第一章 介绍 SOA 与 Web 服务
Introduction to SOA with Web Services

复杂性是信息技术（IT）必须面对的现实。无论是构建新应用、替换现有应用，还是及时处理各种维护与改进要求，都要处理各种复杂状况，这可是一项重大挑战。

然而，如果所有应用都使用公共的编程接口（programming interface）及互操作协议（interoperability protocol）的话，那么将有助于 IT 降低复杂性，已有功能也更容易被再次利用。公共编程接口统一了应用的使用方式——通过设立这样一种公共编程接口，我们将能更轻松地完成对现行 IT 基础设施（IT infrastructure）[1]的替换和更新。

这就是面向服务开发（service-oriented development）所承诺要带给 IT 界的，而且如

[1] 译注：IT 基础设施（IT infrastructure）指机构中与 IT 相关的各种硬件、软件、服务及数据通信设施等的总和。

果在部署服务时采用一种面向服务的架构（Service-Oriented Architecture，SOA），那么服务（services）将有利于创建各种新的战略方案，包括：

- 快速应用集成
- 自动化业务流程
- 支持多渠道服务（multi-channel access）的应用（包括固定设备和移动设备）

SOA 可以促进在不同的软件间进行服务合成（service composition），无论这些软件是已有的还是新的，是部门级的、企业级的，还是跨企业的，也不管它们是运行在大型机、中间层，还是 PC 或移动设备上的，都可以通过 SOA 将它们组合为流畅的 IT 流程，并有助于 IT 环境的改进。

由于 Web 服务的广泛普及以及 SOA 带来的优势，这些合成应用的方案是可以实现的。WSDL（Web Services Description Language）现已成为访问任何应用的标准编程接口，而 SOAP 也已成为连接不同应用的标准互操作协议。这两个标准是重要的开端，更多针对企业特征及服务质量方面的需求而制订的 Web 服务规范将接踵而来，它们将对 Web 服务的安全（security）、可靠性（reliability）、事务（transactions）、编制（orchestration）以及元数据管理（metadata management）等方面加以定义。总之，Web 服务是构建下一代 IT 基础设施——SOA——的最佳平台。

The Service-Oriented Enterprise
面向服务的企业

关键技术的融合以及对 Web 服务的广泛接受，将对面向服务的企业（service-oriented enterprise）产生极大的影响：显著增加机构的机动性，加快新产品、新服务的上市速度，降低 IT 成本，同时提高运营效率。

如图 1-1 所示，多种代表着行业趋势的关键技术正在融合，并共同促使着 IT 在面向服务的概念与实现上发生重大改变。这些关键技术包括：

- 可扩展标记语言（Extensible Markup Language，XML）。一种跨企业（或更广泛）的、公共的、中立的数据格式，它提供了：
  - 与编程语言、开发环境和软件系统无关的标准数据类型与结构。
  - 用于业务文档定义和业务信息（比如标准的行业词汇）交换的通用技术。
  - 广泛的 XML 处理软件（如 XML 解析器、查询器、转换器等）。
- Web 服务（Web Services）。基于 XML 的技术，用于传递消息、描述服务、发现服务以及其他扩充功能，它提供了：
  - 各种被广泛采纳的、用于分布式计算的接口描述，以及通过消息进行文档交换的开放标准。
  - 与下层执行技术和应用平台的无关性。
  - 企业级服务质量（比如安全性、可靠性、事务性等）的可扩展性。
  - 对合成应用（比如业务流程流、多渠道服务、快速集成等）的支持。
- 面向服务的架构（Service-Oriented Architecture，SOA）。一套用于实现应用间互操作以及重用 IT 资产（IT assets）的方法，它具有以下特征：
  - 对架构方面（比如治理、过程、建模和工具等）的强烈关注。
  - 具有恰当的抽象层次，有利于促进业务需求与技术能力的配合与协调，和创建可重用、粗粒度的业务功能。

- 是一种适于快速、方便地构建新应用的部署基础设施（deployment infrastructure）。
- 一个用于常见业务与 IT 功能的可重用服务库。

- 业务流程管理（Business Process Management，BPM）。用于自动化业务操作（business operations）的方法和技术，它：
  - 清晰地描述了业务流程，以便于理解、改进和优化。
  - 易于针对业务需求的变更而快速修改业务流程。
  - 将原来由人工完成的业务流程自动化，并实施业务规则。
  - 为决策者提供有关业务流程的实时信息与分析。

这些技术各自都已对业务计算的一个或多个方面产生了深远影响，如果将它们融合起来，即可提供一种综合平台。该平台将有助于获得面向服务的优点，并在 IT 系统的发展进程中走向下一阶段。

