应用集成方面：从简单的，尤其是数据共享的应用到大规模的互联网零售业和现金交易。目前，不断发展的 Web 服务已经涉足于移动计算、移动设备和网格的应用之中。

作为 Web 服务的突出特点，它提供了在异构的企业计算环境中，分布的软件组件之间良好的互操作性，而这也总是广大 IT 消费者，开发者以及参与者所面临的棘手问题。HTTP 和 SOAP 协议提供了传输和消息一级的互动，WSDL 通过描述了部署的 Web 服务和交换接口定义以提供分布应用间的互操作。这些基本的规范使得消费者和 IT 厂商能够解决许多重要的实际应用问题，在这些成功的基础之上，开发者正试图利用 Web 服务技术解决更为复杂的问题和提供更加优秀的功能。利用已经开发处理的 Web 服务工具，开发者希望加强互操作的功能和质量。而许多商业执行过程也希望中间件提供类似于异步消息、安全、事务和可靠方面的功能。

为了让我们对异步可靠 Web 服务有一个大致的了解，来看图 1.3，该图反映了广泛使用的一种 Web 服务场景，而且也阐明了今天在 Web 服务世界中广泛存在的 Web 服务设计

问题。

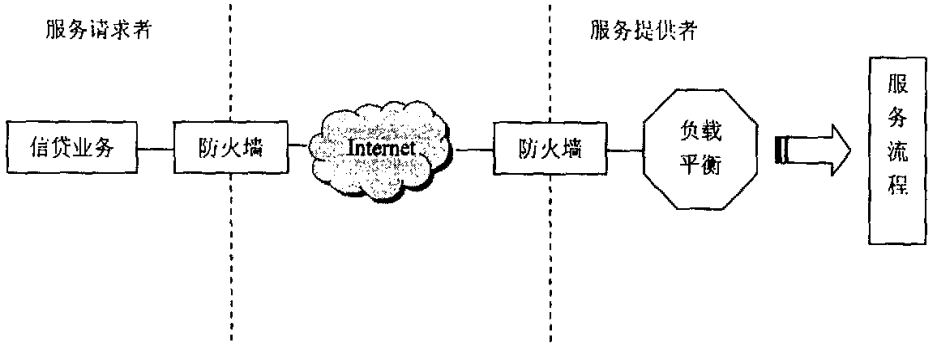

图 1.3 Web服务典型应用场景

图中，服务请求者是带有一个基本产品目录的控制系统，该系统负责运行商用处理逻辑以跟踪并管理所有产品信息的变化。服务提供者是一个带有负责平衡部件的系统，它使用网络复杂平衡技术将到达的消息分配到各个服务流程。通讯的双方使用异构的体系结构（包括代理和防火墙等）连接至互联网上提供服务。双方进行交换的信息对于他们的商业操作来说都是至关重要的。首先，服务流程的处理时间通常较长，往往超过了在一个网络链接超时的时间内返回执行结果；其次，由于客户与业务流程交互过程中，通常传输的数据都是比较重要的商业数据，对数据传输和执行的可靠性有较高的要求。因此，对异步和可靠的消息传递机制的要求随之产生。

目前，对消息传输和可靠性主要有以下的解决方案：

- 消息传输：
  - 试图使基于 SOAP 的消息能够运行在 除 HTTP 之外的其它协议上，如 JMS；
  - 试图使 SOAP 消息能够用于单向消息传递、双向消息传递和对等会话的工作。
- 可靠性：
  - 通过标准 / 合作组织制定新的规范；
  - 通过硬件、操作环境和基础架构供应商改进其产品确保更好的可靠性。

本课题试图在 Web 服务的消息传输和可靠性方面进行一定深度的探索，在消息传输机制上，设计并实现使 SOAP 能够用于异步消息传输的机制，实现双向异步消息的传递；在可靠性方面，以成熟的消息中间件技术为基础框架，力图在服务提供方内部确保消息和服务的可靠性。

§1.3 课题来源

本课题来源于 863 课题，基于 Web Services 的应用集成关键技术研究（No.2002AA116040）。

§1.4 课题目标

以异步性和可靠性为目的，基于分布式计算平台 Web 服务，研究 Web 服务提供的消息机制；之后研究成熟的消息中间件系统及其对消息可靠性的支持；最后，研究在关键消息传递环境中异步消息通信机制和可靠消息通信制；在上述研究的基础上，基于 StarWebAdaptor 平台，设计并实现了 SOAP-JMS 桥接系统。

§1.5 论文结构

在绪论之后，第二章将分别详细介绍异步 Web 服务的概念和实现机制，着重讲述同步传输、异步传输、同步操作和异步操作在 Web 服务环境中的不同含义；以及就可靠性展开讨论，其中着重讲述 Web 服务环境下， MOM 技术与 Web 服务的关系。在这两章的研究基础之上，在第三章中，将就 Web 服务下，实现异步性给出详细的设计模式，在可靠性方面，给出可靠 Web 访问的参考模型，并讨论它的特点。之后的第四章根据前一章的设计原理，进行详细的设计。最后的第五章将对我们实现的系统进行简单的测试，并给出分析结论。应该说，第二章中研究的重点贯穿了全文，并且提供了在第四章中设计和实现的思路。

§1.6 本文的研究成果

本文对 Web 服务环境下的异步消息机制和可靠性进行了深入的研究，在此基础上设计了可以提供异步可靠 Web 的 SJIT 系统，并在 StarWebAdaptor1.0 平台上加以实现。同时，通过对该系统的测试，验证了它的异步性和可靠性，从而能够有效的提高系统的效率。在研究期间，以第一作者在第 20 届全国数据库学术会议论文集上发表文章一篇（见本文附录）。

第二章 课题关键技术研究

异步性作为一种与时间相关的技术手段，一直以来就广泛存在于计算机世界中，本章将在 Web 服务的环境中，详细介绍与异步性相关的特点，并从理论的角度着重讲述四个概念：同步传输、异步传输、同步操作、异步操作；之后，将根据这些概念给出几种异步服务的设计模式；最后简要介绍目前这个领域一些有代表性的研究成果。

§2.1 异步 Web 服务关键技术

2.1.1 同步传输

用于交换 Web 服务消息的传输可以归为同步和异步两类。其中同步传输包括：

- HTTP；
- HTTPS；
- RMI/IIOP；
- SMTP

这里就 HTTP 的同步传输进行简要的介绍。

HTTP 是一个属于传输层的协议，由于其简捷、快速的方式，适用于分布式信息系统。HTTP 协议的主要特点可概括如下：

- 支持客户/服务器模式。
- 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
- 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
- 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
- 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

2.1.2 异步传输

传输协议本身就支持响应消息与请求消息的相关性以便应用程序使用，并支持“推”（push）和“拉”（pull）类型消息交换的传输通常被称为异步传输。可以用于支持异步操作的传输的示例包括：

- HTTPR；
- JMS；
- IBM MQSeries 消息传递（IBM MQSeries Messaging）；
- MS 消息传递（MS Messaging）

这里对 JMS 异步传输机制进行简要的介绍。

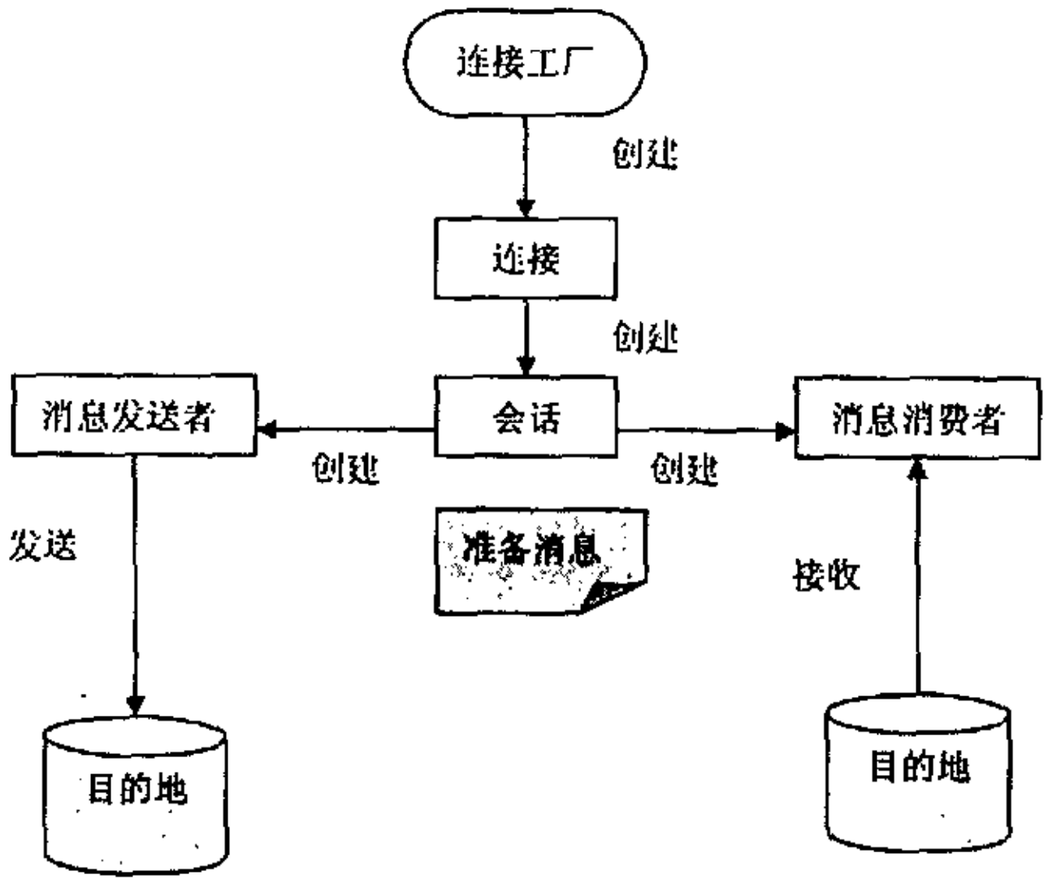

图 2.1 JMS 编程模式

JMS 异步消息传输是在软件组件或应用程序之间进行通信的一种方法。JMS 系统是一种对等的消息设施；一个消息传递客户可以向其他任何客户发送消息；或从任何其他客户接收消息。每个客户连接到一个消息传递代理，该代理提供了创建、发送、接收和读取消息的能力。

JMS 异步消息传输应用了松散耦合的分布式通信。一个组件把消息发送到目的地，接收者能够从目的地获取这个消息。但是，消息的发送者和消息的接收者不需要同时活跃才能进行信息通信。事实上，双方不需要对对方有任何的了解。发送者和接收者只需要使用相同的消息格式，目的地的位置即可。在这方面，JMS 消息传递不同于紧耦合的技术，如远程方法调用（Remote Method Invoke RMI），它要求应用程序了解远程方法接口的详细信息。JMS 消息传递也不同于邮件系统，邮件是人与人之间或人与软件之间进行通信的一种方法，消息传递用于软件应用或软件组件之间的通信。

不管正用于异步操作的传输机制是否异步，客户机（或者客户机所用的服务代理）和服务提供者负责生成一个相关标志，传输在管理请求和响应时可以使用这个相关标志。通常情况下，当商业伙伴利用 Web 服务集成他们的业务流程时，他们更倾向于使用 HTTP、HTTPS 和 HTTPR 作为传输进行跨 Internet 通信；在企业内，当存在相似的应用程序平台

时，将使用本机传输及接口，比如 JMS、RMI/IIOP 和 JCA（Java 连接体系架构，Java Connection Architecture）。异步传输使客户机能够在请求一个服务调用后继续在自己的执行线程上处理；它们还提供一些机制使客户机能够确定它的 Web 服务请求的状态并能够检索对那些请求的响应。不提供在另一个执行线程上初始化响应传输能力的 Web 服务的实现将无法用于异步操作。这种实现的示例是那些在前端数据库应用程序中使用的 EJB 或者通过使用本地接口（比如 JCA）提供对企业系统的访问的实现。

2.1.3 同步操作

在通常的情况下，请求与应答作为一个不可分割的事务被视为整体。为了更好的解释，我们引入以下的定义：

定义 2.1：Web 服务环境Ω是一个四元组（P，MT（Message Transmitter），O，M）。

- P 是客户程序，它由一个操作序列 $op_1op_2op_3\cdots op_n$ 构成。其中，任一操作 op_i 可以是本地操作，也可以是调用远端服务的操作。
- O 是远端服务接口，接收并执行来自客户的请求。
- MT 是 P 和 O 之间的传输渠道，起着通信中介的作用。
- M 是消息集合，可表示为 $M = \{ m_1, m_2, \cdots, m_k \}$，它包括 P 和 O 通过 MT 传递的所有请求和应答。每个 m_i（$m_i \in M$）的生命期都是从 P 开始，经 MT 到 O，然后再从 O 经 MT 返回 P，结束。当 m_i 从 P 经 MT 传递给 O 时，m_i 包含的是请求消息；当 m_i 从 O 经 MT 返回 P 时，m_i 包含的是应答消息。为了描述一个消息的生命期，P 在发出每条消息的时候，都为它们分配了一个标识，我们用下标来表示。消息在传递过程中，内容可以由请求消息转变为应答消息，但消息的标识是不会改变的。为了保障标识的唯一性，也为了便于不同的消息进行比较，消息标识定义为时间的函数，即若 P 先发消息 m_i，后发消息 m_j，则一定 $i < j$。

定义 2.2：若 op_i 是远程操作，则 op_i 定义为一个动作序列 $a_1(m_{k1})a_2(m_{k2})\cdots a_n(m_{kn})$，$(m_{k1}, \cdots, m_{kn} \in M)$。其中，动作 $a_i(m_{ki})$ 表示该动作对标识为 k_i 的消息 m_{ki} 进行处理，且 $a_i(m_{ki}) \in \{S(m_{ki}), R(m_{ki})\}$，其中 $S(m_{ki})$ 表示发送一个标识为 k_i 的消息给 MT，$R(m_{ki})$ 表示从 MT 读取一个标识为 k_i 的消息。

定义 2.3：远程操作 op_i 属于同步调用模型，当且仅当 $op_i = S(m_{ki})R(m_{ki})$，$m_{ki} \in M$。（如图 2.2）

定义 2.4：远程操作 op_i 和 op_j（$j > i$）属于延迟同步调用模型，当且仅当 $op_i = S(m_{ki})$，且 $op_j = R(m_{ki})$，$m_{ki} \in M$。（如图 2.2）

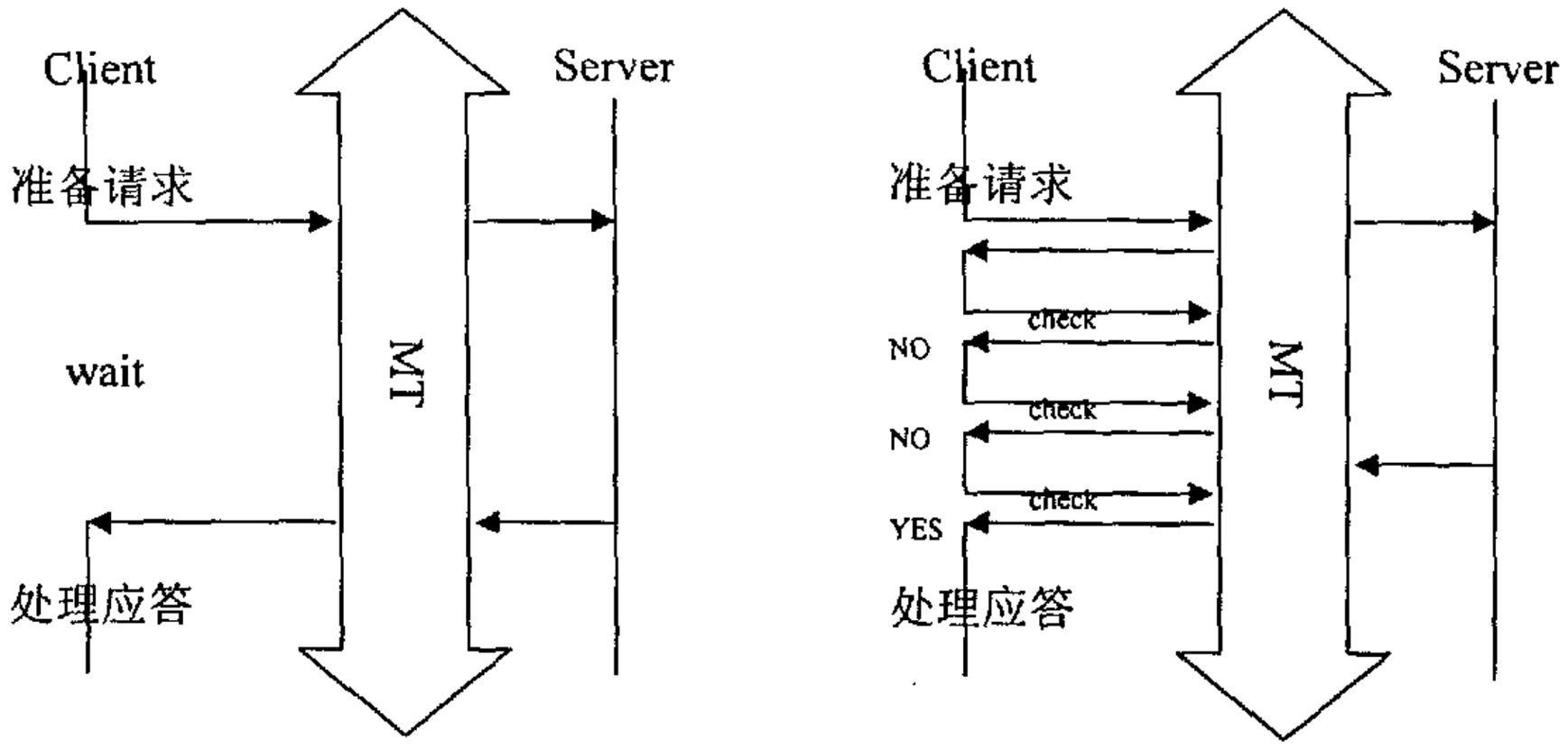

图 2.2 同步模型与延迟同步模型

根据前面的定义，同步模型采用了严格同步的操作处理请求消息和应答消息。如图2.2，client 发出请求后，阻塞等待应答。当应答返回后，client 继续执行应答的处理逻辑。该模型的优点是调用逻辑符合人的自然思考方式，编程简单。而缺点是对于单线程 client，当它阻塞等待应答时，无法处理任何其它工作，不适于需要充当 client 角色的 server 应用。

在实现上，针对同步模型的一个改进是引入多线程机制，即以不同的子线程加载同步操作，当子线程等待应答消息的时候，主线程继续处理其它操作，这在一定程度上可以改善系统资源利用率。但是，线程的使用又带来了新问题。首先，过多线程的使用会降低系统的可伸缩性。当同步操作的数目比较大时，众多的线程本身将占用系统大量的宝贵资源，同时线程之间的切换和管理也使系统增加额外的开销，这些将使系统性能大大降低。其次，在目前的分布对象标准中，线程属于未被标准化的部分，因此线程的使用还存在可移植性的问题。

延迟同步调用模型放松了同步模型的同步要求（如图 2.2），client 发送请求后，执行线程返回，继续处理其它工作。随后，client 可以在多个不同的时间点向 MT 查询应答是否返回。如果应答返回，client 从 MT 提取应答，并执行相应处理。其优点是避免 client 阻塞等待，能够在 server 调用服务期间，并行地处理一些其它事情，有利于提高 client 的资源利用率。其缺点是把应用逻辑分割为不同时间点上的离散步骤，当许多调用交叉重叠在一起时，编程复杂；有些情况下，server 服务于调用的时间可长可短、难以预料，查询应答的成功率有时会很低，随着查询次数的增加，client 的效率越来越低。

oneway 调用模型是对前述模型的简单化，具有一定的异步性，即操作向 MT 提交请求消息后即返回。但是，这种模型没有给应答消息的处理做出定义，客户程序仅通过 oneway 调用模型无法得知请求消息是否被目标对象服务以及服务结果如何。因此，对于许多应用（包括高性能应用），oneway 调用模型出于太简单而不能胜任。

为了克服 oneway 调用模型的问题，有人提出了双向 oneway 的调用模型[4]，即 client

在调用目标对象的一个 oneway 操作的同时，传递一个本地对象的对象引用给目标对象。然后，驻留在 server 上的目标对象就可以利用这个对象引用，反过来向最初的 client 调用另一个 oneway 操作。这种方法的不足之处是，server 方目标对象需要提供两套服务代码，一套针对普通请求即同步 two-way 操作和采用动态调用接口（DII）的延迟同步操作，另一套专门针对 oneway 操作，这使得 server 变得冗余且庞大。另外，在每个请求中增加一个对象引用，使得网络开销明显增加了。特别是，这种方法无法处理系统异常。当请求在 server 方被服务期间产生系统异常，那么这次请求将被流产，但是 client 却无法得知这一情况。这是因为 client 发出的 oneway 调用本身不支持任何应答（包括含有异常的应答）返回，而 server 产生异常的时候，server 也无法通过另一个 oneway 返回任何信息。

在 Web 服务的环境中，SOAP 协议典型的操作方式是同步的 RPC 方式（SOAP/HTTP），因此，以这种方式开发的 Web 服务，就不可避免的要受到同步操作所带来的种种弊端。

2.1.4 异步操作

然而并不是所有的 Web 服务都以同步的方式工作，在某些情况下，对于 Web 服务请求的应答并不能够立即产生，而是在请求过程结束之后，并断开链接之后，所有具有这些特点的操作称为异步操作。目前，Web 服务规范和标准集合并没有显示的给出异步操作的定义，但是，现有的 Web 服务系统框架和机制完全可以作为异步操作的实现基础。

从本质上来说，分布式对象间的操作都是异步的，同步是在异步机制的基础之上加入了某些控制机制。因此，Web 服务调用本质上也是异步的，服务的提供者应该具有在没有任何约束的前提下，接收来自客户方请求的能力。这里把客户方发出请求的过程和服务方给出相应应答的过程看作两个单独分开的事务。这种异步的操作对于需要花费几分钟甚至可能几天的时间才能完成的复杂处理服务来说很常见 — 例如，当 Web 服务实现依赖于批处理或者需要人工介入的手工步骤时。

基于分布对象的异步消息模型是对传统分布对象模型的扩充，它在保持传统分布对象模型基本框架的前提下，增加和补充了支持异步消息传递的异步回调模型和异步轮询模型，下面将详细介绍。

2.1.4.1 回调模式

在异步、松耦合的应用当中，client 和目标对象及 server 之间是个松散的关系，client 发送请求消息之后，可能要等待很久应答消息才能返回。采用延迟同步调用模型，可以使 client 在发送请求消息之后先返回控制权，以便处理一些其它操作。但是，为了取得应答消息，client 必须不断地向 MT 发出应答查询操作，这在编程上是件繁琐的事情，因此开发人员希望应答消息返回时能够自动被执行处理。并且，在松耦合应用当中，client 向 server 发送请求消息后就可能会退出。当 client 再次恢复时，从 server 返回的应答消息如何被提交给 client 处理？传统调用模型无法直接解决这个问题，人们希望应答消息也能够像一个

新的请求消息那样，通过回调 client 中某个对象来执行相应处理。针对以上异步、松耦合应用提出的这些问题和需求，异步回调模型出现了。下面，我们就给出异步回调模型的定义。

首先，客户程序 P 的定义在异步回调模型中得到扩充，新的定义是，

定义 2.5：客户程序 $P = \{ op_1op_2op_3\cdots op_n, s_1, s_2, s_3, \cdots, s_m \}$。其中，操作序列 $op_1op_2op_3\cdots op_n$ 的定义与定义 1 一致。s_1，s_2，s_3，…，s_m 表示驻留在客户程序中的一组回调服务地址。

其次，远程操作和动作的定义得到扩充，增加一种新的异步发送操作和相应动作，即

定义 2.6：若 op_i 是异步发送操作，则 $op_i = AS(m_{ki}, s_{ji})$，$m_{ki}\in M$，$s_{ji}\in P$。其中，动作 $AS(m_{ki}, o_{ji})$ 表示客户把标识为 k_j 的消息 m_{ki} 和标识为 j_i 的回调服务地址 s_{ji} 一起提交给 MT。

最后，我们给出异步回调模型的完整定义，

定义 2.7：远程操作 op_i 属于异步回调模型，当且仅当 $op_i = AS(m_{ki}, s_{ji})$，$m_{ki}\in M$，$s_{ji}\in P$，且 $m_{ki}\in C_{MT}$。其中，C_{MT} 表示 MT 以回调方式处理的消息的集合。所谓回调方式是指 MT 在处理应答消息时，把应答消息作为一个特殊的请求消息，调用前面异步操作 $AS(m_{ki}, s_{ji})$ 所提交的回调对象 s_{ji}。

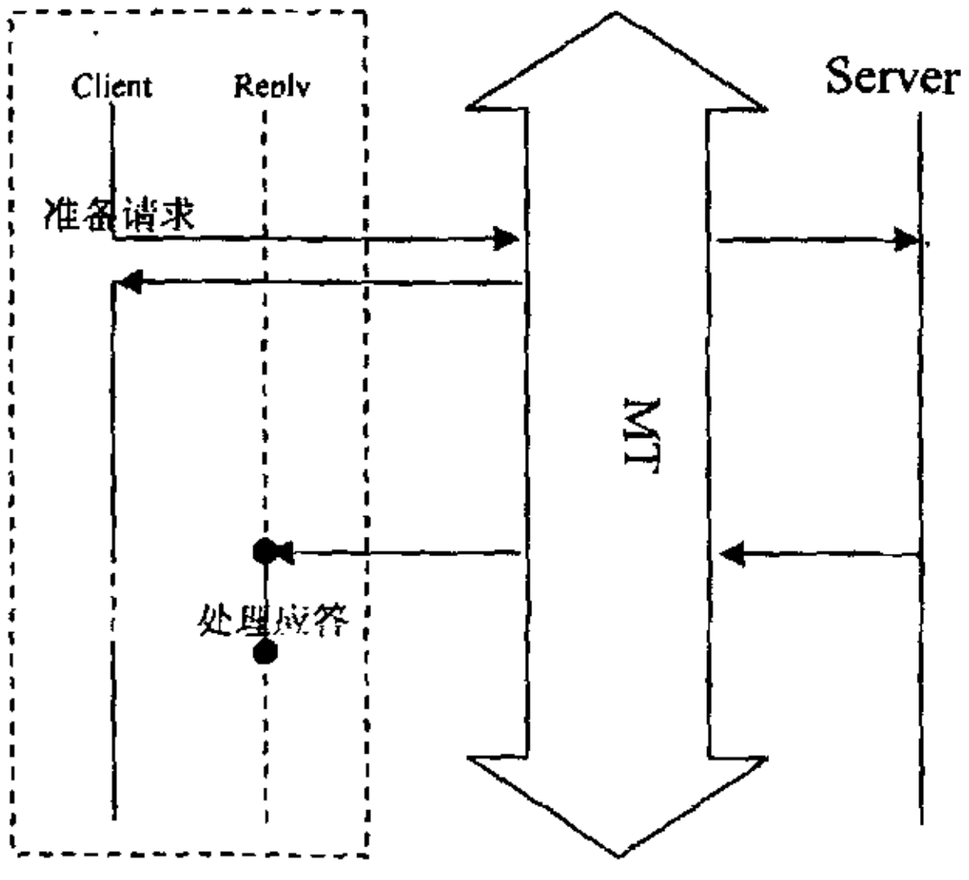

图 2.3 回调模式

根据定义 2.7，client 向 MT 发送请求的时候（如图 2.3），注册一个 ReplyHandler 对象。请求发出后，client 的执行线程返回，继续处理其它事情。当应答返回后且 MT 得到执行线程时，MT 回调 ReplyHandler 对象，执行应答处理逻辑。实际上，异步回调模型使得 client 能够按照一种事件驱动的方式处理应答消息。

这种模型的优点是与延迟同步一样，client 不必阻塞等待应答；模型把应答的处理逻辑分割出来，置于 ReplyHandler 对象中，使得请求后面的应用逻辑不必再考虑应答的接收和处理，这种分割简单而规则，没有导致太多的离散步骤，编程难度不大；对于许多调用交

又重叠在一起的情况，此模型在编程上比延迟同步模型要简单、有效得多；支持基于窗口的 client 应用，该模型可以和窗口事件的处理统一纳入事件驱动的大循环中，实现一致的回调处理；利于实现时间无关的应答处理，即 ReplyHandler 与 client 应用逻辑可以实现在不同的进程空间中，client 发送异步请求后，ReplyHandler 对应答的处理与 client 是否活跃无关。

可以把交换的消息看作这样的数据报，为了使事务被处理，该数据报不需要或不期望任何答复。通过使用这种数据报，考虑到双方之间的真正异步关系，完全可以把消息的发送方（或称始发方）与接收方分开。

2.1.4.2 轮询模式

异步回调模型解决了异步、松耦合应用当中应答消息的自动处理，但是在有些时候，开发人员也需要能够像延迟同步模型那样，读取应答消息并进行相应处理。因此，在异步回调模型的基础上又出现了异步轮询模型。然而，异步轮询模型与延迟同步模型还是有着本质的差别。异步轮询模型基于异步回调模型实现，因而它对异步、松耦合应用具有内在的支持。例如，client 在发送请求消息之后就退出了，当 client 再次恢复时，异步轮询模型支持 client 通过轮询的方式取得应答消息。而延迟同步模型则无法直接满足异步、松耦合应用的这种需求。

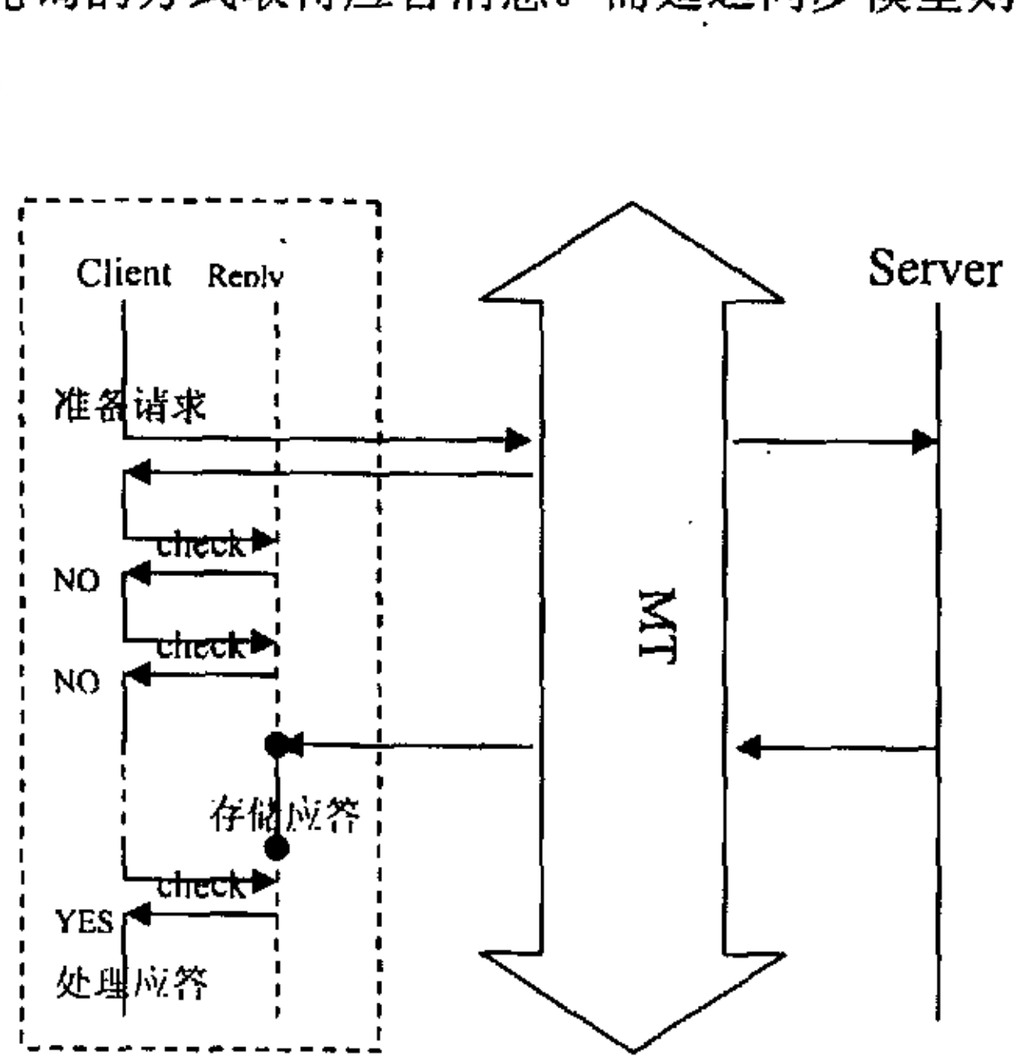

图 2.4 轮询模式

为了准确定义异步轮询模型，我们首先对异步操作定义进行扩充，

定义 2.9：若 op_i 是异步接收操作，则 $op_i = AR(m_{ki}, s_{ji})$，$m_{ki} \in M$，$s_{ji} \in P$。其中，动作 $AR(m_{ki}, o_{ji})$ 表示客户试图从标识为 j_i 的回调服务接口 s_{ji} 读取标识为 k_i 的消息 m_{ki}。如果读取成功，则 op_i 操作顺利返回；否则，op_i 操作可以阻塞等待，也可以立即返回。

下面，我们给出异步轮询模型的完整定义，

定义 2.10：远程操作 op_i 和 op_s 属于异步轮询模型，当且仅当 $op_i = AS(m_{ki}, s_{ji})$，$op_s = AR(m_{ki}, s_{ji})$，其中 $m_{ki} \in M$，$s_{ji} \in P$，$m_{ki} \in C_{MT}$，$s > i$。

根据以上定义，该模型是以异步回调模型为基础的，当 client 引发一个调用时，系统自动为这个调用生成一个缺省 ReplyHandler 对象（如图 2.4），并注册到 MT 中。请求发出后，client 的执行线程返回，继续处理其它事情。在请求发出后的多个不同时间点上，client 可以向缺省 ReplyHandler 对象查询应答返回情况。当应答返回时，MT 通过回调把应答存储到缺省 ReplyHandler 中。然后，当 client 再次向缺省 ReplyHandler 对象查询应答时，就可以提取应答，并执行相应处理。

模型的优点是 client 在发出异步请求后，通过该模型能够在需要时主动取得应答，弥补了异步回调模型的部分不足；利于时间无关的应答查询和处理，即如果我们把缺省 ReplyHandler 的生命期与 client 相分离，为缺省 ReplyHandler 设计持久的生命期和存储能力，那么即使 client 不活跃，应答也能通过回调而被持久存储下来。当 client 再次活跃时，它仍能从 ReplyHandler 获得应答。

通过以上的定义和分析表明，回调和轮询两种异步调用模型克服了传统调用模型存在的一些弊端，更能够适合于异步、松耦合的需要。

2.1.5 小 结

传输与操作是传输层和应用层的不同概念，而同步与异步则是与时间相关的两种交互模式。值得注意的是，同步/异步传输与同步/异步操作在逻辑上是正交的关系，即同步/异步操作既可以使用同步传输机制，也可以使用异步传输机制。

将同步操作转变为异步操作时，必须解决许多同步时不必关心的问题。异步实现需要解决的问题包括：

- 定义一个相关标志和一种相关机制；
- 定义一个回复地址（reply-to address）指定应该把响应发送到何处，并确保向服务提供者通知了这个目的地；
- 服务提供者生成响应的过程作为一个事务与请求分开；
- 客户机处理异步响应的方式；
- 客户机和服务提供者把响应与请求关联起来。

这些问题将在第三章提出的设计模式中逐一解决。

§2.2 可靠 Web 服务关键技术

随着 Web 服务在实际应用中所扮演的角色越来越重要，服务间进行的消息传递和事务流程的可靠性也日益重要。尤其是在那些对在服务执行过程中所产生的数据丢失问题敏感

下面，我们给出异步轮询模型的完整定义，

定义 2.10：远程操作 op_i 和 op_s 属于异步轮询模型，当且仅当 $op_i = AS(m_{ki}, s_{ji})$ ， $op_s = AR(m_{ki}, s_{ji})$，其中 $m_{ki} \in M$， $s_{ji} \in P$， $m_{ki} \in C_{MT}$， $s > i$。

根据以上定义，该模型是以异步回调模型为基础的，当 client 引发一个调用时，系统自动为这个调用生成一个缺省 ReplyHandler 对象（如图 2.4），并注册到 MT 中。请求发出后，client 的执行线程返回，继续处理其它事情。在请求发出后的多个不同时间点上，client 可以向缺省 ReplyHandler 对象查询应答返回情况。当应答返回时，MT 通过回调把应答存储到缺省 ReplyHandler 中。然后，当 client 再次向缺省 ReplyHandler 对象查询应答时，就可以提取应答，并执行相应处理。

模型的优点是 client 在发出异步请求后，通过该模型能够在需要时主动取得应答，弥补了异步回调模型的部分不足；利于时间无关的应答查询和处理，即如果我们把缺省 ReplyHandler 的生命期与 client 相分离，为缺省 ReplyHandler 设计持久的生命期和存储能力，那么即使 client 不活跃，应答也能通过回调而被持久存储下来。当 client 再次活跃时，它仍能从 ReplyHandler 获得应答。

通过以上的定义和分析表明，回调和轮询两种异步调用模型克服了传统调用模型存在的一些弊端，更能够适合于异步、松耦合的需要。

2.1.5 小 结

传输与操作是传输层和应用层的不同概念，而同步与异步则是与时间相关的两种交互模式。值得注意的是，同步/异步传输与同步/异步操作在逻辑上是正交的关系，即同步/异步操作既可以使用同步传输机制，也可以使用异步传输机制。

将同步操作转变为异步操作时，必须解决许多同步时不必关心的问题。异步实现需要解决的问题包括：

- 定义一个相关标志和一种相关机制；
- 定义一个回复地址（reply-to address）指定应该把响应发送到何处，并确保向服务提供者通知了这个目的地；
- 服务提供者生成响应的过程作为一个事务与请求分开；
- 客户机处理异步响应的方式；
- 客户机和服务提供者把响应与请求关联起来。