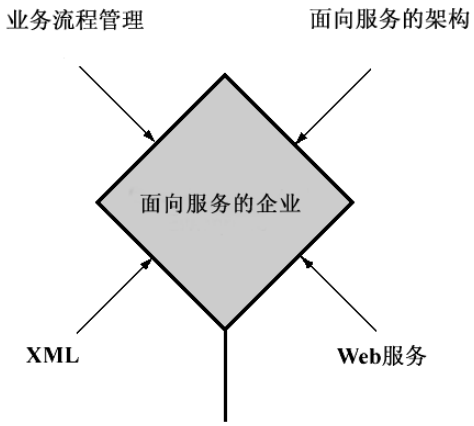

图 1-1 共同造就面向服务的企业的关键技术

Service-Oriented Development
面向服务开发

软件厂商们已经广泛接受了“采用 Web 服务进行面向服务开发”这一模式。面向服务开发（service-oriented development）是对之前的面向对象、面向过程、面向消息及面向数据库等开发方法的补充。

面向服务开发具有以下优点：

- 重用——创建可重用于各种应用的服务的能力。
- 效率——通过组合现有服务以快速创建新的服务与应用的能力，以及集中精力于数据共享而非底层实现的能力。
- 与技术的松耦合——独立于服务的执行环境进行服务建模（比如定义服务能够收发的消息）的能力。
- 职责的划分——令业务人员和技术人员分别关注业务问题和技术问题、两组人员通过服务契约进行协同的能力。

开发服务与开发对象不同：因为一个服务，是由它与其他服务交换的消息（exchanged messages）而不是由一个方法的型构（method signature）来定义的；服务定义所在的抽象层次必须比对象定义所在的抽象层次要高，因为这样可以把一个服务的定义映射到某种面向过程的语言（比如 COBOL 或 PL/I）、消息排队系统（比如 JMS 或 MSMQ）或面向对象系统（比如 J2EE 或.NET 框架）上。

理解服务定义的粒度（granularity）也同样重要。一般来说，服务都会定义粗粒度的接口。相对于对象的调用来说，对服务的单次调用会接收更多的数据，消耗更多的计算

资源，因为它需要进行到执行环境的映射和 XML 处理等工作，而且对服务的访问通常都是远程的。当然，对象接口也可以是粗粒度的。关键在于，服务是用来解决应用间的互操作问题以及用于组合新应用或应用系统，而不是为应用创建具体业务逻辑。

通过进行 Web 服务的聚合（aggregation），你可以发布封装了多个其他服务的 Web 服务。这样，可以将一个粗粒度的接口分解为多个细粒度的服务（或者，用多个细粒度的服务组合为一个粗粒度的接口）。粗粒度的服务更适合用于发布，而细粒度的服务则更适合作为仅供前者调用的“私有”Web 服务。

服务的执行即根据服务所支持的一种或多种消息交换模式（Message Exchange Patters，MEPs）进行消息交换，可用的消息交换模式包括请求/响应（request/response）、单向异步（one-way asynchronous）及发布/订阅（publish/subscribe）等。

在项目层次上，架构师通常要指导可重用服务的开发，并确定一种存放、管理和检索服务描述的方法。可重用服务层（reusable services layer）将业务操作（比如“获取客户信息”、“下订单”等）与下层软件平台的实现差异相隔离，就像 Web 服务器和浏览器将万维网与操作系统和编程语言的差异隔离开一样。将可重用的服务快速组合为更大服务的能力，令机构具有流程自动化和快速适应环境变化的优点。

How XML Helps Simplify Systems Development and Integration
为什么 XML 有助于简化系统的开发与集成

在 Web 服务中使用 XML 可以将服务的定义与执行明确分离——这种分离是特意的（可以参阅有关标准），它使 Web 服务可在任何软件系统上工作。利用 XML Schema 来表达

服务中的数据类型与数据结构，令开发者在处理传递于服务间的数据时，可以不必考虑特定服务的具体实现细节。这体现了在集成问题上的一个根本改变，即不必再为了与服务交互而了解其实现细节。无论服务的执行环境是一个对象还是一个消息队列或者存储过程，这已经不重要了，因为数据在被使用之前要经过 Web 服务的过滤处理，其中有一层过滤将会完成 Web 服务到实现它的执行环境的映射。

用服务来设计、开发和部署应用，需要思考方式上的重大转变。帮助完成这一转变的一个方法，是将 IT 部门的职责划分为两个部分：

- 创建服务——处理部署服务所涉及的下层技术的复杂性，并确保 XML/Web 服务的描述与服务消费者的需要相一致，而且双方共享着应有的数据。

- 使用服务——组装新的合成应用（composite applications）与业务流程流（business process flows），确保共享数据及流程流能够准确反映业务的运营和战略需求。

上述职责划分实现了技术问题与业务问题的明确分离。

Organizational Implication of SOA
SOA 对机构的影响

在过去的 IT 部门里，业务功能的理解和技术功能的理解是由同一个人负责的——即让同一个或同一些人完成业务与技术领域的衔接，这是 IT 领域的老问题了。要获得 Web 服务、SOA 以及 BPM 技术的全部优势，IT 部门必须考虑采用最佳的组织与能力的组合。