这些问题将在第三章提出的设计模式中逐一解决。

§2.2 可靠 Web 服务关键技术

随着 Web 服务在实际应用中所扮演的角色越来越重要，服务间进行的消息传递和事务流程的可靠性也日益重要。尤其是在那些对在服务执行过程中所产生的数据丢失问题敏感

的业务，Web服务的可靠性成了急待解决的问题。

2.2.1 典型场景

为了更好的说明Web服务对可靠性的需求，我们从一个典型的Web服务应用场景说起，这个场景代表了目前Web服务在设计过程中不得不面对的一些棘手的问题。

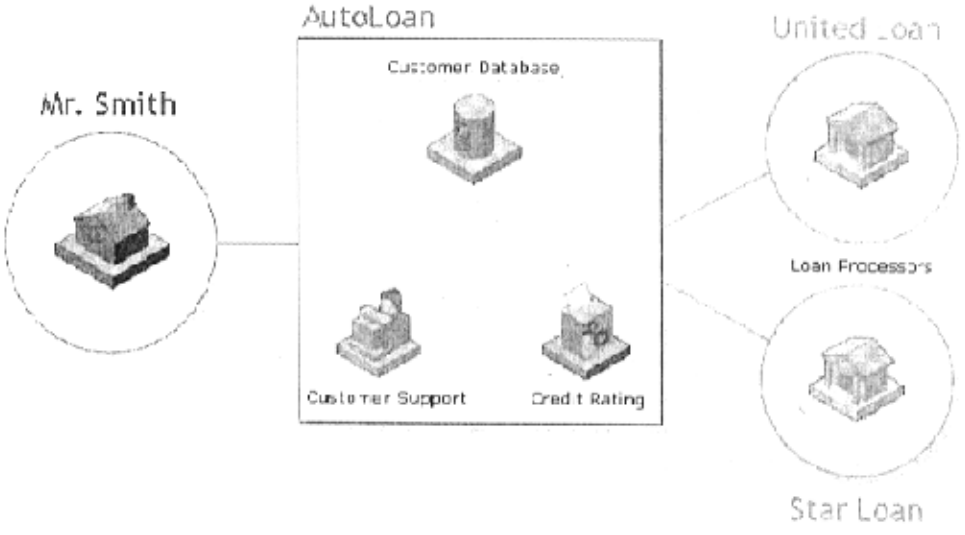

图 2.5 典型场景

假设有一家名为“AutoLoan”的信贷公司，通过其在线服务，可以向客户提供诸如购买汽车的贷款业务。AutoLoan需要和资金方、信息系统提供商、客户信息系统提供商一同来完成该业务，除此之外，它还需要一个类似客户服务代理之类的提供与客户交互的业务方。所有这些都可以通过Web服务来实现，但是，与普通的Web服务明显区别在于，这种应用场景下，对信息交换和事务处理的可靠性要求较高，因为这些被用来交换的信息是至关重要的。

2.2.2 MOM简介

企业消息传递系统（Enterprise messaging systems），通常称为面向消息的中间件（Message Oriented Middleware），它提供以松耦合的、灵活的方式来集成应用程序的机制。它们在存储和转发的基础上支持应用程序间数据的异步传递，即每个应用程序彼此不直接通信，而是与作为中介的MOM通信。

MOM使用在应用组件间放置一个逻辑队列的方法达到给紧密耦合的组件关系进行松绑的目的。其中的一个组件充当消息的发送者，将消息传递至队列中；而在之后的某一时刻，充当消息接受者的另一个组件从该队列中读取消息并进行相应的处理。

MOM 提供了两种消息传递模式，他们分别是 publish/subscribe（pub/sub）、point-to-point（P2P）。在 pub/sub 模式中，它允许一条消息可以被多个 subscriber 消费，即所有登记为某一消息主题（Topic）的订阅者都会收到订阅消息的一份拷贝。

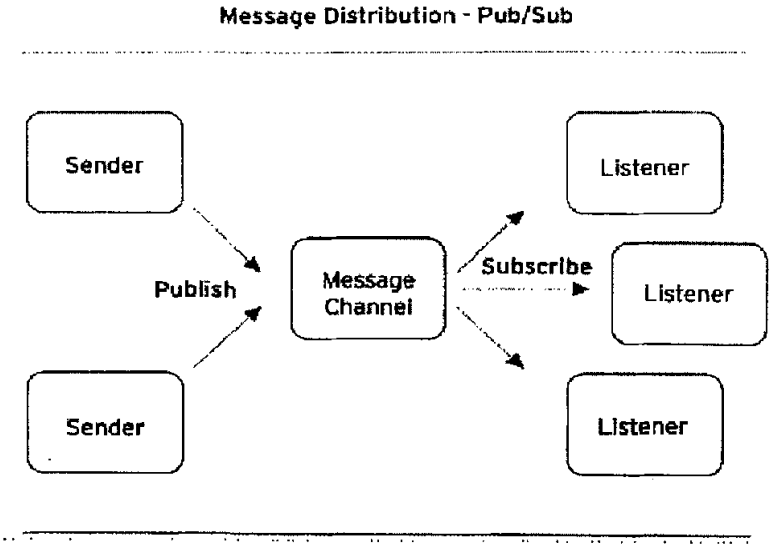

图 2.6 Pub/Sub 模式

在 P2P 模式中，支持多个消息的发送者向同一个队列（Queue）发送消息，但确保每一条消息只会被一个特定的消息接收者获取。

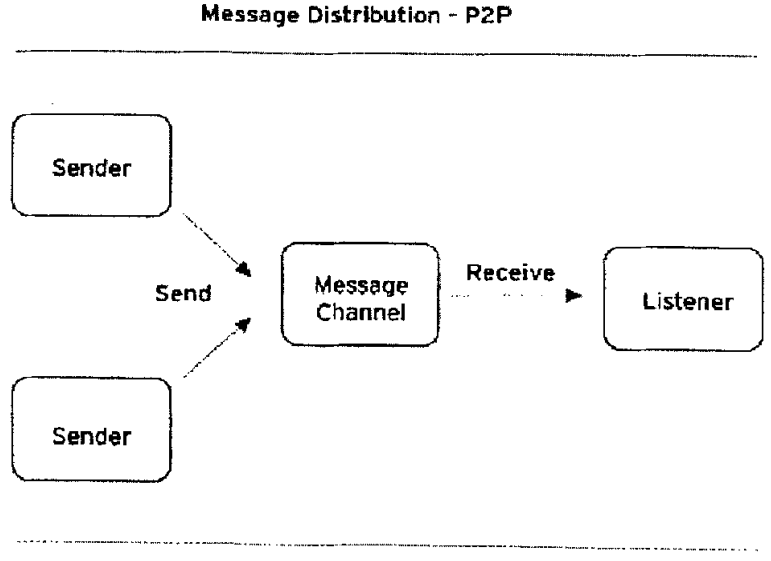

图 2.7 P2P 模式

我们来看这样一个场景：

应用程序 A 通过 MOM 的应用程序编程接口（API）来发送消息，从而和应用程序 B 通信。

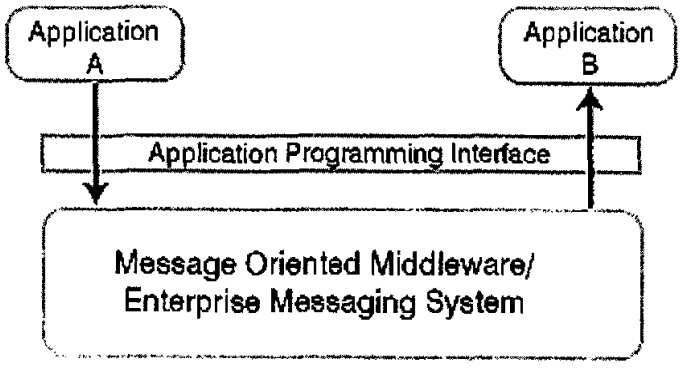

图 2.8 JMS 应用场景

通过 MOM 路由消息给应用程序 B（可能位于完全不同的计算机上）；MOM 对网络通信进行处理。如果没有网络连接，MOM 将一直存储消息直至获得网络连接，然后将消息转发给应用程序 B。

灵活性的另一个方面是应用程序 A 发送消息时，应用程序 B 甚至可以不处于执行状态。MOM 会一直保留消息到应用程序 B 开始执行并试图取回消息为止。这还可以避免应用程序 A 在等待应用程序 B 接收消息时阻塞。

这种异步通信要求应用程序的设计与现在已设计的大部分应用程序稍有不同。不过对于时间独立或并行处理来说，这会成为一种比较有效的方法。企业消息系统真正的力量在于应用程序的松耦合。在图 3.6 中，应用程序 A 发送它的消息，其中指示了特定的目标，例如“订单处理”程序。以后我们可以用不同的订单处理程序代替应用程序 B，但应用程序 A 不会察觉这一点。它会继续发送它的消息给“订单处理”，而消息也会继续被处理。同样，我们也可以替换应用程序 A，只要那个替代者继续为“订单处理”发送消息，订单处理程序不必知道有一个新的应用程序正在发送订单。

从上述可以看出，以 MOM 为基础的结构框架提供了非常重要的优势，包括：

- 消息的确保发送和消息的持久化；
- 松散耦合的系统建模方式；
- 系统结构的可扩展性和聚类性（clustering）；
- 对异步编程模式的支持；

由 MOM 提供的这些特性正好符合了 Web 服务的发展需求，整合两种技术可以为我们获得以下目标：

- 可靠性：对于大多数使用 HTTP、HTTPS、SMTP 作为传输协议的 Web 服务来说，MOM

提供的经过验证可靠的（tried-and-trusted）的异步性、大容量、事务性特点都是必不可少的；

- 可扩展性：某些软件对系统的可扩展性有着较强的要求，因此必须采用异步的或者是非阻塞的设计方式；
- 异步性：对于某些面向工作流的 Web 服务或者需要较长时间，甚至需要人工干预才能够完成的 Web 服务，提供异步操作和异步的消息传输机制是必然的。而利用 MOM 的异步编程特性可以很好的为这些 Web 服务提供异步性。

JMS（Java Message Service）是一组定义 JMS 客户如何访问企业消息传递产品设施的接口和相关语义。在 JMS 出现之前，每个 MOM 供应商通过专有的 API（通常有多种语言编写的版本，包括 Java 语言）支持应用程序访问它们的产品。JMS 通过 MOM 产品为 Java 程序发送和接收消息提供了一种标准可移植的方法。用 JMS 编写的程序能够在实现 JMS 标准的任何 MOM 上运行。所以，我们的设计中将采用 JMS 作为访问 MOM 系统的接口。

2.2.3 MOM 与 Web 服务

2.2.3.1 场景一 ：SOAP/JMS

目前，作为 Web 服务发送 SOAP 消息的标准传输层协议，HTTP 简单、普及并且有着良好的格式，而这些特点都应归功于 HTTP 的无状态性的设计。它可以在不修改网络拓扑结构的前提下穿越防火墙。

然而，HTTP 并不适合于某些应用，这些应用需要他们发送的消息确保到达目的地。HTTP 是非事务性的，这表示如果在消息传输的过程中出现问题，则消息的发送者无法判断服务方是否成功的收到了消息。而且，Internet 网络环境下，对于一些消息类型的传输来说也是不可靠的。A 客户在调用 Web 服务之后，如果在调用超时之前没有获得结果，则没有办法决定消息是否被服务方接收，或者消息正在被处理、或者在传递的过程中消息丢失。除此之外，如果重新发送请求，又无法确保请求是否冗余。可选的解决办法是使用 JMS 而非 HTTP 作为底层传输协议，这尽管会带来一定的开销，譬如预先定义 JMS 消息队列和主题，但由 JMS 带来的可靠性却是 HTTP 所无法提供的。SOAP 规范中说明，SOAP 消息可以使用许多可选的格式作为底层传输协议。因此，SOAP/JMS 是一种可选的方案。

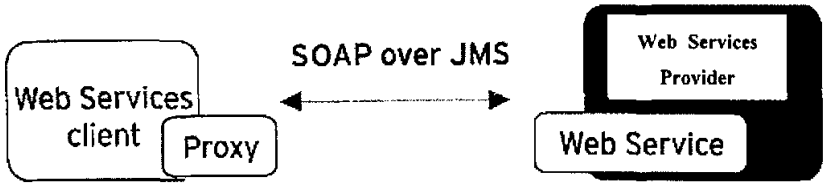

图 2.9 SOAP/JMS

SOAP/JMS 对于 Web 服务的设计存在一定的影响。对于使用 request/response 模式，标准 RPC 类型的 Web 服务而言，可以方便的使用 MOM 带来的好处而不需修改。系统从一个队列中读入 SOAP 请求处理之后，放入另一个应答队列中，所有配置的和使用消息传输带来的复杂性全部由 MOM 系统处理。

而对于使用以异步 MOM 为传输机制为基础的 Web 服务来说，则需要使用在 request/response 方式下所没有使用的 Web 服务结构模型和编程模型。在 WSDL 中定义了以下的交互模式：

- Request/Response：客户进程向 Web 服务发送一条消息并接收从 Web 服务返回的相应的应答消息；
- One-way：客户发送消息，但不期望从 Web 服务那里获得应答消息；
- Solicit-response：Web 服务向客户发送一条消息，之后客户返回给相应的应答消息；
- Notification：Web 服务发送给客户一条消息，但客户并不发送相关的响应消息。

在开发异步客户时，SOAP 客户发送消息之后继续自己的进程，这可以有效的提高系统的效率。这种类型的客户不会在消息发送之后就立刻收到返回消息，而是使用某些机制例如轮询和回调的方式处理异步的应答消息。

2.2.3.2 场景二 ：作为 Web 服务的 MOM 应用

对于那些已经实现基于消息的应用，则可以利用发布 WSDL 来部署成为可以接收 SOAP 消息的 Web 服务，而这些系统可以利用不同的技术，诸如 J2EE 和 CORBA 来实现 Web 服务。

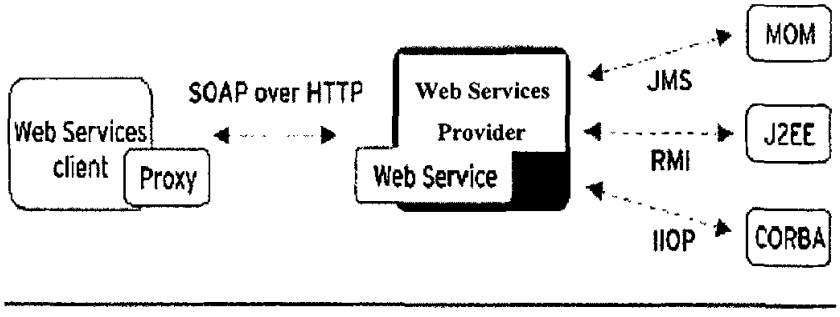

图 2.10 MOM 作为一种 Web 服务

通常在 MOM 为基础的应用中，最常见的有三种消息类型：

- XML
- Text
- Binary

在某些情况下，如果放入 MOM 数据队列中的 XML 格式的消息已经是正确的 XML 式，则只需要简单的从 SOAP 信封中提取数据并直接将该 XML 数据传递给 MOM 系统进行处理。不然，在传递给 MOM 系统之前，就需要进行某种 XML 数据格式的转换，此时可以利用 XSLT 脚本和 XSLT 匹配器。XSLT 匹配器是一个图形化工具，使用它产生 XSLT 脚本以匹配两种不同的 XML schemas。要处理 MOM 应用期望的 text 类型的消息，则需要一些额外的配置信息以从 WSDL 和 XML SOAP 有效载荷中进行映射。而如果 SOAP 载荷已经是正确的二进制格式，则可以解析该数据，转变为 JMS 消息并发送至指定的 MOM 消息队列或主题。

2.2.3.3 场景三 ：调用 Web 服务的 MOM 应用

最后一种就是将 MOM 应用作为 Web 服务的客户端，来调用远端的 Web 服务，这样使得已经存在的消息应用可以使用 Web 服务。应用逻辑可以操作输入消息并作为参数，通过本地代理调用发送给 Web 服务。这可以利用 HTTP 或者其他的 SOAP 传输机制。

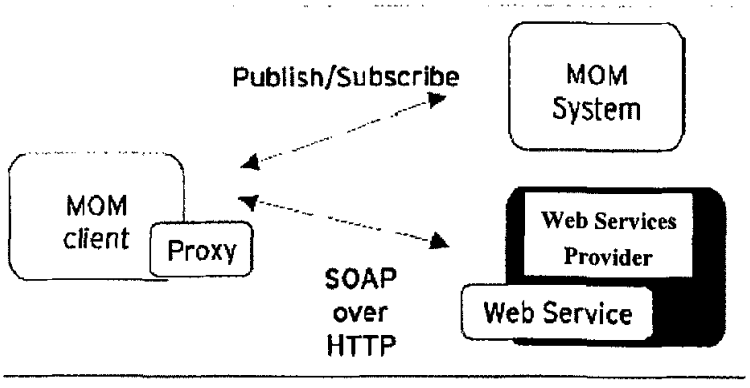

图 2.11 MOM 调用 Web 服务

这种应用模型会随着 Web 服务的增长而变得越来越有价值。通过部署它自己的 Web 服务定义，消息应用的开发者现在可以利用 Web 服务构建企业内部或者外部的整合应用。

§2.3 相关技术的研究和比较

2.3.1 SOAP/JMS

SOAP/HTTP 是目前最常用的协议绑定，它描述了如何将这些消息打包放入 HTTP 请求和应答中。由于 SOAP 协议并不限定底层传输机制，因此，SOAP 消息也可以打包放入一个 JMS 消息中，依赖 JMS 平台的传输机制发送到 Web Services。这就是 SOAP/JMS。如下图所示，SOAP/JMS 将 SOAP 信封作为一个待传输的消息封装在 JMS 消息体中，此时通常将 JMS 的消息设为 Text 类型。接收该消息的 JMS 客户负责理解 SOAP 信封，从中抽取要执行的调用或传递的消息。

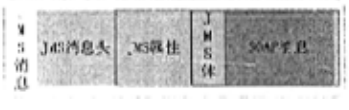

图 2.12 SOAP/JMS 消息格式

这种集成方式旨在利用 JMS 异步消息传递的可靠性，确保所有的请求消息最终都会发送到目的服务，而所有的应答最终也会被客户消费。

但是，由于 JMS 没有规定 JMS 消息的线上编码格式，因此 JMS 产品之间存在着互操作性问题，这就使得基于 JMS 传递 SOAP 消息也存在不同的实现厂商产品之间的兼容性问题，SOAP 客户与 Web Services 服务器必须使用同一厂商的运行环境（JMS 实现与

SOAP/JMS 绑定库）。而且，由于 JMS 多种于企业内部环境，因此这种方案不适合作为广域网下 web service 解决方案。右图 2 解释了这种解决方案的应用场景。

2.3.2 JMS/SOAP

JMS/SOAP 方式将 JMS 消息作为 SOAP 消息的有效载荷打包，通过 SOAP/HTTP 或其它协议绑定传递，如图所示。这种集成方案主要是为了利用 SOAP 良好的互操作性和跨广域网的能力，使得 JMS 产品可以跨广域网传送和消费 JMS 消息，而对于 JMS 用户而言，这种跨网的特点是透明的。下图给出了 JMS/SOAP 的消息格式：

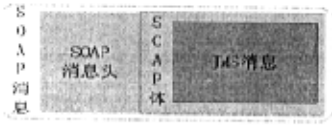

图 2.13 JMS/SOAP 消息格式

这种集成的缺点与 SOAP/JMS 相同，也是由于不同的 JMS 厂商的实现不同而带来的兼容性问题。

§2.4 本章小结

本章先从异步的角度，深入分析了 Web 服务环境下，已有的传输机制对 Web 服务的支持，并且阐明了异步传输、同步传输、异步操作和同步操作四个概念。之后分析了 MOM 与 Web 服务的关系，表明利用 MOM 技术可以提供 Web 服务器端对于消息传递和应用处理的可靠性。本章中就异步消息机制和可靠性与 Web 关系的分析，对之后的详细设计无疑是非常重要的。

第三章 异步可靠 Web 服务的设计

§3.1 异步消息处理功能的设计

3.1.1 Web 服务异步模式和基本传输类型

根据 Web 服务描述语言（Services Description Language，WSDL）规范版本 1.1 中的定义，这里讨论的支持异步 Web 服务操作的四个模式以端点可以支持的四种基本传输类型为基础：

- 单向（One-way）：端点接收一条消息；
- 请求 / 响应（Request/response）：端点接收一条消息并发送一条对应的消息；
- 征求 / 响应（Solicit/response）：端点发送一条消息并接收一条对应的消息；
- 通知（Notification）：端点发送一条消息。

每一个模式都要引入一个在客户机和服务提供者之间交换的相关标志，用于把响应与请求关联起来。相关标志可以由参与交换的任一端提供，它的创建者可以根据底层传输来确定。例如，在使用 HTTPR 和 JMS 时，消息源提供相关标志：一个事务标识或者 JMSMessageID 与 JMSCorrelationID 的一个组合。

对于单方向操作，如果使用 HTTP 或者 HTTPS，并且服务调用的接收需要由客户机确认，那么客户机的 HTTP 协议处理程序在等待 HTTP 响应（例如，一个 200 状态码）时应该阻塞调用以确保请求已被服务提供者的 HTTP 侦听器成功地接收。对于客户机使用服务代理的情况，任何与请求有关的错误状态都应该会导致异常被抛出。为服务代理的接口建模使其模型与服务提供者定义的 WSDL 操作相匹配很重要。例如，如果一个客户机调用一个单向操作，代理将永远不会向客户机返回一个参数（例如，一个状态码）。这种交换将有效地使操作成为一个请求 / 应答操作，并且应答信息不是来自服务提供者。

3.1.1.1 模式一：单向和通知操作

在这个模式中，请求和响应是单独的 WSDL 操作内定义的两条消息。请求被建模为一个入站单向操作，响应被建模为一个出站通知操作。每条消息都被作为单独的传输层发送。这个模式提供了客户机和服务提供者之间的高级分离，因为它支持使用两个数据报在双方之间进行交换，一个用于请求，一个用于响应。

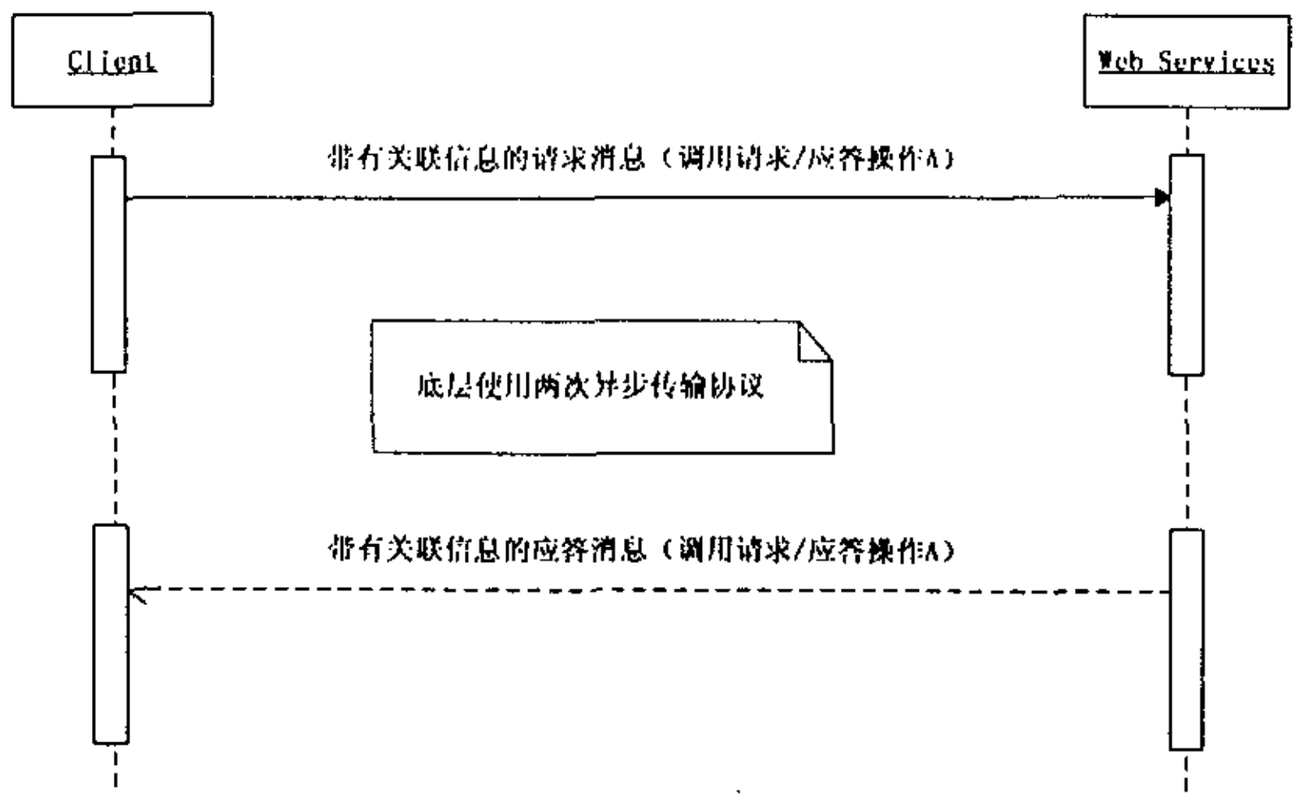

图 3.1 模式一：单向和通知操作

对于这个模式，客户机负责创建相关性标识（CorrelationID）并通过双方已同意的任意一种机制将其发送给服务提供者。SOAP 报头、HTTP 报头和 JMSCorrelationID 都是可选的机制。

定义答复地址 — 它指出响应应该发送到何处 — 也是客户机的职责，而向服务提供者通知这个地址的方法取决于 WSDL 是如何为操作定义的。如果客户机发布了一个支持单向操作的通知侦听器服务，它的 WSDL 将包含该服务的端口地址。同样，服务提供者将需要访问客户机服务的 WSDL 来确定将响应发送到何处。通过在初始请求上传递 WSDL 的引用可以在提供者的 Web 服务被部署时或运行时提供对通知侦听器服务的 WSDL 的访问权。作为替代方法，也可以把指出响应将发送到何处的特定地址（例如 URI）作为请求的参数显式提供。

3.1.1.2 模式二：基于异步传输的异步操作

在这个模式中，请求和响应是单个请求 / 应答操作内定义的两条消息，并作为两个独立无关的传输层传送发送。这个模式也可以提供客户机和服务提供者之间的高级分离，因为它支持为请求和响应使用两个数据报在双方之间进行交换。但是，要使用这个模式，服务供提供者在运行时处理信息时肯定会更复杂一点。例如，服务提供者将需要能够把它要发送响应的目标地址（例如答复地址）作为输入参数处理。

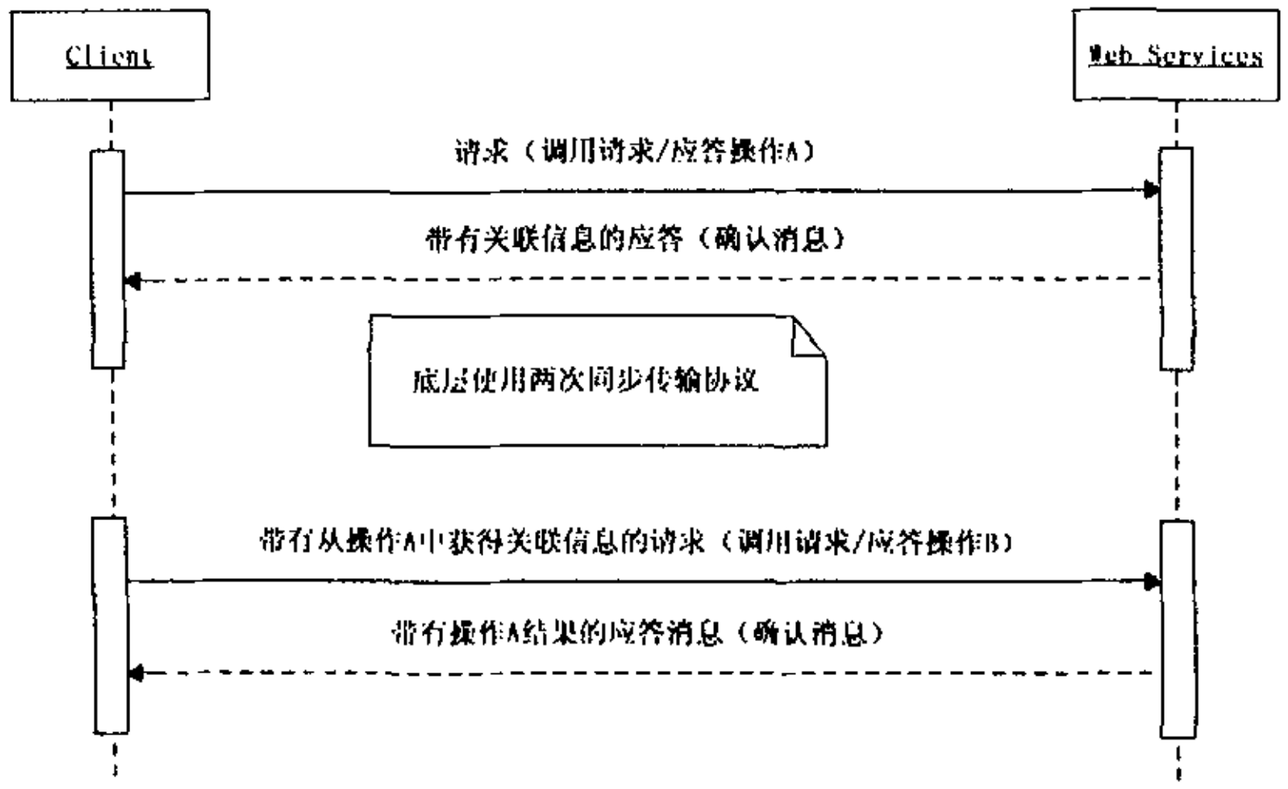

图 3.2 模式二：基于异步传输的异步操作

对于这个模式，客户机负责创建相关性标识并通过双方已同意的任一种机制将其发送给服务提供者。与前面一样，SOAP 报头、HTTP 报头和 JMSCorrelationID 都是可选的机制。定义指出响应应该发送到何处的答复地址也是客户机的职责。由于对这个模式使用单向操作，对地址或显式地址自身的引用必须作为请求的参数提供。例如，如果客户机发布了一个支持单向操作的异步响应侦听器服务，那么在初始请求上就可以提供对服务的 WSDL 的引用。这个模式也适用于其请求只产生一个响应的通用服务。模式 2 与模式 1 相似，在这种模式中，使用单独的传输层发送消息而不需要在客户机和服务提供者之间交换应用层的确认。因此，如果支持消息流的业务流程非常重要，用于这个模式的传输也必须是可靠的。

3.1.1.3 模式三：基于同步传输的轮询模式

在这个模式中，使用两个单独的 WSDL 操作内定义的四条消息处理请求和响应。初始请求被建模为请求 / 应答操作，有两条消息（一个传送和一个应答）被作为单个传输层交换发送。响应由第二个请求检索，它也被建模为请求 / 应答操作，同时有两条消息被作为单个传输层交换发送。这两个操作应该被作为同步流实现，同时，从服务提供者为每个请求返回的信息为客户机提供每个请求的一定级别的确认。这个模式使得客户机端实现在支持基于自助的解决方案时更简单，在这种模式中，客户机应用程序启动所有的交互，同时还提供客户机和服务提供者之间一定级别的分离。但是，假设请求 / 应答操作是同步的，那么应答消息流将使用本机传输应答机制（例如，HTTP 响应（HTTP Response）操作）。

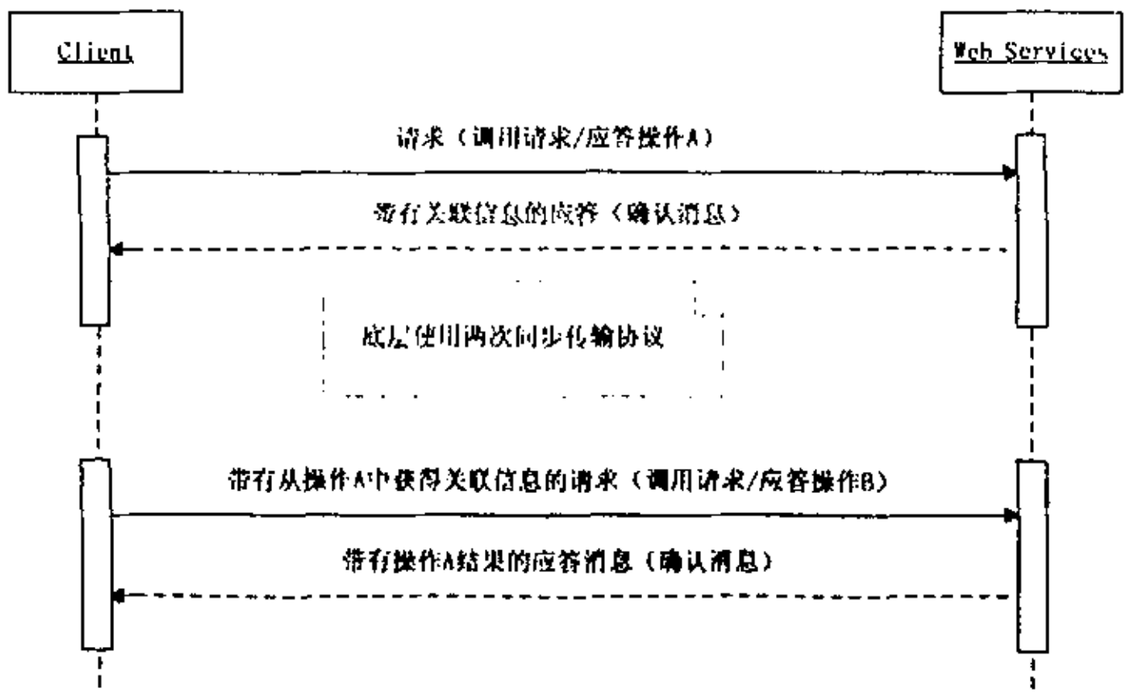

图 3.3 模式三：基于同步传输的轮询模式

如图 3.3 的示例中，服务提供者生成相关标识，客户机负责使用它检索响应；但理论上参与交换的任一方都可以创建相关性标识。由于不需要通知机制和侦听器组件，这个模式使得客户机实现更为简单。但是，客户机必须实现一个工具，使用这个工具，它可以周期性地轮询来自服务提供者的响应。这个模式的效率可能不是最高的（因为如果初始服务请求尚未完成的话，每个响应可能需要多个请求来检索响应），但它使得实现更为简单。因而，这个模式适合优先考虑简单性和服务的期望负载比较低的情况。某些服务类型可以从轮询中受益，它们的示例包括一个长期运行的业务流程的开始、生成复杂报表的请求以及基于浏览器的、面向客户的解决方案所使用的服务。

3.1.1.4 模式四：基于同步传输的回调模式

在这个模式中，使用两个单独的 WSDL 操作内定义的四条消息处理请求和响应。初始请求被建模为请求 / 应答操作，同时有两条消息被作为单个传输层交换发送。响应被建模为征求 / 应答操作，同时有两条消息被作为单个传输层交换发送。这两个操作被作为同步传输机制实现，同时，从消费方为每个请求返回的信息为请求方提供每个请求的一定级别的确认。这个模式与模式 1 相似，当使用同步传输以及客户机和服务提供者要求一个应用层确认时这个模式对 pub/sub 或事件通知服务很有用。由于这种相似性，模式 1 下给出的示例情况也可由模式四来解决；其它要求显式确认的服务类型的示例包括用来交换医疗业或金融业的对业务至关重要的信息或机密信息的服务。

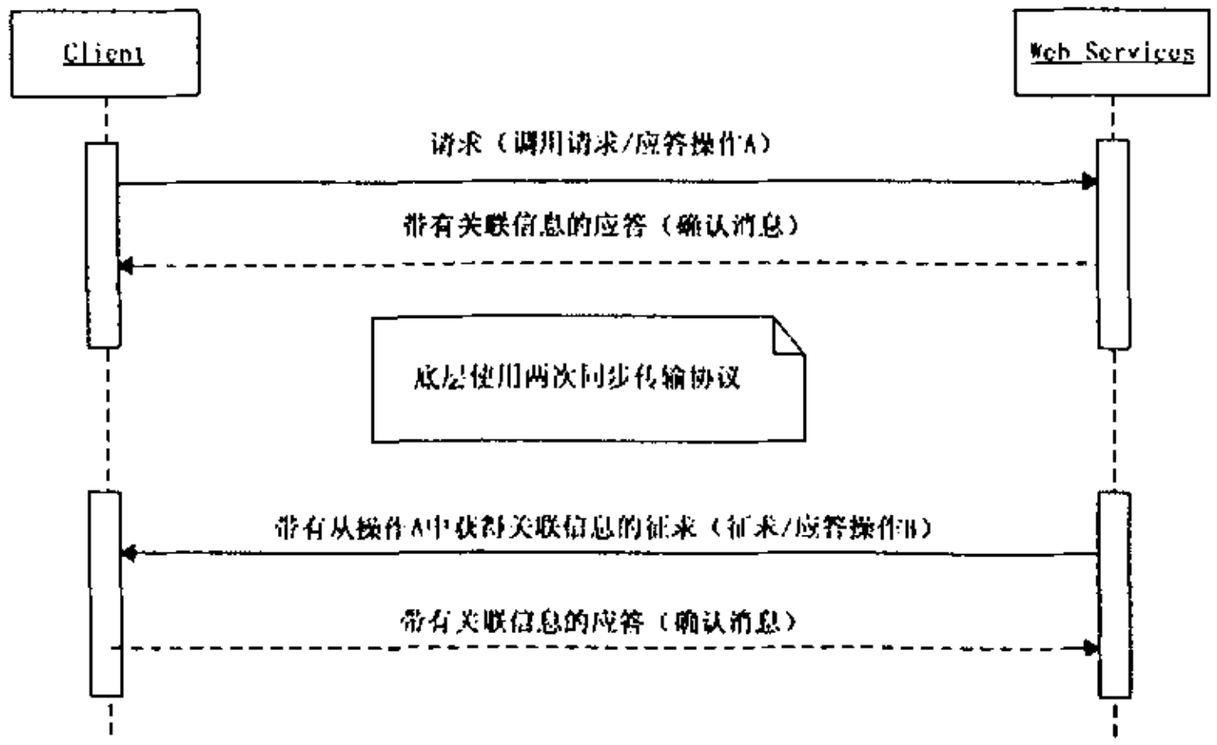

图 3.4 模式四：基于同步传输的回调模式

如果为处理响应把客户机和服务提供者的角色反过来，也可以为客户机把发送响应的 WSDL 操作定义为请求 / 应答操作。角色互换只是为了方便；这样做便于模式的开发，因为目前的工具不支持征求 / 应答操作。在双方之间流动的消息不被更改；只是本文用来描述客户机的和服务提供者的角度的模型会有不同。