在采用一种新的架构和技术时，确定新角色和新职责是很重要的。技术人员必须能够适应从做全部工作到做部分工作，并与他人共同完成整个工作的转变。与对象或过程相比，服务的开发应面向一个更为宽广的环境，因为它被重用的机会更大。实际上，定义可重用的服务也许是面向服务中最重要的方面。要实现服务的最高价值，必须在开发时就考虑与其他服务的互操作，并通过与其他服务的组合来构建应用。这一思想上的转变，可能需要某个处于部门或机构领导职位上的人来协助完成检查设计，并确保它们与新的 IT 目标一致。

Service Abstraction
服务抽象

服务就是具有机器可读的（machine-readable）消息（接受或返回的）描述的网络位置，即服务是由它所支持的消息交换模式（MEPs）定义的。消息（message）中包含的数据具有相应的模式（schema），模式用于在服务请求者与服务提供者之间建立契约（contract）（即描述）。其他一些元数据项（metadata items）分别描述了服务的网络地址、所支持的操作，以及对可靠性、安全性及事务性方面的要求。

图 1-2 展示了服务的各个部分（包括服务描述、服务实现、映射层等）间的关系。任何提供 Web 服务支持的执行环境都可以作为服务实现（service implementation）。服务实现也被称作可执行代理*（executable agent）*，它负责实现各种 Web 服务规范中定义的 Web 服务处理模型（processing model）。可执行代理在执行环境*（execution environment）*中运行，这里的执行环境通常是某个软件系统或编程语言。

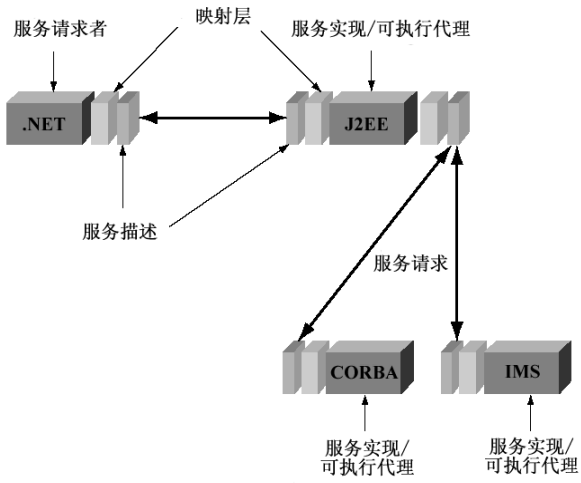

图 1-2　Web 服务的各个组成部分

服务定义中的一个重要方面是：服务描述（service description）与其可执行代理（executable agent）是分开的。一个服务描述可以有多个不同的可执行代理与之相关联，而一个可执行代理也可以支持多个服务描述。服务描述通过一个映射层（有时也称作转换层）实现与执行环境的分离。映射层（mapping layer）通常以代理（proxy）或桩（stub）[2] 的形式实现。映射层负责接收消息、转换 XML 数据到本地格式（native format）、将数据派送到可执行代理。

Web 服务区分请求者与提供者两种角色：服务请求者*（service requester）*通过向服务提供者发送一个消息来启动服务的执行；服务提供者*（service provider）*收到消息后便执行服务，然后将结果（如果有的话）返回给服务请求者。一个服务请求者可以同时也是一个服务提供者，反之亦然。也就是说，可执行代理可以具有这两种角色中的任一种、或者同时兼有两种角色。

如图 1-3 所示，服务抽象的好处之一是：易于访问各种不同类型的服务，比如新开发的服务、经包装的传统应用（wrapped legacy applications）或由其他服务（新的或传统的）合成的应用等。

[2] 译注：stub 也译作“存根”、“占位”或“占位程序”，这里采用“桩”这一译法。

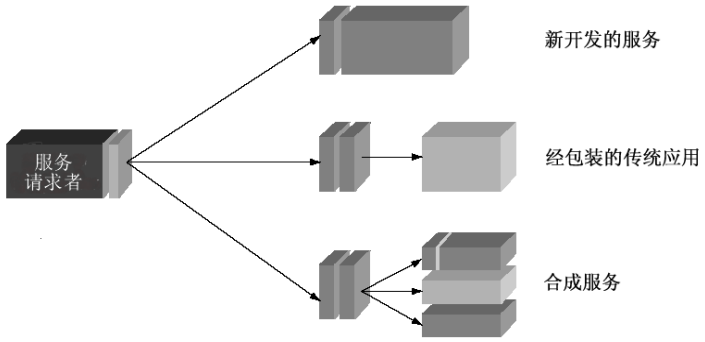

图 1-3　请求不同类型的服务

Separating the Service from the Product
让服务与产品相分离

有些软件厂商仍未能将服务的概念与执行环境的概念相分离，将 Web 服务实现仅作为另一个已有产品的一部分进行销售。这将较难获得 Web 服务带来的优点，因为这些产品中包含了并非执行 Web 服务所需的特性，而且如果服务还要依赖于这些特性的话，那么会造成不兼容。