对异步 Web 服务操作的支持既可以用同步传输协议，也可以用异步传输协议来实现。异步传输的使用（这时内在提供请求和响应消息的关联性并提供独立查询状态和检索响应消息的机制）使得支持客户机端和服务端的异步操作更加容易，因为面向消息的中间件为 Web 服务请求和响应的传输提供可靠的消息传递。同样，同步传输也可以支持异步操作的更简单的实现。

§3.2 可靠消息处理功能的设计

3.2.1 Web 服务可靠消息模型

在通常的网络会话中，可能产生会话中断、消息丢失、消息重复或者消息乱序，甚至消息发送的目标系统出现故障或不稳定状态。可靠消息参考模型必须提供消息的确保发送—消息发送方发出的消息保证被消息的接收方接收。

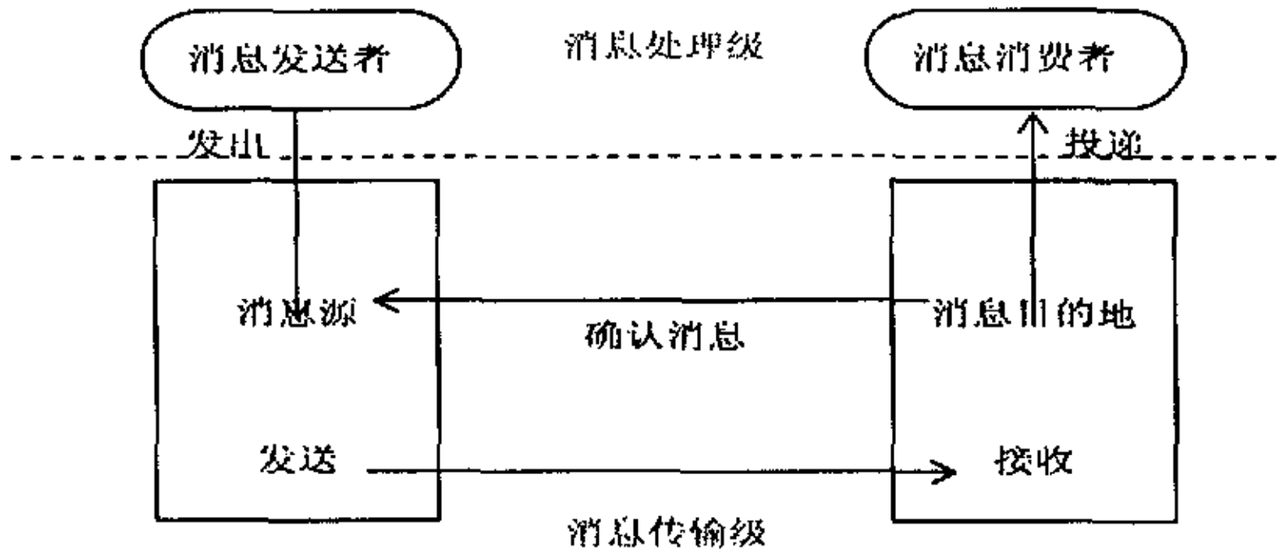

图 3.5 Web 服务可靠消息模型

首先，消息发送者向消息源提交一个等待可靠发送的消息，源接收到消息之后，向目的地发送该消息一次或多次。目的地接收之后，立刻返回该消息的确认消息。最后，消息被目的地投递给消息的最终接收者。整个模型中的角色和动作介绍如下：

- 消息结点：与 Web 服务消息相关的结点，可以是处理或者源结点等；
- 消息发送者：消息结点中负责可靠的发送消息；
- 消息消费者：消息结点中最终接收消息并进行消息可靠的处理；
- 消息确保发送：可靠消息模型对消息发送的保证，即消息的接收者最终能够收到来自消息发送者发送的消息；
- 消息源：发送消息的结点；
- 消息目的地：接收消息的结点；
- 发出：向消息源提交可靠消息传输的动作；
- 投递：将传递的消息提交给消息的接收者；
- 发送：向网络链接中写入消息内容；
- 接收：从网络链接中读取消息内容；
- 确认：从消息目的地发给消息源，指出消息成功接收的信息。

从图中可以看出，Web 服务的可靠性主要有两个部分组成：

- 消息的可靠传输过程；
- 消息的可靠产生和消费过程。

3.2.1.1 模型的前提条件

在消息传递之前，必须有一些前提提交得到满足：

- 消息源必须有目的地址的索引，并且该索引必须唯一的标识该目的地址；
- 消息源必须有关于目的地的策略知识，消息源必须有能力将自己发出的消息满足消息目的地所规定的策略；
- 在某些安全消息交换情况下，消息源和目的地必须有同一个安全的上下文

3.2.1.2 模型的满足条件

有两个满足条件必须在消息传递过程中保证满足：

- 消息源必须将自己发送的消息序列进行编号，从 1 开始，依次递增 1；
- 确认消息中，必须含有相对应目的地成功接收的消息编号的确认消息编号，并且不含有没有收到的消息编号的确认消息编号；

3.2.2 可靠 Web 服务的特点

从上面的参考模型中可以看出，作为可靠 Web 服务的关键特点，消息确保发送、重复消息消减和消息的按序发送是必须保证的。

3.2.2.1 消息确保发送机制

消息确保发送机制主要有以下四种：

- 至多一次：消息没有重复地至少发送一次，允许消息序列中的某些消息省略不被发送；
- 至少一次：每一条发出的消息都会被发送，允许消息序列中的某些消息重复发送；
- 只有一次：每一条发出的消息会被发送并且没有重复发送。这可以看作是前两种的与关系；
- 按序发送：消息发出的顺序与消息被投递的顺序相同。该特性可以与上述任何一个特点关联使用。

如果上述机制在发送的过程中出错，那么消息源或者消息目的地任意一个消息结点报出错信息。

3.2.2.2 特点间的关系

三个特点间的关系可以简单描述如下：

- 重复消减可以在没有消息按序条件下使用；

- 重复消减可以在没有消息确保发送的条件下使用；
- 消息确保发送可以在没有重复消减的条件下使用；
- 消息确保发送可以在没有消息按序发送的条件下使用。

3.2.3 一个例子

下图解释了在 Web 服务的环境下，消息可靠传递的情况：

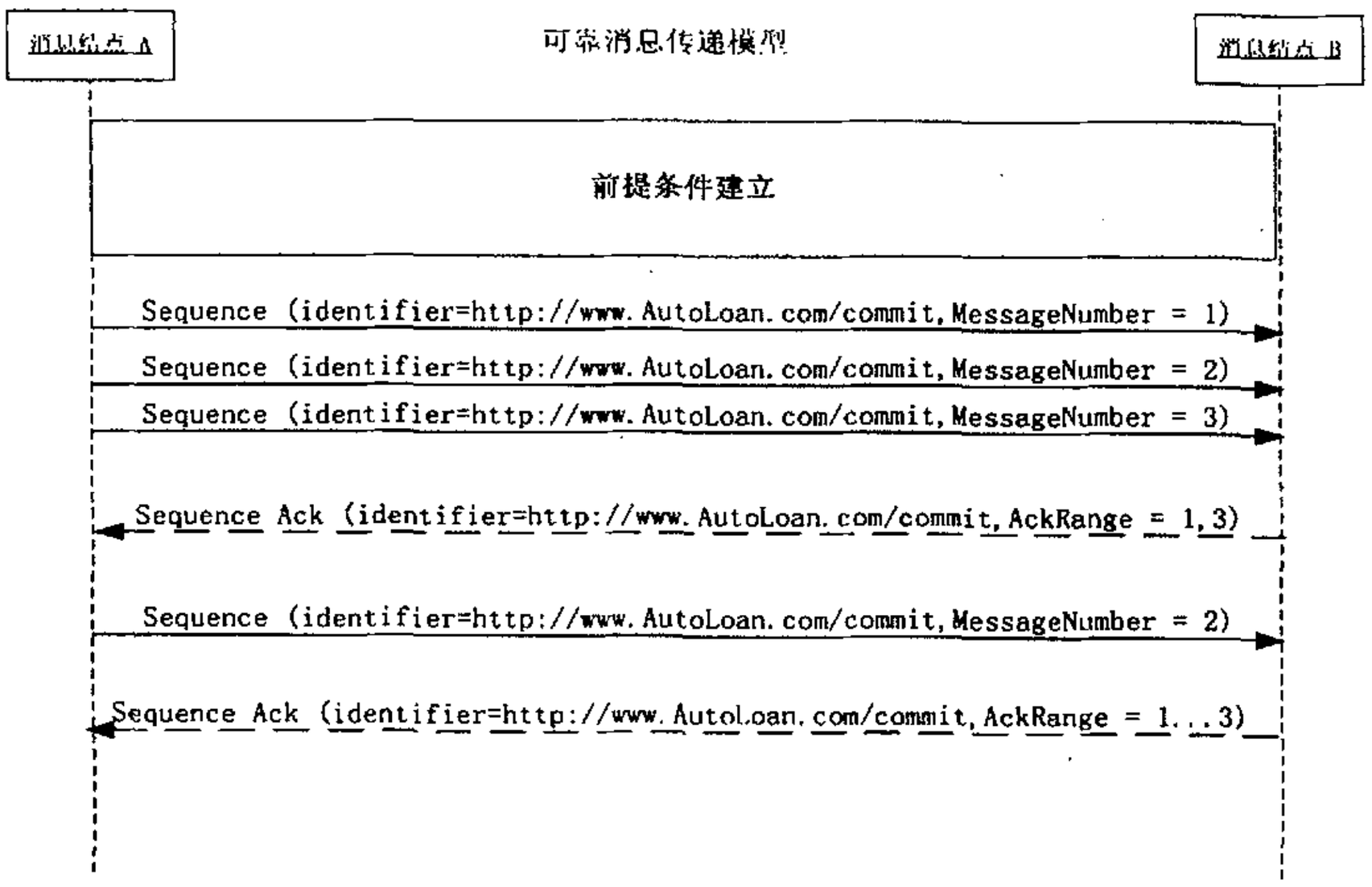

图 3.6 可靠消息传递示例

1. 首先，协议的前提条件建立，这包括策略交换、信任度的建立；
2. 消息源建立新的消息序列，在其中加入三个从 1 开始标号的消息，之后源结点发送出三条消息；
3. 在最后的第三条消息发送之后，源在序列末尾加上结束的消息标识；
4. 第二条消息在传输的过程中丢失；
5. 目的地确认收到了消息 1 和 3；
6. 源重传第二条消息，此时是在一个新的底层链接上传递消息，但由于有着相同的序列号和消息标识，所以目的地可以确认这是早期丢失的消息，并加入成功收到的消息中；
7. 源在消息体中有确认元素，所以消息目的地在其中加入确认消息，返回给源结点；
8. 目的地收到重新传递的消息标识为 2 的消息，并发送确认消息，范围从 1 至 3；源接收到确认消息，知道序列发送成功。

§3.3 本章小结

本章以前一章中对异步消息机制和可靠 Web 服务的研究，分别就异步性和可靠性给出了设计思路。在异步消息传输和处理机制中，分别给出了失效异步的四种设计模式；在可靠方面，结合 Web 服务的实际情况，提出了可靠 Web 服务的参考模型。这些设计思路，为下一章的详细设计提供了基础。

第四章 异步可靠 Web 服务的实现

本章将详细介绍异步可靠 Web 服务的实现技术，包括面向消息的中间件（Message Oriented Middleware）与实现异步可靠 Web 服务之间的关系，实现系统的总体结构以及同步和异步各自的实现细节。

§4.1 系统总体结构

系统的总体结构可以划分为两大子系统：SJIT（SOAP-JMS Inter-Transformer）、AM（Asynchronous Messaging）。SJIT 负责在 SOAP 消息与 JMS 消息之间进行互译；AM 则主

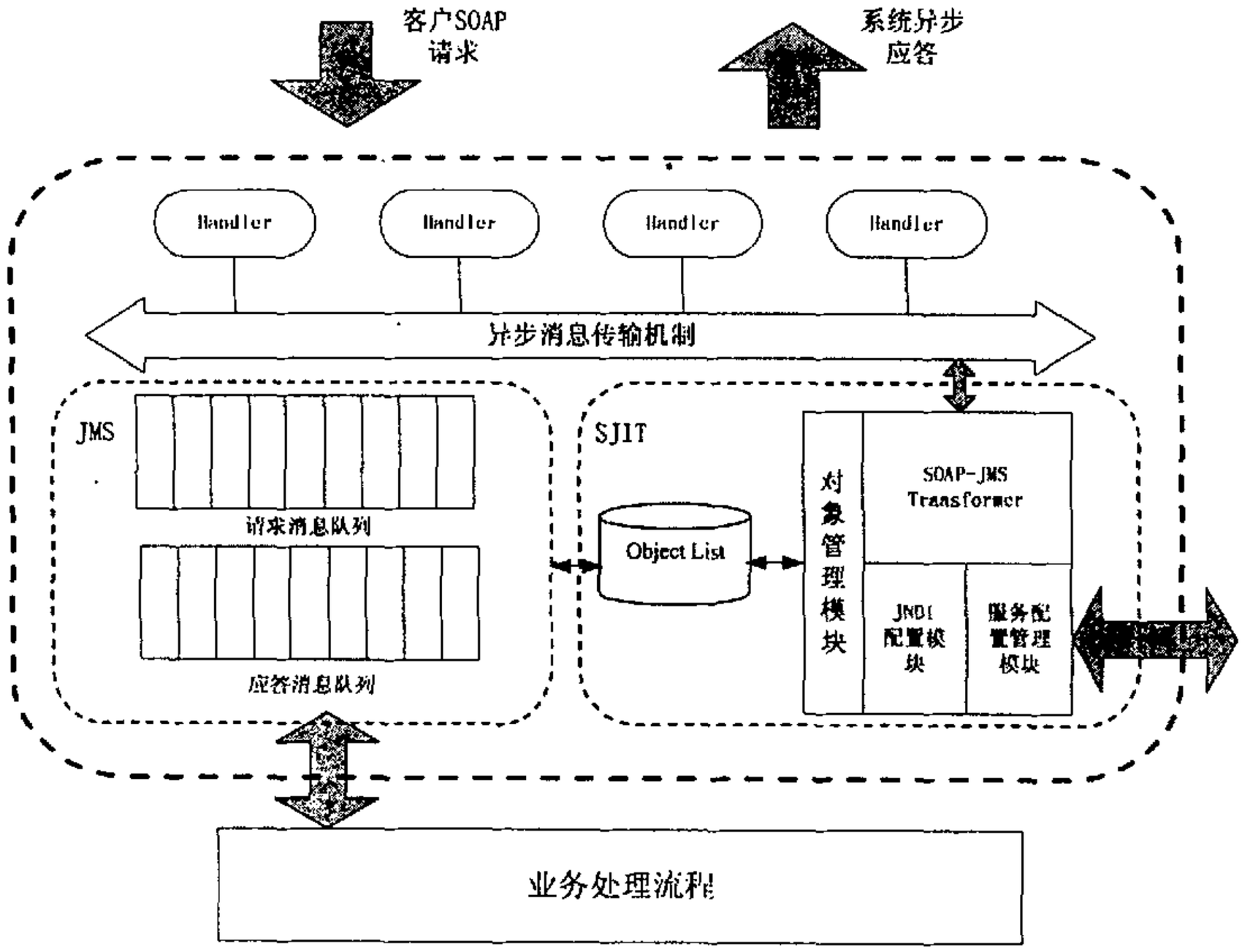

图 4.1 系统总体结构

要完成消息的异步传输。两者在逻辑实现上和物理实现上都是紧密相连的，与 JMS 一起向外完整的提供异步可靠的 Web 服务。

后面的几个小节将分别就这两个子系统和辅助模块的对象模型、结构功能、接口说明、逻辑流程、关键技术等方面进行详细的叙述。

§4.2 异步消息处理功能的实现

根据第二章中阐述的有关异步 Web 服务的实现问题，本小节将分别就异步回调和异步轮训两种机制下一一给出设计和实现细节。

4.2.1 异步回调的实现

4.2.1.1 相关标识与交换机制

由于异步场景下，客户提交的请求与系统返回的应答不在一个网络链接或者事务中，因此，需要建立一套将不同客户的请求与应答相关和交换的机制。

这里相关标识和交换机制利用的是 JMS 消息头域中的 JMSCorrelationID 域。系统使用 JMSCorrelationID 将请求消息于应答消息进行关联。由于系统在接收每一个 JMS 消息时，会指定一个唯一的 ID 号，所以利用该 ID 号，可以将请求和应答关联起来。示例代码段如下：

```
if (inMessage instanceof TextMessage) {
    msg = (TextMessage) inMessage;
    System.out.println(" ReplyMsgBean: " +
                       "Received message: " + msg.getText());
    tc = tcf.createTopicConnection();
    ts = tc.createTopicSession(true, 0);
    tp = ts.createPublisher((Topic) msg.getJMSReplyTo());
    replyMsg = ts.createTextMessage("ReplyMsgBean " +
                "processed message: " + msg.getText());
    replyMsg.setJMSCorrelationID(msg.getJMSMessageID());
    replyMsg.setIntProperty("id", msg.getIntProperty("id"));
    tp.publish(replyMsg);
    tc.close();
} else
{
    System.err.println("Message of wrong type: " + inMessage.getClass().getName());
```

4.2.1.2 回调地址

在同步交互模式下，Web 服务的客户端发送的 SOAP 消息中不会带有回调地址，只需要处于阻塞方式在一个 HTTP 链接中等待服务方的应答消息或者失败消息。而在异步方式

下，客户方请求与服务方应答出于不同的链接中。为了服务方返回应答消息，客户方需要确认服务方收到了含有回调地址的约定格式的 SOAP 消息。之后，服务方在处理完请求返回应答的过程中，自身充当一个普通的 Web 服务客户方，调用原来客户方提供的回调地址，返回应答消息。过程如下图所示：

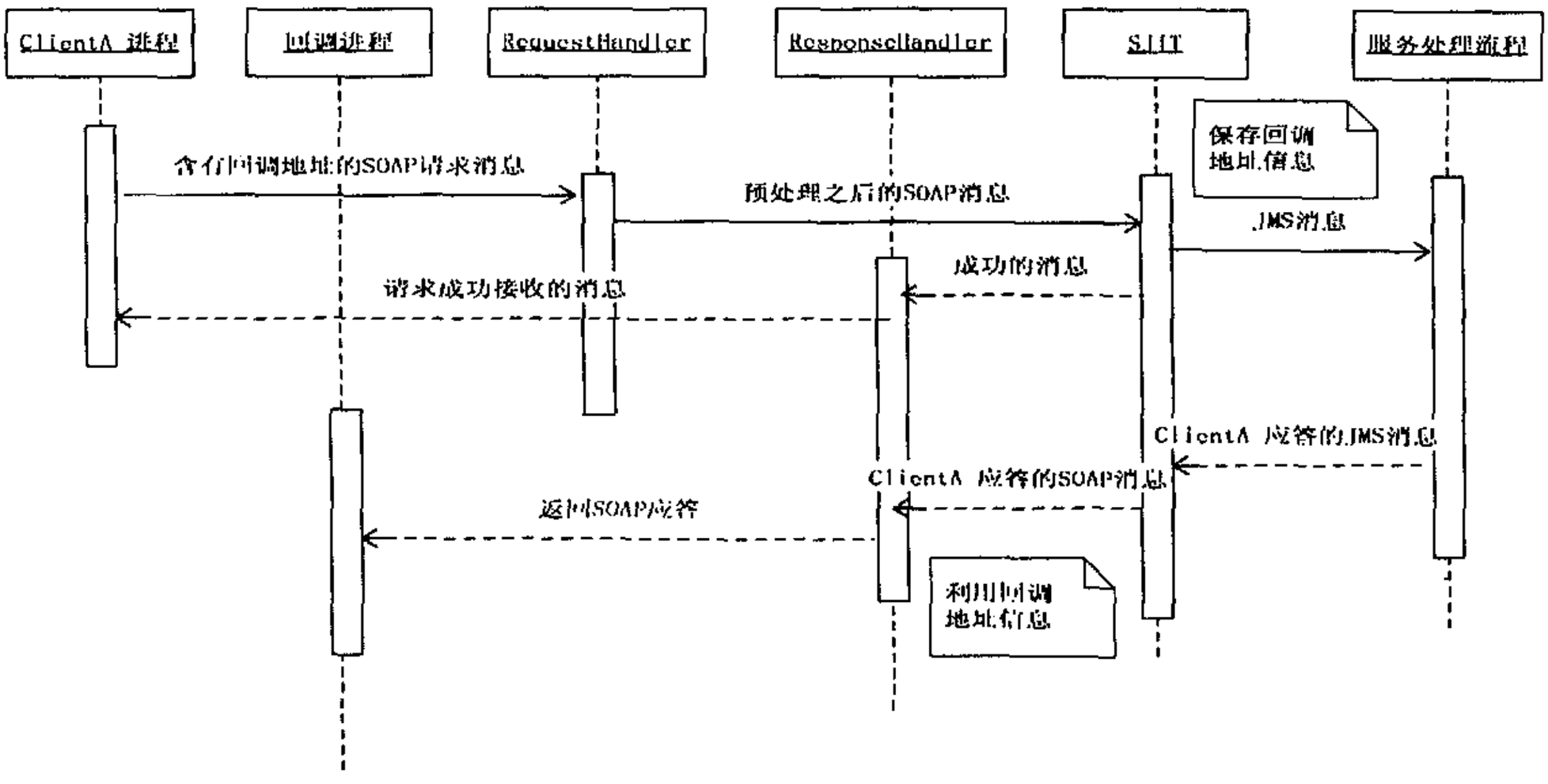

图 4.2 回调时序图

4.2.1.3 请求与应答的划分

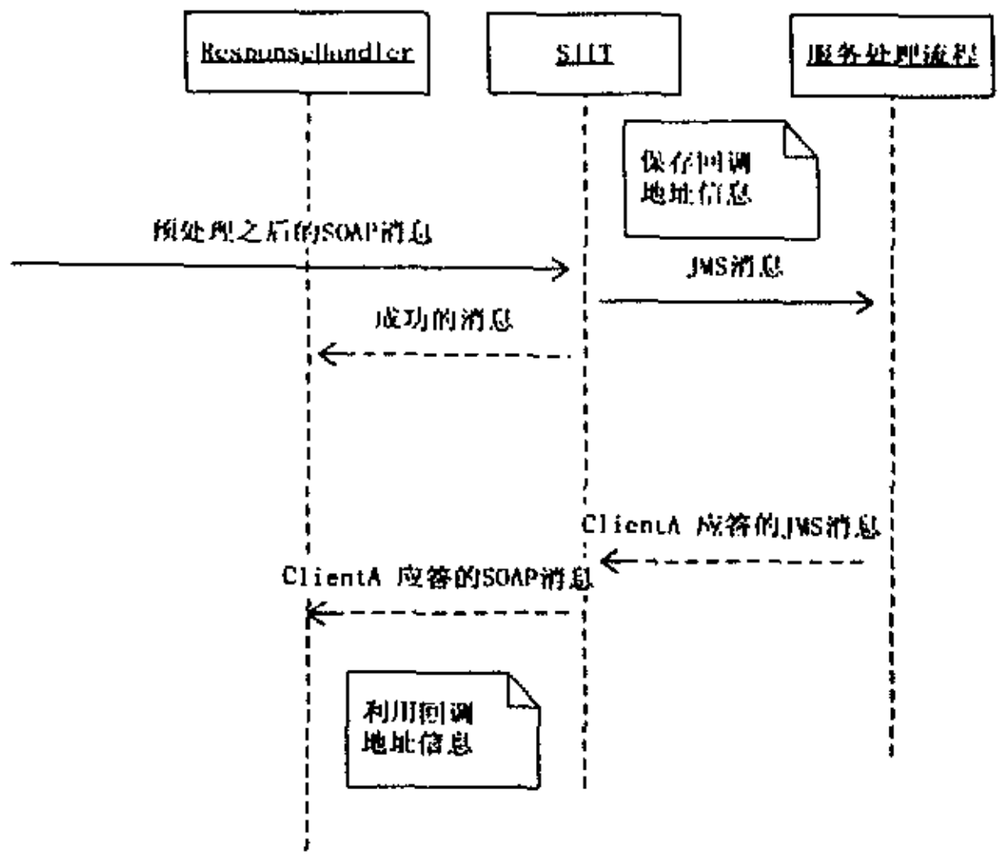

图 4.3 请求/应答划分时序图

将请求与应答划分开来的主要是 SJIT 部件，其逻辑流程如下：

1. 将 SOAP 消息转换为 JMS 消息，保存回调地址于 ObjectList 中；
2. 利用新生成的 JMS 客户，将 JMS 消息发送给 JMS，等待确认消息；
3. 若成功，则同步返回一个成功的 SOAP 应答，否则同步返回一个失败信息；

这里，将请求链接和应答链接一分为二。

1. 在取得应答之后，再利用 SJIT 转换 JMS 消息为 SOAP 消息；
2. 创建一个新的 SOAP 客户，从 SJIT 中检索该应答消息的回调接口；
3. 利用该回调接口，产生一个新的 SOAP 请求链接，向回调接口以 one-way 的方式返回应答消息；

4.2.1.4 客户对异步响应的处理方式

根据回调模式，这里我们创建了一个 ReplyHandler 对象，该对象在 SOAP 客户启动之后被隐含的产生，并监听回调接口是否有应答消息。该类图如下：

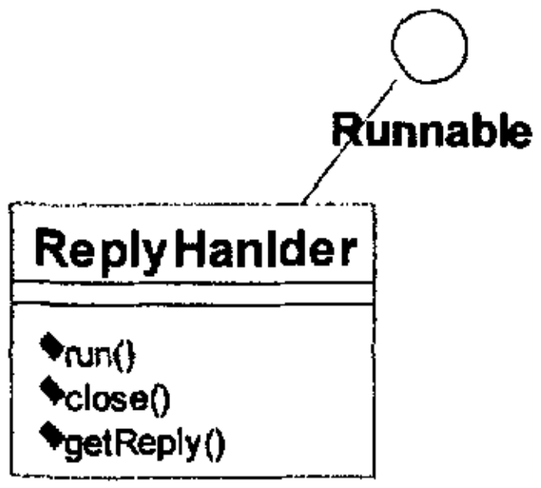

图 4.4 ReplyHandler 类图

值得注意的是，在可以有效分割的前提下，客户端对应答的处理可以放入 ReplyHandler 对象中，这样，请求逻辑和应答处理逻辑被清晰的划分开来。

4.2.1.5 请求与应答的关联

在这里，使用 JMS 中提供的临时队列机制 ，可以有效的将客户的请求与应答关联起来，如下图所示：

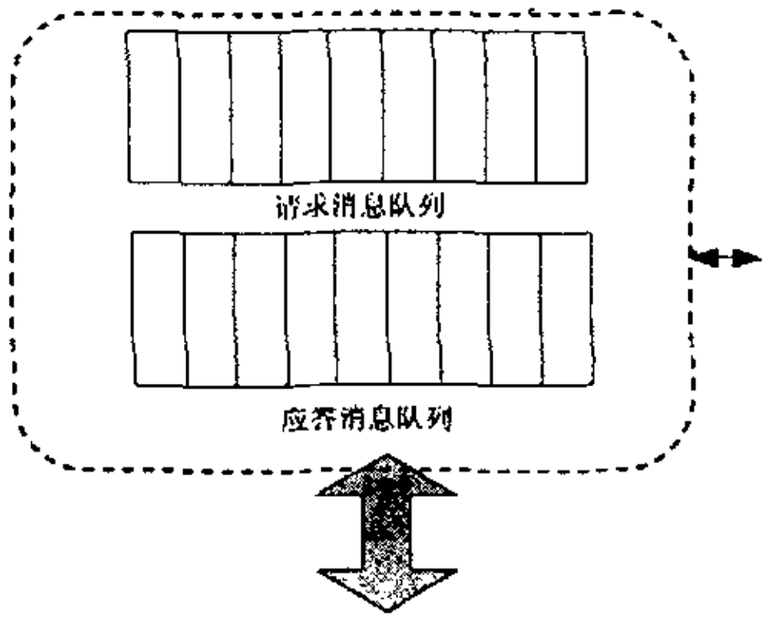

图 4.5 请求/应答关联图

关联逻辑流程如下：

1. 在 JMS 中，每次有一个客户 SOAP 请求，经过 SJIT 转换为对应的 JMS 消息之后，生成新的 JMS 客户，并向 ObjectList 中登记新的客户和回调地址；
2. 新的 JMS 客户向 JMS 目的地队列申请创建链接；
3. 该 JMS 客户产生一个根据该链接的临时队列，并登记为该链接的返回队列；
4. 向 JMS 发送完 JMS 消息之后，该客户出于活跃期，随时等待收到来自临时队列的应答消息；
5. 客户一旦从临时队列中取得应答消息，马上调用 SJIT 将 JMS 格式的应答消息转换为 SOAP 消息，同时生成新的 SOAP 客户；
6. 新的 SOAP 客户利用原来 JMS 客户 ID 检索 ObjectList，获得该请求的回调地址，作为参数传递给新生产的 SOAP 客户；
7. 该 SOAP 客户完成回调过程；
8. ReplyHandler 监听回调接口，一旦有应答消息，则利用自身的逻辑完成对应答的处理。

与建立 JMS 客户，创建临时队列相关的代码段如下：

```
qConnection = queueConnectionFactory.createQueueConnection();
qSession = qConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
replyQueue = qSession.createTemporaryQueue();
qReceiver = qSession.createReceiver(replyQueue);
qReceiver.setMessageListener(new HRListener());
qConnection.start();
```

4.2.2 异步轮询的实现

从回调和轮询的实现策略来看，我们实现系统应最小化的区别两者，而将不同交给客户段实现。所以，与回调相比，轮询的区别只是在客户对异步应答的处理方式的不同。

4.2.2.1 客户对异步应答的处理方式

有些时候，开发人员需要能够像延迟同步模型那样，读取应答消息并进行相应处理。因此，在异步回调模型的基础上可以实现异步轮询模式。

我们的设计原则是：client 在发送请求消息之后就退出了，当 client 再次恢复时，异步轮询模式支持 client 通过轮询的方式取得应答消息。因此与回调不同点几种在 ReplyHandler 上，其对于客户端不再是透明的，而是需要提供查询接口，以客户轮询的方式提供是否有异步应答的信息或者应答消息的内容。

§4.3 可靠消息处理功能的实现

在现有 Web 服务规范集合中没有与可靠相关的规范情况下，利用 JMS 机制，可以在实现服务的企业内部或者发出调用的 Web 服务客户端实现一定程度的可靠性。

在 JMS 消息头域中有 JMSDeliveryMode 属性，该域有两种配置值：

- NON_PERSISTENT：是对可靠性要求最低的消息传输模式，在这种模式下，JMS 消息并不会存放到永久存储介质中，一旦 JMS 系统故障，则所有标志为 NON_PERSISTENT 传输模式的 JMS 消息都会被丢失；
- PERSISTENT：该域指出，JMS 系统即使在故障时，也必须 JMS 消息在传输的过程中是可靠的，也就是在必要的时候对 JMS 消息在永久存储介质上进行备份；

对 NON_PERSISTENT 标志的消息，JMS 系统保证该消息最多被发送一次，也就是说，消息传输过程中可能丢失，但保证不会被发送多次；

对 PERSISTENT 标志的消息，JMS 系统保证该消息发送一次且只有一次，这意味着即使 JMS 系统发生故障，也要保证 JMS 消息不被丢失，并且不会被发送多次。

示例代码段如下：

```
topicPublisher. publish (message, DeliveryMode. NON_PERSISTENT, 3, 10000) ;
```

§4.4 SOAP2JMS 功能的实现

由于整个系统的输入是同步的 SOAP 协议，而为了利用后端的 JMS 系统提供的种种特性，让服务的真正实现者——服务处理流程——可以以异步可靠的方式处理请求消息，需要子系统 SJIT 在同步的 SOAP 消息和异步的 JMS 消息之间进行互译，而且，这个过程对于 Web 服务的客户端而言是透明的，而对于利用服务配置模块进行管理的服务管理员来说，则是可选择的。从图 4.2 中可以看出，SJIT 包含 SOAP-JMS Transformer、服务配置

模块、对象管理模块和 Object List 、JNDI 适配模块，下面将展开叙述。

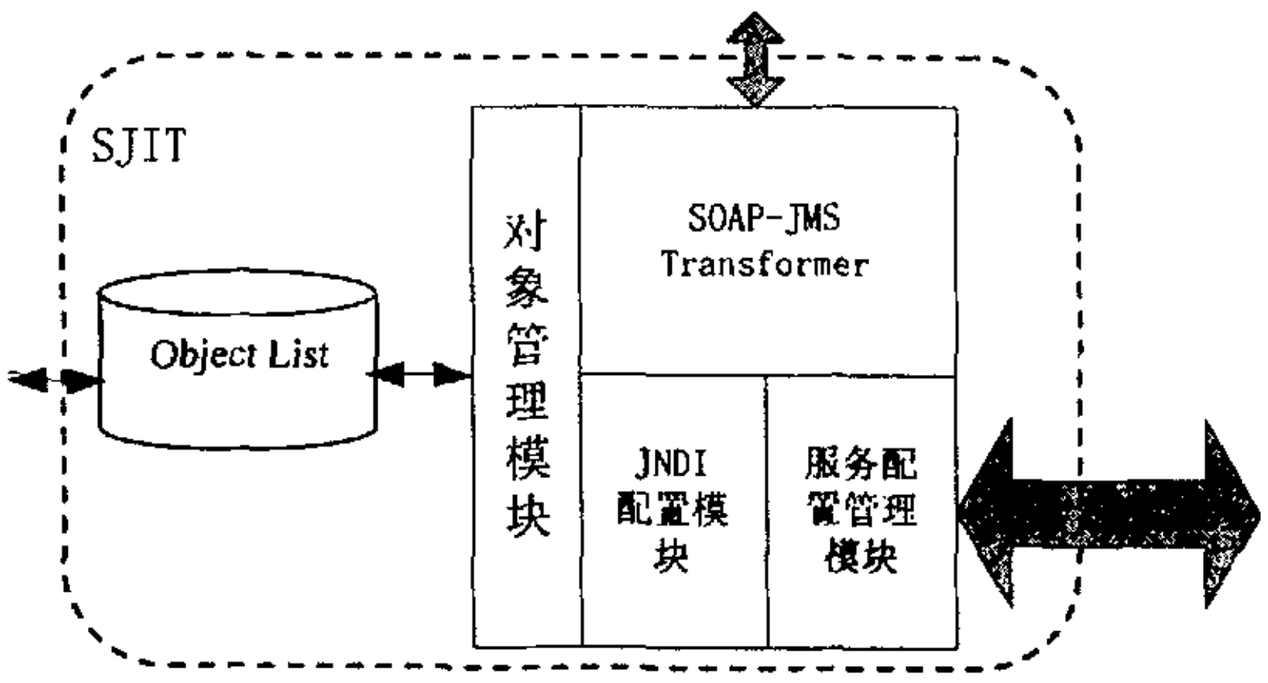

图 4.6 SJIT 系统结构

4.4.1 SOAP-JMS Transformer

由于 SOAP 与 JMS 是两种不同的协议，SOAP-JMS 需要在两种协议之间完成以下的映射关系：

- 如何指定 JMS 消息发送的目的地；
- 使用哪种 JMS 消息类型，将 SOAP 消息的有效载荷转换为 JMS 消息；

如何异步的将 JMS 消息转换为 SOAP 消息返回；

4.4.1.1 JMS 目的地址的映射

可能有以下多种方式决定如何指定 SOAP 请求的 JMS 目的地：

- 在 WSDL 中指定；
- HTTP 头中的某个子元素或者 SOAPAction 元素指定；
- 在 SOAP Header 中指定；
- 基于内容的路由；

在这一层进行 SOAP-JMS 消息的转换方法多种多样。最简单的做法是将整个 SOAP Envelope 直接当作一个 JMS 的 TextMessage 或者 ByteMessage 不过这要求消息的消费者知道如何解析并理解 SOAP Envelope 的内容。当然，还可以有其他的做法，例如，可以将 HTTP header 或者 SOAP headers 中的内容放入 JMS 消息的属性域中，然后将 SOAP 请求的每一个子元素放入 JMS 消息的有效载荷中。还有一种最纯粹的方式，完全将 SOAP envelope 中的内容转换为 JMS 消息，这样做的好处，使得 JMS 的消费者可以不必知道任何有关 SOAP 的知识。

4.4.1.2 SOAP 有效载荷的互译

对于 SOAP envelope 中的内容，应该转换成何种 JMS 类型的消息，最直接和自然的方式是将整个 SOAP envelope 中的有效载荷转换为 JMS TextMessage。由于 JMS TextMessage 消息的设计初衷，就是为了传递 XML 格式的消息，因此这种转换方式最为简单。但这种转换的前提条件是，消息的消费者自己能够完全处理 XML 文档的所有内容。如果需要根据 SOAP envelope 的内容指定 JMS 消息的属性，则需要 SOAP-JMS 桥接系统进行预处理，将 SOAP envelope 根据其子元素逐一解析，将相关的元素指定为 JMS 消息的某一个属性，然后将 SOAP envelope 中的消息内容转换为 JMS 消息的消息体。这种方式要求 SOAP-JMS 桥接系统对 SOAP 请求的内容进行完全的处理，使得 JMS 消息的消费者完全不需要知道任何有关 SOAP 的知识，它看到的只是一个普通的 JMS 消息。

4.4.2 服务配置管理模块

本小节概括介绍服务配置管理模块的功能、在系统中的地位、组成部分及基本原理。

4.4.2.1 功能

服务器管理模块是系统中最重要的模块之一，负责管理服务器上部署的服务以及整个系统的系统配置。具体描述如下：

- 服务管理

  服务管理模块负责管理各个服务的服务描述符（Web Services Deployment Descriptor），这些信息在部署一个新服务时被加入到系统中，并在该服务被去部署时从系统中删除；在运行时刻，对某服务的请求将依赖于这些部署信息进行参数编解码，并最终投递到正确的后端系统（JMS 中指定的请求队列）。

- 系统配置

  服务管理器模块还负责管理整个系统各个部件的配置信息。系统初启时，系统按照配置信息设置属性、加载模块；系统运行期间，也可根据用户请求动态修改系统配置。系统配置信息出一个 XML 格式的配置文件描述，从而使得 SOAP 服务器成为一个可配置、可插拔的灵活的体系结构。

4.4.2.2 地位

从一次请求处理的过程来看服务管理模块在系统中所处的位置及与其它模块的关系：

- 根据目标服务 ID，向服务管理模块查询获得目标服务的部署描述符，部署描述符中记录了该服务的 provider 类型；
- 根据 provider 类型，服务管理模块将载入正确的 provider 实例，从而解析、激活 Object