Service-Oriented Architecture
面向服务的架构

本节介绍服务及面向服务的架构（SOA）主要概念与定义。

What Are Services
什么是服务

在继续技术讨论之前，我们先从业务的观点来看一下服务（services）与流程（processes）的概念。大多数机构（无论是商业的或政府的）都要为顾客、客户、公民、员工或合作伙伴等提供服务。我们来看一个实际的面向服务的例子。

如图 1-4 所示，银行出纳员要为银行客户提供服务。不同的出纳员可提供不同类型的服务，某些出纳员可能是专门为客户处理某类服务的。典型的服务包括：

- 账户管理（开户与销户）
- 贷款（处理申请、查询条款与细则、同意放款）
- 提款、存款与转账
- 外汇兑换

存在多个提供一组相同服务的出纳员——这有利于分摊工作量和提供较高的可用率，不过就客户而言，他们只关心能否完成服务，至于银行内部如何安排则与他无关。如果是一个复杂的交易，客户可能需要与多个出纳员接触才能完成处理——这就是一个业务流程流（business process flow）。

图 1-4　银行服务的例子

实现银行服务自动化的 IT 系统存在于银行内部，对客户是透明的，因为服务最终是通过出纳员提供给客户的。因此，IT 系统所实现的服务，必须与出纳员向客户提供的服

务一致，并且必须向出纳员提供支持。如果能确保“IT 系统所实现服务的定义”与“业务功能和业务流程”一致，那么 IT 系统将更易于支持业务目标，更适于提供由人或 ATM 以及在 Web 上提供的服务。

图 1-5 显示了 ATM 机用户、办公网络上的出纳员，以及使用 PC 的 Web 用户是如何使用相同的服务的。这样的服务设计与部署满足了客户的需要。服务的实现环境并不重要，重要的是服务本身。图中也显示了如何由两个服务组合为一个新服务，比如将取款与存款服务组合为一个转账服务。

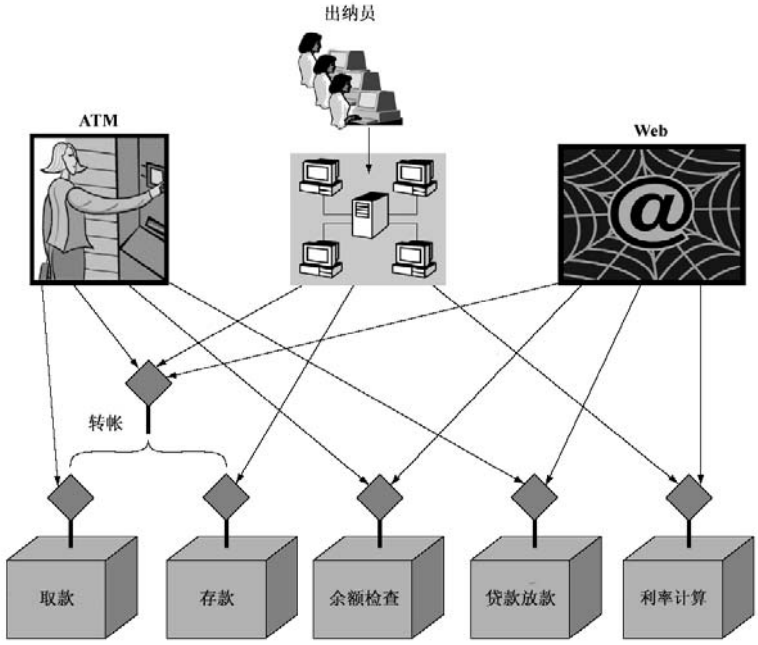

图 1-5　使用和组合服务

软件服务（software services）[3]的定义与银行提供的业务服务（business services）相一致，将确保业务运营的流畅，并有助于实现战略目标（比如：除了通过柜台方式以外，

[3] 译注：这里的软件服务（software services），是相对业务服务（business services）而言的，不要理解为某种具体的服务。

还允许通过 ATM 和 Web 方式使用银行服务）。

我们将看到，复杂的服务（比如处理一个订单或一次保险索赔）可由多个服务组合而成。在 SOA 环境中部署服务，令服务合成更加容易；而由此合成的应用，也可以发布为服务，供人或 IT 系统使用。

What is Service-Oriented Architecture
什么是面向服务的架构

面向服务的架构（Service-Oriented Architecture，SOA）是一种设计方式，它指导着业务服务（business services）在其生命周期（从构思开始，直至停止使用）中包括创建和使用的方方面面。SOA 也是一种定义和提供 IT 基础设施（IT infrastructure）的方式，它允许不同应用相互交换数据、参与业务流程（business processes），无论它们各自背后使用的是何种操作系统或采用了何种编程语言。

SOA 可被看作是一种构建 IT 系统的方案，它将业务服务（即一个机构向顾客、客户、公民、员工、合作伙伴或其他机构直接或间接提供的服务）作为协调 IT 系统与业务需求的关键组织原则。相反，早期构建 IT 系统的方案，大多是直接使用特定的实现环境（比如面向对象、面向过程、面向消息等）来处理这些业务问题，结果造成 IT 系统依赖于具体执行环境（比如 CICS、IMS、CORBA、J2EE 和 COM/DCOM 等）的特性与功能。