4.4.1.2 SOAP 有效载荷的互译

对于 SOAP envelope 中的内容，应该转换成何种 JMS 类型的消息，最直接和自然的方式是将整个 SOAP envelope 中的有效载荷转换为 JMS TextMessage。由于 JMS TextMessage 消息的设计初衷，就是为了传递 XML 格式的消息，因此这种转换方式最为简单。但这种转换的前提条件是，消息的消费者自己能够完全处理 XML 文档的所有内容。如果需要根据 SOAP envelope 的内容指定 JMS 消息的属性，则需要 SOAP-JMS 桥接系统进行预处理，将 SOAP envelope 根据其子元素逐一解析，将相关的元素指定为 JMS 消息的某一个属性，然后将 SOAP envelope 中的消息内容转换为 JMS 消息的消息体。这种方式要求 SOAP-JMS 桥接系统对 SOAP 请求的内容进行完全的处理，使得 JMS 消息的消费者完全不需要知道任何有关 SOAP 的知识，它看到的只是一个普通的 JMS 消息。

4.4.2 服务配置管理模块

本小节概括介绍服务配置管理模块的功能、在系统中的地位、组成部分及基本原理。

4.4.2.1 功能

服务器管理模块是系统中最重要的模块之一，负责管理服务器上部署的服务以及整个系统的系统配置。具体描述如下：

- 服务管理

  服务管理模块负责管理各个服务的服务描述符（Web Services Deployment Descriptor），这些信息在部署一个新服务时被加入到系统中，并在该服务被去部署时从系统中删除；在运行时刻，对某服务的请求将依赖于这些部署信息进行参数编解码，并最终投递到正确的后端系统（JMS 中指定的请求队列）。

- 系统配置

  服务管理器模块还负责管理整个系统各个部件的配置信息。系统初启时，系统按照配置信息设置属性、加载模块；系统运行期间，也可根据用户请求动态修改系统配置。系统配置信息由一个 XML 格式的配置文件描述，从而使得 SOAP 服务器成为一个可配置、可插拔的灵活的体系结构。

4.4.2.2 地位

从一次请求处理的过程来看服务管理模块在系统中所处的位置及与其它模块的关系：

- 根据目标服务 ID，向服务管理模块查询获得目标服务的部署描述符，部署描述符中记录了该服务的 provider 类型；
- 根据 provider 类型，服务管理模块将载入正确的 provider 实例，从而解析、激活 Object

List 中的对象；

- SOAP 消息经过 Transformer 的翻译之后，将 JMS 消息根据服务管理模块中指定的消息请求队列名，投递到 JMS 中对应的队列。

此外，服务管理模块还将提供一个预定义的服务作为系统的管理接口，用户可以通过部署配置工具访问该服务，从而部署、去部署、查看其它服务，以及进行系统动态配置。

4.4.2.3 组成

服务器管理模块主要由以下几部分（类）组成：

- 服务管理器（ServiceManager）：服务管理模块的总控部件；
- 配置管理器（ConfigManager 等）：管理服务的部署描述符；
- Provider 映射器（ProviderMapper）：管理系统可插拔的 Provider 组件；
- 辅助工具：服务管理模块的辅助工具类；
- 部署配置工具（ServiceManagerClient）：用户访问服务管理模块的接口

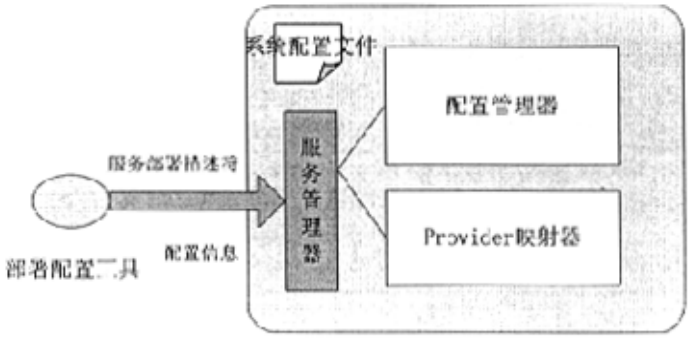

图 4.7 管理模块

4.4.2.4 异步可靠 Web 服务的实现类型

为了了解异步可靠 Web 服务实现类型的划分，我们先来看看 SOAP 消息模型。

- SOAP 消息模型

  带有 XML 语义的 SOAP 消息结构是比较简单的，它有一个 envelope 组成，其中包括：
  - 可选的 header 元素，其中可以包含零个或多个 header entries；
  - 一个 body 元素，包含一个或多个 body entries；
  - 零个或多个可附加的，非标准的元素。

  所以，SOAP 消息模型如下图所示：

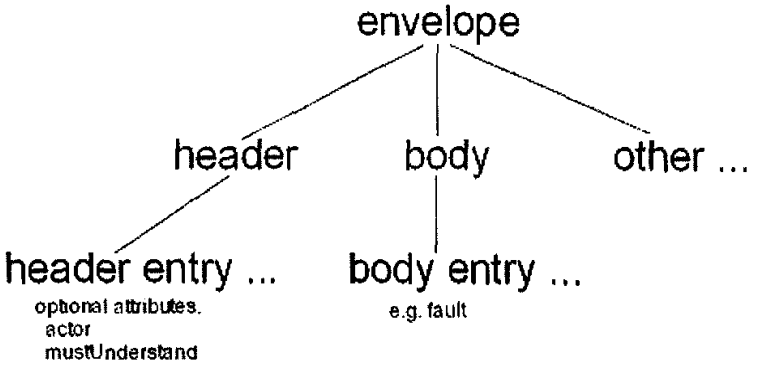

图 4.8 SOAP 消息组成

所有代表 SOAP 消息类全都继承自 MessageElement 类，该类负责处理与 SOAP 名空间和编码方式有关的信息，整个类图如下：

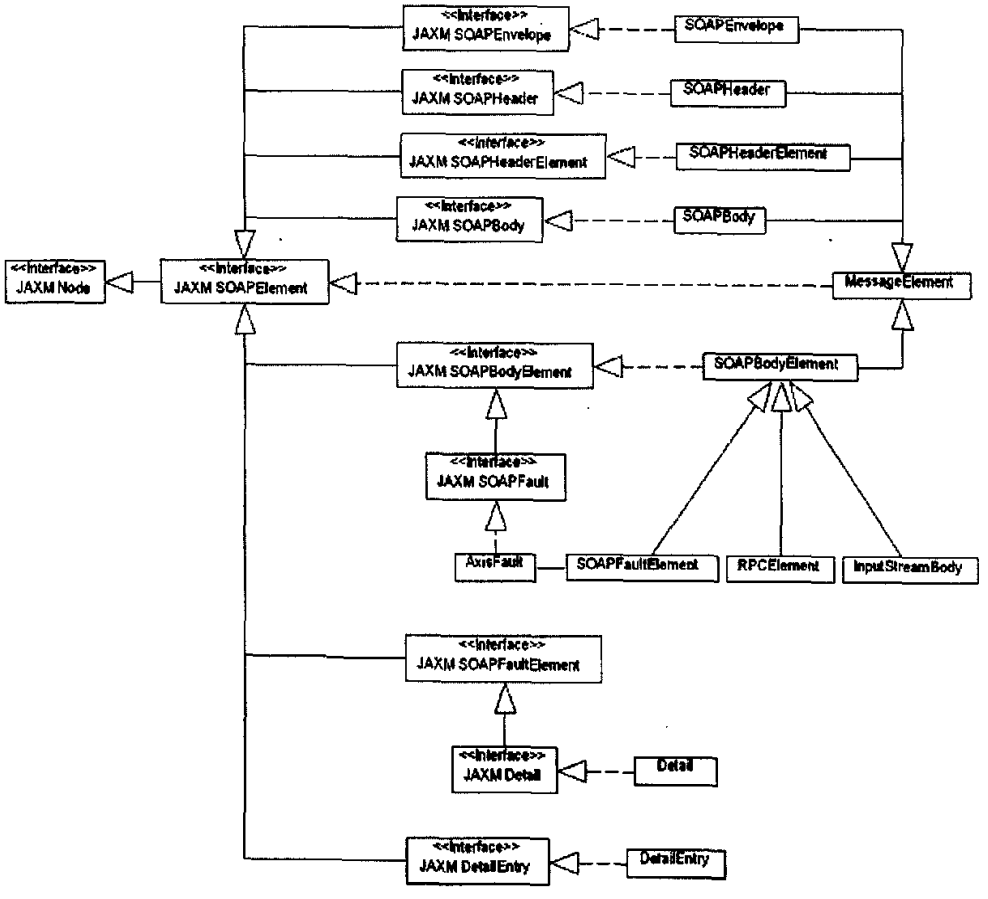

图 4.9 消息类图

所有类从 SOAP 消息类的角度来看，关系如下：

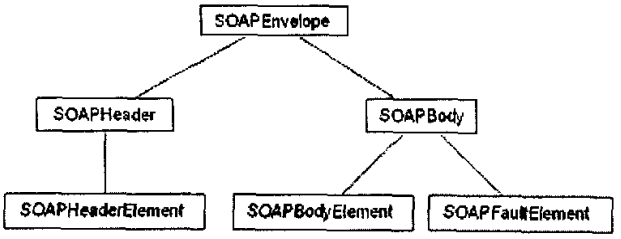

图 4.10 SOAP 消息类图

- 异步可靠服务类型

  基于消息的异步可靠 Web 服务可以根据 SOAP 消息的解析粒度划分为以下四种类型：

  - Element：这种服务在进行 SOAP 消息的解码时，将 SOAP 消息中的每个子元素当作一个 Element，并提取其中的有效载荷，放入产生的 JMS 消息体中的每个子域中。这种类型的服务认为 JMS 消息的处理流程需要的消息粒度达到了对应的 SOAP 消息体中每一个 Element；
  - SOAPBodyElement：这种服务在进行 SOAP 消息的解码时，将消息 body 中的每个子元素当作一个 SOAPBodyElement，并提取其中的有效载荷，放入产生的 JMS 消息体中的每个子域中。这种类型的服务认为 JMS 消息的处理流程需要的消息粒度达到了对应的 SOAP 消息体中每一个 SOAPBodyElement；
  - SOAPEnvelope：这种服务在进行 SOAP 消息的解码时，将整个 SOAPEnvelope 消息当作一个 JMS 消息，并提取其中的有效载荷，放入产生的 JMS 消息体中。这种类型的服务认为 JMS 消息的处理流程需要的消息粒度达到了对应的整个 SOAP SOAPEnvelope；
  - Document：这种服务在进行 SOAP 消息的解码时，将整个 SOAP 消息当作一个 Document，并提取其中的有效载荷，放入产生的 JMS 消息体中的每个子域中。这种类型的服务认为 JMS 消息的处理流程需要的消息粒度达到了对应的整个 SOAP 消息；

值得注意的是，这四种服务的粒度由小到大，并且除了 Document 类型的服务之外，其他类型的服务都需要进行 XML<->Java 的类型和数据映射。

在实现中，系统提供了这四种服务的默认实现类，他们分别是：processElementsMessageService、processSoapBodyElementMessageService、processSoapEnvelopeMessageService、processDocumentMessageSerivce，用户在部署服务的时候，只需要选择这四种服务其中的一种即可。

4.4.2.5 配置文件

系统配置文件用于配置系统服务，配置文件用 XML 格式书写。系统在初始化时将使用 XML 解析器从该配置文件中获得配置信息，从而实现系统的正确配置。配置文件主要包含三部分配置内容：

- 服务名称和类型（Message 类型）；
- 服务预处理配置的 Handler；
- 参数列表。

下面是一个配置文件的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="processElementsMessageService" provider="java:MSG">
  <requestFlow>
   <handler
type="java:org.apache.axis.components.bridge.SOAPJMS.SOAPJMSHeaderHandler"/
>
  </requestFlow>
  <parameter name="allowedMethods" value="processElements"/>
  <parameter name="JMSDestinationName" value="queue"/>
  <parameter name="jndiproperties" value="jndi-connection-factory.properties"/>
  <parameter name="className"
value="org.apache.axis.components.bridge.SOAPJMS.processElementsMessageService
"/>
  <parameter name="JMSMessageType" value="TextMessage"/>
 </service>
 <service name="processDocumentMessageSerivce" provider="java:MSG">
  <requestFlow>
   <handler
type="java:org.apache.axis.components.bridge.SOAPJMS.SOAPJMSHeaderHandler"/
>
  </requestFlow>
 <parameter name="allowedMethods" value="processDocument"/>
  <parameter name="JMSDestinationName" value="MyTopic"/>
  <parameter name="jndiproperties" value="new-jndi.properties"/>
  <parameter name="className"
```