Competitive Value of SOA
SOA 的竞争优势

用 Web 服务成功地实现了一种面向服务的架构（SOA）的企业将具有一定竞争优势，因为针对战略性业务目标构建的服务可以对不断变化的业务需求做出更快的响应，而那些针对具体执行环境构建的 IT 系统则不可能做到。其他优势包括：易于进行 Web 服务的合成及改变 Web 服务的合成方式；另外，修改 Web 服务及 XML 数据的代价，也比修

改执行环境的代价低。用 Web 服务实现的 SOA 具有以下优势和优点：在项目的 IT 开销上具有更理想的投资回报，项目见效更快，对不断变化的商业和政府需求的响应更快。如果一个企业具有一种有助于其自身快速改变的 IT 基础设施，那它就比不具有此特征的企业更具优势。此外，将 SOA 用于应用集成（application integration）、业务流程管理（business process management）及多渠道服务（multi-channel access），可以令企业创建更具战略性的 IT 环境，以更符合业务的运营特征。

SOA 是利用“面向服务开发（service-oriented development）”之优点的最好方式。与之前的 IT 系统关注于技术本身、让人们适应技术不同，面向服务（service orientation）通过将技术更自然地贴近于需要它的人们，实现了项目成本的缩减和项目成功率的提高。面向服务开发（service-oriented development）与先前开发方法的主要不同在于，它使你可以专注于业务问题本身的描述，而先前的开发方法则要求你更关注特定执行环境技术的使用。面向服务开发比先前技术更面向业务问题的解决 。

SOA 的概念并不新，新颖之处在于它能够混合搭配各种执行环境、令服务接口与执行技术明确分离、令 IT 部门可以为各项工作（无论是新的还是现有的应用）选择最佳的执行环境，并采用一致的架构方式将它们结合起来。而先前的 SOA 实现，是基于一种单一的执行环境技术的。

SOA isn't New, So What Is
SOA 的概念早就有了，其新颖之处何在

尽管 SOA 背后并没有什么新概念，但人人都在谈论着 SOA。在进行软件服务（software services）的定义时所采用的"接口与实现分离"的思想，是早已在 J2EE、CORBA、COM 甚至 DCE 中得到充分检验的。但是 SOA 主要是通过文本文件来实现服务与其执行环境的分离，而且分离得更明确、更完全，SOA 的这种能力是新增的——这主要借鉴自 Web 的概念和技术。传统的接口实现没有考虑这样一种"松散的"分离，因为它会影响效率。然而在许多情况下，易于获得互操作性的能力要比效率问题更为重要——互操作性是业界一直在努力争取、但至今仍未彻底实现的目标。SOA 成功与否，并不依赖于 Web 服务给 IT 软件带来的发展，而是依赖于方法的转变。Web 服务中接口与执行环境的进一步分离促进了工作职责的分离。将服务描述与其技术实现相分离，意味着企业可以根据服务描述对业务运营问题的实现进行考虑和进行 IT 投资，而那些依赖个别产品或软件技术来执行服务描述的方案则不行。这样，服务描述所定义的特征与功能可以通过任何技术来实现。但是，只有转变了对 IT 的思考方式的企业才能实现这一点。服务就是发布了的、可供用户使用的事物。当然，这只是夸张的说法；许多服务与应用的开发，仍然以快速而灵活地实现业务操作的自动化为目标，就像办公软件的自动化那样。不存在既能操纵车辆又能计算火星漫游者（Mars Rover）的运动轨迹的计算机系统，同样也不存在能够解决所有需求的软件解决方案。同样的计算机系统不可能也不应当——比如说——既要控制胰岛素注射器，又要处理 Amazon.com 的订单，同时还要为 Google 跟踪 Web 链接。多样性是世界的本质特征，而 SOA 与 Web 服务就具有这里所说的多样

性，因而具有构建更贴近于业务操作（business operations）的 IT 系统的能力——这是之前的软件方案所不能及的。然而，这要求整个软件产业（不仅仅是 IT 部门）在观念上的重大转变。很难想象会有在各方面都很内行的硬件厂商；同样地，软件厂商也不会是所有软件方面的专家。专业化生产、以及利用专业化生产得到的组件进行可重用的组装（assembly），正是软件行业所需要。根据这样的设想，软件厂商将变得更加专业，比如专注于进行服务组装而不是提供整个产品。在支持 SOA 的环境中，企业可能不仅要学会考虑不同执行环境中的服务，还要知道如何用来自各个厂商的组件来组装应用。

SOA 的真正优点体现在部署以后的各阶段中。那时，就可以通过组合现有服务来开发整个或近乎整个新应用。一旦可以通过现有的可重用服务来组合新应用，便可以实现最低成本、最短时间以及最佳投资收益率（ROI）。但是，实现这一点是需要一些时间的，并且需要在服务开发上作相当大的投资。

重用通用的业务服务（business services）（比如客户姓名查找、邮政编码验证以及信用检查等）的益处是容易理解的。在出现面向服务开发环境之前，这些功能一般是通过在新应用中使用可重用的代码库或类库完成的。而在基于 SOA 的应用中，这些通用功能以及一些常见的系统功能（比如安全检查、事务协调及审核等）是通过服务实现的。使用服务，不仅减少了需要部署的代码量，而且集中化的代码部署与管理还减轻了管理、维护和支持方面的负担。不过，访问服务的效率是必须要评估的，因为通常使用服务比使用可重用代码库要消耗更多的计算资源。

成功实现 SOA 的关键，是为可重用服务库中的每个服务确定正确的设计与功能。这些服务一定要反映机构的运营特点，比如如何申请授权、如何管理隔夜现金（overnight cash）以及如何把集装箱从货船移到卡车上，等等。SOA 要将业务的运营特性自动化，而成功的 SOA 项目应确保可重用的软件服务（software services）与实际的业务流程（business processes）完全一致。业务流程与其软件实现的良好配合与协调，令业务的运营流程可以根据外界环境的变化作出快速变化，从而令机构能够适应发展的需要。

CORBA vs. Web Services for SOA
实现 SOA，用 CORBA 还是 Web 服务

熟悉 CORBA 标准的人经常会说，Web 服务就是用 XML 实现的 CORBA，并例举出 Web 服务比 CORBA 缺少的种种特性。其实，不少 CORBA 部署都是面向服务的架构（SOA），而且 CORBA 的初衷与 Web 服务的目标也非常相似。有些人会说，CORBA 未能取得广泛成功，源于商业上的政治因素；虽然这么说有点道理，但 CORBA 在早期没有为互操作定义标准，也确实阻碍了其自身发展。过去，在被问及这一点时，OMG 的人总是说："这是留给厂商和用户们去完成的"；其言下之意就是（有时也直接这么说），如果你已经定义了标准接口，那么互操作就不成问题了。Web 服务是基于 SOAP 的，而后者是一个互操作标准。在使用 SOAP 时可以不使用 WSDL（许多人都是这么做的），这体现了 CORBA 和 Web 服务的区别。在 CORBA 中，如果要实现传输的互操作，就必须使用 CORBA 的接口定义语言（Interface Definition Language，IDL）；实际上，所有与互操作相关的代码都是由 IDL 生成的。许多 Web 服务工具包，也支持根据 WSDL 生成代理（proxy）和桩（stub）代码，而且还能生成 SOAP 消息。但这只是一种选择，并不是 SOAP 标准的规定。从技术角度上看，几乎所有 Web 服务能实现的，用 CORBA 也都可以实现；

并且对某些应用来说，CORBA 仍是一个较好的选择。但是从人的角度来看，Web 服务最重要的方面在于：不熟悉 CORBA 或不了解分布式计算及 Web 服务的人，也能容易地学习并使用之，而且互操作性要比缺少的某些特性更为重要。

Challenges to Adoption
影响 SOA 被接受的诸多因素

影响 SOA 被接受的主要因素在于：必须投入足够的人员进行训练，并保持相当的能力，才能确保所开发的服务是可重用的。任何技术，无论看上去多么有前途，都有被误用的可能。服务的开发，不应只顾及眼前利益，也要（或许是更重要的）考虑长期利益。换句话说，各个服务的单独存在并无太大价值，除非这些服务能与其他服务一起被其他应用所使用，并能用于合成各种新的应用。另外，为可重用服务进行准确的定义也并非一蹴而就的。

另一个因素是短期成本的管理。构建一个 SOA 的成本并不低；对现有系统进行再工程（reengineering）的耗费是巨大的，并且回报期也比较长。所以，应由业务分析师来定义业务流程（business processes），由系统架构师来将流程（processes）变为规约（specifications），再由软件工程师来开发新的代码，而项目经理要跟踪整个过程。

再一个因素就是：某些应用要加以调整后才能融入 SOA，某些应用可能还没有可以支持服务的调用接口，有些应用仅能通过文件传输或者通过对数据输入/输出进行批处理来访问，因此需要借助另外的程序才能融入 SOA。

当然，逐渐采用 SOA，并在能够产生重大商业影响的场合利用 SOA，可以减缓困难

和分摊成本，尤其是在可用服务来解决战术问题的时候。因此，接受 Web 服务与 SOA，在某种程度上就是要识别哪些是可以通过解决当前的困难（比如集成 J2EE 和.NET 框架应用）而立即受益的项目，并同时设立用于部门或企业 SOA 的基金。