```
  <parameter name="JMSMessageType" value="TextMessage"/>
 </service>

<service name="processSoapBodyElementMessageService" provider="java:MSG">
  <requestFlow>
   <handler
type="java:org.apache.axis.components.bridge.SOAPJMS.SOAPJMSHeaderHandler"/
>
  </requestFlow>
  <parameter name="allowedMethods" value="processSoapBodyElement"/>
  <parameter name="JMSDestinationName" value="MyTopic"/>
  <parameter name="jndiproperties" value="new-jndi.properties"/>
  <parameter name="className"
value="org.apache.axis.components.bridge.SOAPJMS.processSoapBodyElementMessa
geService"/>.
  <parameter name="JMSMessageType" value="TextMessage"/>
</service>
<service name="processSoapEnvelopeMessageService" provider="java:MSG">
  <parameter name="allowedMethods" value="processsoapEnvelope"/>
  <parameter name="JMSDestinationName" value="MyTopic"/>
  <parameter name="jndiproperties" value="new-jndi.properties"/>
  <parameter name="className"
value="org.apache.axis.components.bridge.SOAPJMS.processSoapEnvelopeMessageSe
rvice"/>
  <parameter name="JMSMessageType" value="TextMessage"/>
 </service>
 <service name="urn:xmltoday-delayed-quotes" provider="java:RPC">
  <parameter name="allowedMethods" value="getQuote"/>
  <parameter name="className" value="samples.stock.StockQuoteService"/>
 </service>
</deployment>
```

服务器管理模块是系统中最重要的模块之一，负责管理服务器上部署的服务以及整个系统的系统配置。具体描述如下：

- 服务管理

  服务管理模块负责管理各个服务的服务描述符（Web Services Deployment Descriptor），这些信息在部署一个新服务时被加入到系统中，并在该服务被去部署时从系统中删除；

在运行时刻，对某服务的请求将依赖于这些部署信息进行参数编解码，并最终投递到正确的后端系统（JMS 中指定的请求队列）。

- 系统配置

服务管理器模块还负责管理整个系统各个部件的配置信息。系统初启时，系统按照配置信息设置属性、加载模块；系统运行期间，也可根据用户请求动态修改系统配置。系统配置信息由一个 XML 格式的配置文件描述，从而使得 SOAP 服务器成为一个可配置、可插拔的灵活的体系结构。

4.4.3 对象管理模块和 Object List

SOAP 消息经过翻译之后，成为相应的 JMS 消息等待传递至 JMS，这时对象管理模块会根据这个新 JMS 消息和服务管理模块中提供的消息队列名称和 JNDI 文件，在名空间中查找有关绑定的消息队列信息，并生成一个新的 JMS 客户对象和存放应答消息的临时队列，将该对象加入至 Object List 中，之后，利用该对象向 JMS 指定的队列发送 JMS 消息。对象管理模块还需要对 Object List 进行一定的维护，比如定义对象的生存时间，在指定的时间内没有从临时应答队列中读取到应答消息，则异步的返回给 SOAP 客户一个失败信息，然后从 Object List 中销毁该对象。

4.4.4 JNDI 适配模块

该模块的相关信息放在服务配置管理模块中指定的 jndi. properties 文件中，该文件与具体的 JMS 产品相关，主要是配置 JNDI 所需要的初始化上下文 initial context、绑定消息请求队列等，以下是一个 jndi. properties 的示例：

JNDI 适配模块类图如下：

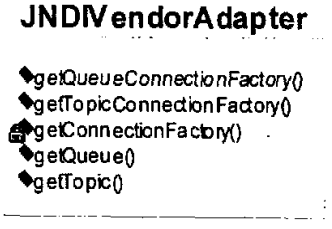

图 4.11 JNDI 适配模块类图

该模块负责查询 JNDI，并获得链接工厂和队列、主题。

§4.5 本章小结

本章从总体结构、对象模型、功能说明、接口说明、逻辑流程等几个方面介绍了 SJIT 和异步传输系统的实现。在系统实现中，我们还遇到了许多困难，但是经过我们对问题的仔细推敲和对 StarWebAdaptor 平台的深入了解，这些困难终于被我们所克服了。应该说，

在运行时刻，对某服务的请求将依赖于这些部署信息进行参数编解码，并最终投递到正确的后端系统（JMS 中指定的请求队列）。

- 系统配置

服务管理器模块还负责管理整个系统各个部件的配置信息。系统初启时，系统按照配置信息设置属性、加载模块；系统运行期间，也可根据用户请求动态修改系统配置。系统配置信息由一个 XML 格式的配置文件描述，从而使得 SOAP 服务器成为一个可配置、可插拔的灵活的体系结构。

4.4.3 对象管理模块和 Object List

SOAP 消息经过翻译之后，成为相应的 JMS 消息等待传递至 JMS，这时对象管理模块会根据这个新 JMS 消息和服务管理模块中提供的消息队列名称和 JNDI 文件，在名空间中查找有关绑定的消息队列信息，并生成一个新的 JMS 客户对象和存放应答消息的临时队列，将该对象加入至 Object List 中，之后，利用该对象向 JMS 指定的队列发送 JMS 消息。对象管理模块还需要对 Object List 进行一定的维护，比如定义对象的生存时间，在指定的时间内没有从临时应答队列中读取到应答消息，则异步的返回给 SOAP 客户一个失败信息，然后从 Object List 中销毁该对象。

4.4.4 JNDI 适配模块

该模块的相关信息放在服务配置管理模块中指定的 jndi. properties 文件中，该文件与具体的 JMS 产品相关，主要是配置 JNDI 所需要的初始化上下文 initial context、绑定消息请求队列等，以下是一个 jndi. properties 的示例：

JNDI 适配模块类图如下：

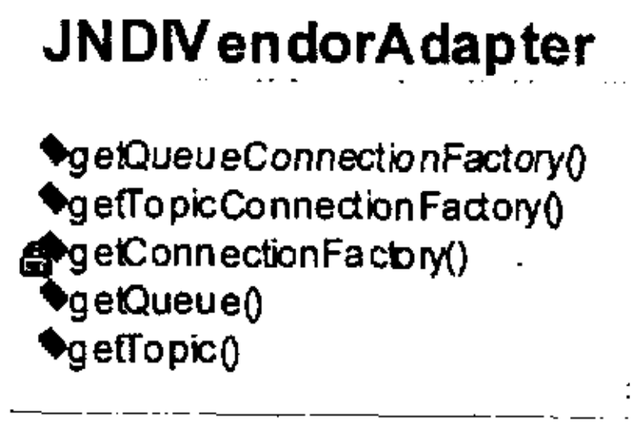

图 4.11 JNDI 适配模块类图

该模块负责查询 JNDI，并获得链接工厂和队列、主题。

§4.5 本章小结

本章从总体结构、对象模型、功能说明、接口说明、逻辑流程等几个方面介绍了 SJIT 和异步传输系统的实现。在系统实现中，我们还遇到了许多困难，但是经过我们对问题的仔细推敲和对 StarWebAdaptor 平台的深入了解，这些困难终于被我们所克服了。应该说，

我们在异步可靠服务方面进行了有价值的尝试。但是，随着 Web 服务技术的发展，还有许多问题等待我们去研究。

第五章 系统测试与结论

§5.1 测试的应用案例

为了对自己设计的系统性能有一个直观的了解，我们编写了一个简单但典型的应用作为系统的测试案例。请看下图：

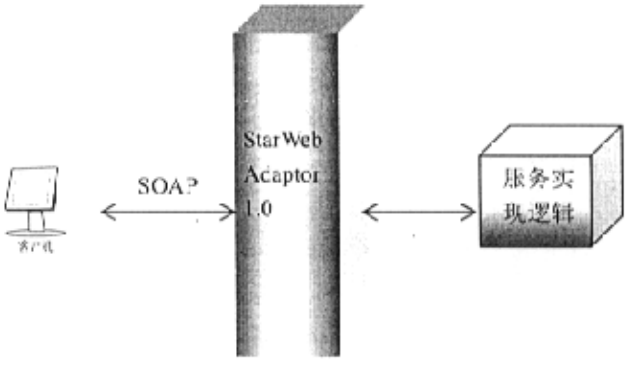

图 5.1 测试案例图

在此应用案例中，我们在 StarWebAdapter1.0 平台系统上部署了一个由 Java 实现的 Web 服务，该服务在收到 Web 服务客户的服务请求后，返回一个简单问候信息，此时，我们记录下客户请求的成功率作为测试指标，分别用同步调用方式和异步可靠调用方式实现该服务，特别在异步可靠模式下，平台将内嵌我们设计的 SJIT 系统，结构关系如下图：

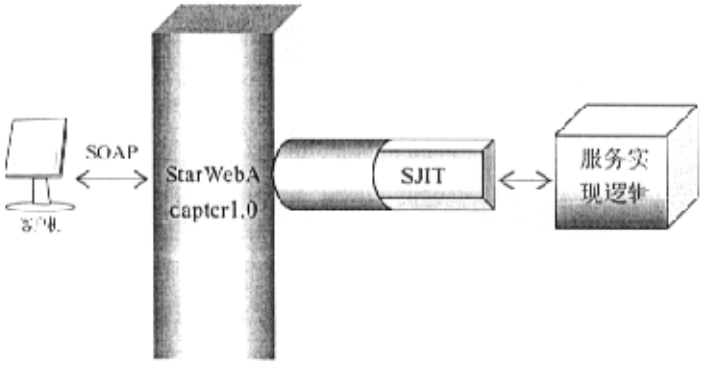

图 5.2 带有 SJIT 的测试案例图

测试中用 U 表示用户数(线程数)，R 表示请求个数，N 表示循环次数；例如，U5*R1*N20 表示模拟 5 个用户并发访问，每次发出一个请求，并且该过程循环 20 次。

我们在获得测试结果之前可以设想：在客户访问量较少时，由于需要 SJIT 系统进行桥接，因此第二种访问方式的响应时间（除非特别声明，所有涉及到的时间都以 ms 为单位）会较长；随着客户访问量的增加，由于异步操作的灵活性和可靠性的保证，客户请求的成功率会明显增加，而第一种方式的成功率将会有所降低。

§5.2 JMeter 与测试环境简介

5.5.2 JMeter 简介

Apache Jakarta 组织提供的完全由 Java 实现的 JMeter，是一种专门为 C/S 结构的应用提供测试的工具，与市场上使用的专业负载测试工具如 WinRunner 相比较，JMeter 有以下突出的特点：

- 免费使用；
- 使用简单。可以通过它功能丰富的 GUI 完成诸多的测试功能，能够满足系统在开发过程中的各种测试要求；
- 开源项目。因此用户还可以根据自己的喜好，对它进行个性的修改和特征修改。

JMeter 的执行主界面如下：

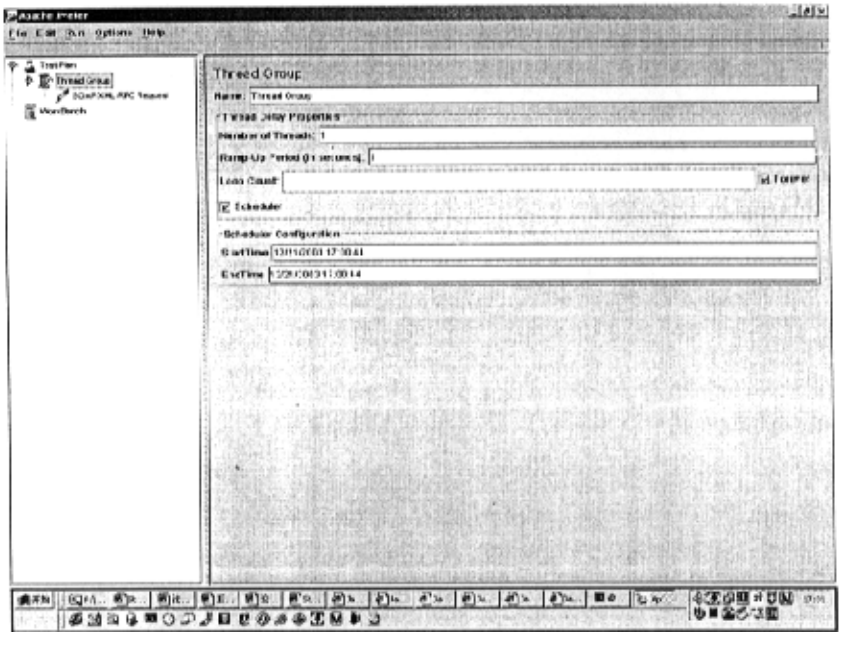

图 5.3 JMeter 的执行主界面

图 5.4 JMeter 执行 SOAP 调用的主界面

5.5.2 测试环境简介

- 操作系统：Windows Professional 2000；
- 硬件配置：CPU 2.0G、内存 512M；

客户端的配置：

- JMeter1.9(+Starws.jar+mail.jar+activation.jar)；
- JDK1.4；

服务端的配置：

- JDK1.3；
- StarWebAdapter1.0；
- SJIT(异步访问时需要)；

§5.3 同步方式的测试

在同步方式中，客户以同步的方式向服务方发送请求消息，在任意时刻，系统中的客户以时间的先后顺序获取服务访问的使用权。

在下表中，横坐标代表客户并发线程数（U 以 100 为单位，R1*N1），纵坐标代表每秒服务对于请求的失效率。

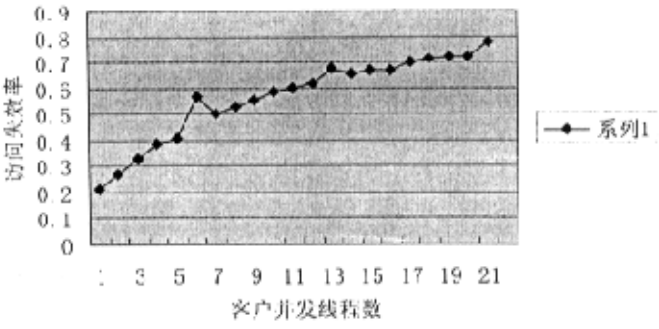

图 5.5 同步测试结果图

§5.4 异步方式的测试

在异步方式中，客户的访问请求会经过 SJIT 存储在 JMS 系统中的请求队列中，以等待目标服务的处理，在任意时刻，系统中的客户以时间的先后顺序获取服务访问的使用权。

在下表中，横坐标代表客户并发线程数（U 以 100 为单位，R1*N1），纵坐标代表每秒服务对于请求的失效率。

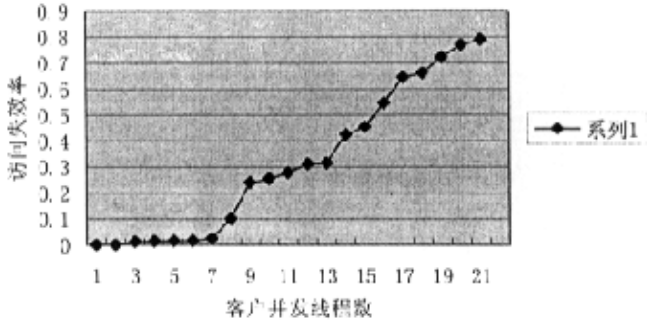

图 5.6 异步测试结果图

§5.5 分析与结论

将同步与异步调用的情况归结在同一张表上，可以产生出以下结论：

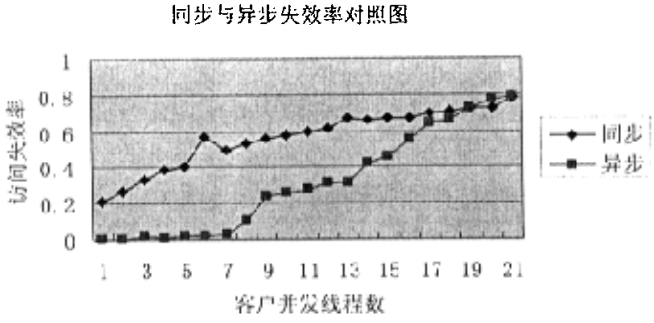

图 5.7 同步与异步比较图

1. 访问的失效率会随着客户并发线程数的增加而略成线性增长；
2. 当客户并发线程数达到 1200 个之后，失效率会达到 70%以上，并不断增加；
3. 当客户并发线程数在 800 个之后，异步访问的失效率突然上升；
4. 当客户并发线程数在 1600 个之后，异步访问的失效率与同步访问的失效率基本相同；

分析上面的现象，原因分别是：

1. 随着并发访问量的增加， StarWebAdaptor 的负载程度不断增加，对于响应的处理成功率明显下降；
2. 1200 个是 StarWebAdaptor 并发访问的默认最大值，所以当超过瓶颈值时，失效率会迅速增加；
3. 达到 800 个以后，异步访问的失效率突然增加的原因同 2，但参考图 5.2 发现，由于请求到达目标服务之前需要经过 SJIT 的处理，所以相对的并发数会小于 1200；
4. 最后，由于 SJIT 中需要连接 JMS 系统，将翻译后的 JMS 消息发送给指定的请求队列中，但 JMS 系统的默认连接数为 1500，所以当超过瓶颈值时，失效率会迅速增加；

通过对比测试可以看到异步访问方式下，在相同的用户并发线程数时，访问的失效率明显好于同步，虽然由于瓶颈值的存在，在并发数较大时，同步与异步的访问方式失效率大致相同，但可以适当的调整瓶颈值，使得异步的访问失效率在一定程度上得到降低。

§5.6 本章小结

优良的系统性能与具有代表性的测试分不开，为了对设计系统的性能有一个直观的表述，我们选择了一个简单但比较有代表性的应用作为测试案例，分别就同步方式和异步方式进行了测试。从中我们得到了四条有意义的结论，这些结论对系统以后的改进给出了有益的建议。

第六章 结束语

§6.1 全文工作总结

Web 服务的研究与发展为开发分布式应用提供了有效的支持，使客户应用程序能够以更为松散耦合的方式向服务对象发送请求和接受应答。但随着应用的深入，Web 服务对异步可靠应用的支持不足成为其发展的瓶颈，支持分布式异步可靠的 Web 服务也就应运而生。如何将成熟的异步可靠技术应用于松散耦合的 Web 服务领域，本文对这个方向进行了讨论。

本文以现有的 Web 服务体系结构和规范集合为基础，基于消息中间件技术，以信贷业务领域为典型的应用场景，研究了 Web 服务中的异步消息传递机制和可靠性，并以此为基础开发了一个能够提供异步可靠 Web 服务的中间件系统。

§6.2 工作展望

随着对异步性和可靠性关注的不断深入，我们下一步的研究任务主要有：

- 跟踪与可靠性相关的规范制定，包括 WS-R(Web Services Reliable)、WS-RM(Web Services Reliable Messaging)，掌握研究的主流方向；
- 与 CORBA 的集成研究，本文的主要工作集中在如何将异步和可靠性引入 Web 服务之中，为了将 StarBus 作为异步可靠 Web 服务的实现者，需要进行以下的工作，例如 JMS 与 CORBA 通告服务的链接；

致　谢

在本文即将完成之际，首先要感谢的是我的导师，国防科技大学网络所所长王怀民教授！回想这两年多的工作学习生活，王老师给了我太多的教诲和帮助。王老师学识渊博，治学严谨，思维敏锐，使我在学业上受益匪浅；在生活上，王老师也给予我无微不至的关怀、指导和鼓励。高山仰止，景行行止！在此谨向王老师表达我最衷心的感谢和最诚挚的敬意！王老师的谆谆教导，学生铭记在心！

感谢国防科大计算机学院网络与信息安全研究所613室主任贾焰教授！贾老师从课题选择、课题研究到最后论文写作都给了我精心的指导，我前进的每一步都倾注着贾老师的心血。贾老师高深的学术造诣和无私的敬业精神也是我学习的楷模。

感谢国防科大计算机学院吴泉源教授、邹鹏副教育长和高洪奎研究员！他们深厚的专业知识和勤奋的工作精神对我有极大影响。感谢613室所有的老师，感谢他们对我在课题研究中的支持和帮助。

特别要感谢周斌老师！感谢他对课题自始至终的关注，感谢他提供的大量资料以及在研究方法上对我的指导。其踏实严谨的学风，谦虚热情的为人对我产生了深深的影响。

特别感谢刘必欣、孙海燕、刘玲霞、郑笛师姐、王玉锋、水超、胡建强、王俊等师兄以及苏亮、官延安、杜凯、袁志坚等同学，同他们的交流探讨使我解决了许多问题。感谢刘必欣师姐认真校对了全文。

感谢梁妍大姐提供的热情帮助。

最后，要感谢我的父母。停笔在即，儿无论在哪里最想见到的是你们。二十多年的养育之恩，无以为报。谨以此文献给父母，希望能使你们感到欣慰。我爱你们。

附录 A：攻读硕士期间发表的论文

左克、王怀民、刘必欣、周斌 基于MOM技术的Web Services异步性的研究 第20届全国数据库学术会议论文集 B集 257～360 湖南 长沙 2003年10月

附录 B：攻读硕士期间参加的科研项目

863 项目基于 Web Services 的应用集成关键技术研究（No.2002AA116040）。

参考文献

[1] Dave Chappell Sonic Software Corporation, Doug Bunting Sun Microsystems, Inc., Martin Chapman Oracle Corporation, Web Services Reliability (WS-Reliability) Ver1.0 January 8, 2003.

[2] Ruslan Bilorusets, BEA, Don Box, Microsoft, Web Services Reliable Messaging Protocol, March 13, 2003.

[3] Newman P., ATM Local Area Networks, IEEE Communications Magazine, 1994, 32(3): 86~98.

[4] R. Braden et al., Resource ReSerVation Protocol(RSVP) Version 1 Functional Specification, Internet Draft, 1997, ftp://letf.org/internet-drafts/draft-ietf-rsvp-spec-15.txt.

[5] J D Boissonnat. Shape Reconstruction from Planar Cross-Sections. Computer Vision, Graphics, and Image Processing, 1988, 44: 1-29.

[6] S. Khanna and et. al., Realtime Scheduling in SunOS 5.0, In Proceedings of the USENIX Winter Conference, USENIX Association, 1992: 375~390.

[7] Scalable Real-Time Computing in the Solaris Operating Environment: A Technical White Paper, Sun Press, California U.S.A., 2000.

[8] Dr. ir. Martin Timmerman, Managing Director, Windows NT as Real-Time OS, Real Time Magazine, 1997, Q2: 6~13.

[9] Technology Brief: Real-Time Systems with Microsoft Windows NT, Publication of the Microsoft DeveloperNetwork, Microsoft Corporation, 1995.

[10] J. W. S. Liu, R. Rajkumar, Z. Deng, M. Seri, A. Frei, L. Zhang and C. S shih, How to Get Most Predictability Out of Windows NT, QUITE, 1999 http://quite.teknowledge.com/WorkingGroups/NT_RealTime_predict.htm.

[11] D. Schmidt., Middleware for Real-Time and Embedded Systems, Communication of the ACM, 2002, 45(6): 43~48.

[12] J. A. Stankovic, M. Spuri, K. Ramamritham und G. C. Buttazzo, Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms, Kluwer Academic Publishers, Dordrecht, 1998.

[13] C. L. Liu and James W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, Journal of the ACM, 1973, 20(1): 46~61.

[14] G. Cooper, L. DiPippo, L. Esibov, R. Ginis, R. Johnston, P. Kortman, P. Krupp,

J. Mauer, M. Squadrito, B. Thuraisingham, S. Wohlever, Real-Time CORBA Development at MITRE, NRaD, TriPacific, and URI, In Proceedings of the Workshop on Middleware for Real-Time Systems and Services, San Francisco, CA, 1997: 69~74.

[15] Yuruo Chen, Integration of Real-Time CORBA into an Existing Distributed Planning Architecture, University of Rhode Island Technical Report, TR98-266, 1998.

[16] Object Management Group, The Common Object Request Broker: Architecture and Specification, 2.4 edition, 2000.

[17] A. Courtney, W. Hanssen, et al., Inter-language unification, release 1.5, technical report ISTL-CSA94 -01-01, Xerox PRC, 1994.

[18] B. Thuraisingham, P. Krupp, A. Schafer, and V. Wolfe, On Real-Time Extensions To The Common Object Request Broker Architecture, In Proceedings of the Object Oriented Programming, Systems, Languages, and Applications (OOPSLA) Workshop on Experiences with CORBA, 1994.

[19] E. Bensley, P. Krupp et. Al, Object-oriented approach for designing evolvable real-time command and control systems, In Proceedings of the Workshop on Object-Oriented Real-Time Dependable Systems, 1996.

[20] D C Schmidt, D L Levine and S Mungee., The design of the TAO real-time object request broker, Computer Communications, 1998, 21(4): 294~324.

[21] The TAO Project, http://www.cs.wustl.edu/~schmidt/TAO.html.

[22] Chorus Systems, CHORUS/COOL-ORB Programmer's Guide, Technical Report, CS/TR-96-2.1, Chorus Systems, 1996.

[23] Chin, R. and Chanson S., Distributed object-based programming systems, ACM comput. Surv., 1991, 91~124.

[24] Abhishek Singh and Jonathan Billington, Creating an Internet Inter-ORB Protocol Service Specification, In the Workshop on Software Engineering and Formal Methods, Adelaide, Australia, 2002.

[25] William Stallings, High Speed Networks: TCP/IP and ATM Design Principles, Prentice Hall, 2001.

[26] Bulmer M, Principles of statistics, New York: Dover, 1979.

[27] Kleinrock, Queueing Systems, Volumn I:Theory, New York, Wiley, 1975.

[28] Kleinrock, Queueing Systems, Volumn II:Computer Applications, New York, Wiley, 1976.

[29] 张小明，吴泉源，王怀民，贾焰，基于分布对象的异步回调模型的研究与实现，计算机科学与工程，2001.

[30] 王玉峰，CORBA进程内对象访问的优化，南京大学学报增刊，2001.

[31] 王学斌，自调度线程库的研究与实现，2002年全国开放式分布与并行计算学术会议，武汉，2002.

[32] 徐光辉主编，运筹学基础手册，第十六章，计算机随机模拟（郭绍僖 任重），北京：科学出版社，1999：772~804.

[33] 国防科技大学网络技术与信息安全研究所613教研室，StarBus 技术报告，2001.

[34] 国防科技大学网络技术与信息安全研究所613教研室，StarBus服务手册，2002.

[35] 国防科技大学网络技术与信息安全研究所613教研室，StarBus用户使用手册，2002.

[36] 王怀民，史殿习，吴泉源，新一代分布式系统集成中间件，计算机学报，1997，20（增刊）：90~96