What Web Services Are Good For
Web 服务有什么好处

你会在一些 Web 服务的文献中看到一些有关 Web 服务所不适用的场合（比如任务关键型（mission-critical）应用的开发与部署）的讨论。但是，认为 Web 服务不适于构建任务关键型应用的观点是不正确的。你也会看到一些关于为特定的数据类型（比如客户 ID、客户姓名等）识别“黄金副本（golden copy）”（也称作“单一引用”）[4]数据项实例的讨论。虽然这确实是一个有待解决的问题，但它并不应由 Web 服务来解决——即它应在 SOA 层而不是在 Web 服务层上解决。这些讨论常常反映出对 Web 服务以及 Web 服务所要解决问题的不了解。虽然人们容易理解“工场里的一个工具不可能用于各种用途”的道理，但是他们却会对 Web 服务抱有错误的想法，认为 Web 服务必须适用于先前其他技术所适用的所有场合。他们认为，每一轮新技术，都会以某种方式取代上一轮技术或之前的所有技术。Web 服务不仅为解决 IT 问题增加了新技术，而且还提供了一种不同的解决方案；另外，Web 服务具有的新能力，令它能够解决集成方面的难题。其实 Web 服务并不是一种替代技术，它不同于一种新的编程语言（比如 Java、C#等），因此不要认为它必须具有那些成功的编程语言所具有的各种主要特征。Web 服务也不同于像 J2EE、CORBA 及.NET 框架那样的中间件系统。Web 服务是基于 XML 的接口技术；Web 服务是非可执行的，它没有执行环境，需要依赖其他技术来创建执行环境。你只有根据 Web 服务的特

[4] 译注：黄金副本（golden copy）指的是一种资源的副本，它被用作该资源的引用副本。所有涉及修改资源的操作，都必须在黄金副本上进行。与黄金副本相对的是镜像副本（shadow copy），即除黄金副本以外的所有副本。

点、功能和能力来重新考虑你的 IT 方案，才会获得应有的优点。成功地应用 Web 服务技术，需要一个对待技术的思想转变，而不是学了新原理却用旧方法来构建和部署系统。Web 服务总是需要进行技术融合的；因此，不仅要知道 Web 服务取代了什么，还需要了解 Web 服务增添了什么。

SOA and Web Services
SOA 与 Web 服务

用 Web 服务实现 SOA 的主要优点在于：Web 服务是广泛普及的、简单的和平台中立的。

如图 1-6 所示，基本的 Web 服务架构包含了 SOAP、WSDL、UDDI 等支持服务请求者与服务提供者进行交互，以及用于 Web 服务发现的规范。服务提供者通常用 WSDL 来描述它所提供的 Web 服务，然后将该 WSDL 描述发布；服务请求者可以通过 UDDI 或其他注册库（registry）来获取 WSDL 描述，并通过向服务提供者发送一个 SOAP 消息来请求执行服务。基本的 Web 服务标准，已经适于构建基于 SOA 的应用了，但还不足以构建其他应用。

Why UDDI Is Not a Core Web Services Specification
为什么 UDDI 不属于基本的 Web 服务规范

可以肯定地说，UDDI 的最初设想并没有实现。UDDI 于 2000 年底推出，其初衷是要实现一个公共目录（public directory）。原先的设想是：一家公司在 UDDI 中注册它的 Web 服务，随后其他公司便可动态地发现它们所需要的、支持在 Internet 上使用的服务。当时 UDDI 的假定是：各个公司将乐于发现和请求来自之前无合作关系的提供者的服务。这一点已被证明是错误的。另外，UDDI 是在 WSDL 之前被开发的，所以 UDDI 最初并不能很好地支持 WSDL。UDDI 的数据结构是无限制的，只规定了很少的信息，而且结

构所基于的分类数据（categorization data）也不是被广泛认同的——这样的数据结构造成了极多问题。UDDI 曾被定位为企业内部的技术，也就是在这方面 UDDI 取得了一些成功，但是 UDDI 在这方面的标准工作做得仍不够。在内部使用 UDDI 的公司必须定义它们自己的命名规则和分类结构（categorization structure）与元数据（metadata），这些都妨碍了 UDDI 被采纳。随着 SOAP 与 WSDL 获得了极大的成功和普遍的采纳，UDDI 仍在为寻找其在 Web 服务中的适当应用场合而挣扎。不错，Web 服务平台是需要一个服务注册库（service registry），但 UDDI 是否能够成为解决方案尚不可知。

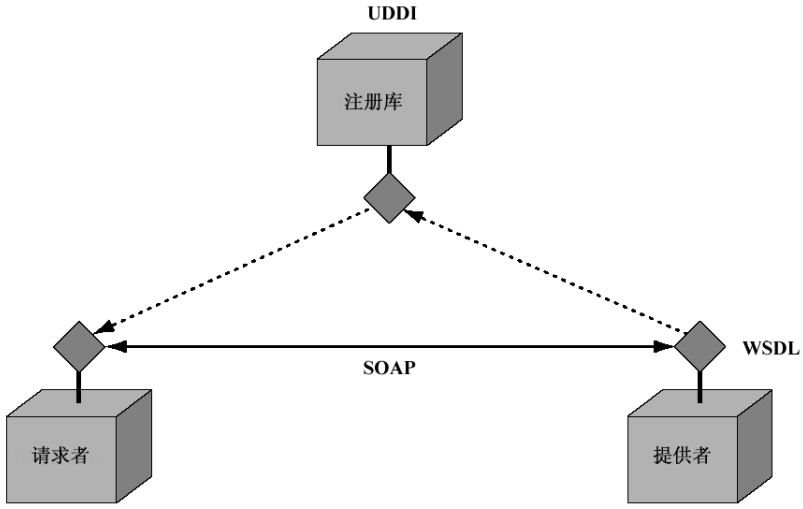

图 1-6　基本的 Web 服务架构

除基本的 Web 服务规范（SOAP 和 WSDL）以外，其他各种用于安全性（security）、可靠性（reliability）、事务性（transactions）、元数据管理（metadata management）以及服务编制（orchestration）等方面的补充 Web 服务规范也正在标准化的过程之中，这些都为基于 SOA 的解决方案提供了必需的企业级服务质量，以支持各类任务关键型（mission-critical）、企业级的项目。

Borrowing from the Web
借鉴 Web 的成功经验

用 Web 服务作为实现 SOA 的技术平台具有许多优点，这些优点的获得与万维网（World Wide Web）取得巨大成功的方式是一脉相承的。其中最主要的是：一种简单的文档标记语言（比如 HTML 或 XML）和一个轻量级的（lightweight）文档传送协议（比如 HTTP）；前者提供了强大的互操作性，后者提供了有效的、统一的数据传送机制。在 Web 上，使用哪个操作系统（比如 Linux、Windows、OS390、HP NonStop 或 Solaris），采用何种 Web 服务端（比如 Apache 或 IIS），用什么语言（比如 Java、C#、COBOL、Perl 或 LIST）编写业务逻辑，以及使用哪种浏览器（比如 Netscape、Internet Explorer、Mozilla 或 W3C 的 Amaya）都无关紧要。最重要的是 Web 服务端能够理解请求 HTML 文件的 HTTP 请求，以及浏览器知道如何将 HTML 文件显示出来。Web 服务也为 IT 系统提供了同样层次的抽象。类似地，对于 Web 服务来说，最重要的是能够理解并处理收到的 XML 消息并发出回复（如果有定义的话）。任何具备 TCP 连接的计算机系统都可以增加对 HTML 和 HTTP 的支持。同样地，任何能够理解 XML 和 HTTP（或者 XML 和其他普及的通信传输协议）的计算机都可以增加对 Web 服务的支持。

图 1-7 显示了完整的 Web 服务平台所具有的特性与能力，在该平台上可以构建各种基于 SOA 的应用。图中既包括了基本的 Web 服务规范，也包括了一些补充的 Web 服务规范。关于 Web 服务平台的详细描述，请参见第二章“SOA 概述”。

该 Web 服务平台包含支持 SOA 所需的各种基本与补充特性，以及一个用于连接各个服务的企业服务总线（Enterprise Service Bus，ESB）。

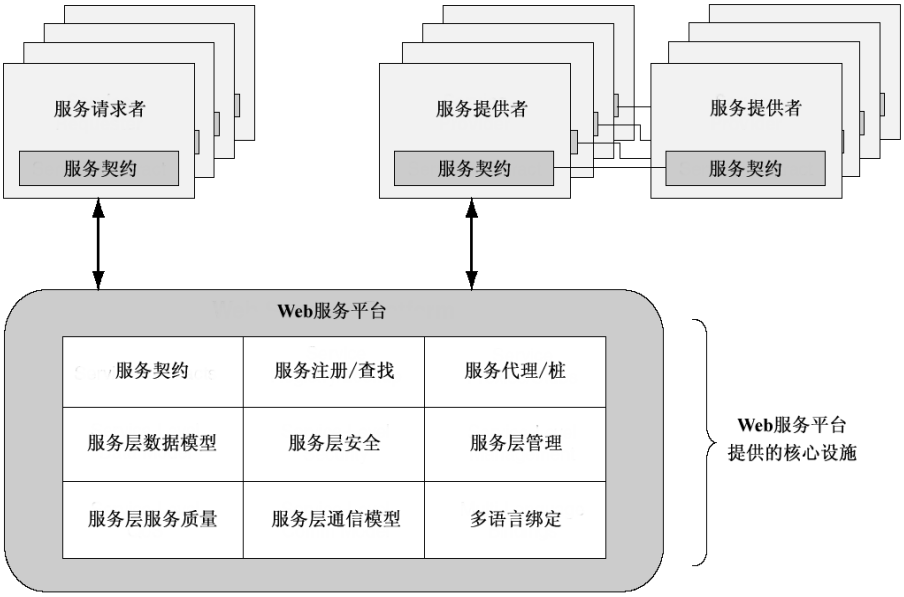

图 1-7　Web 服务平台

Rapid Integration
快速集成

几年前，觉得需要综合集成方案（integration solution）的企业转向了专用于此用途的产品与实务（practices）。可是，这些企业应用集成（Enterprise Application Integration，EAI）产品是昂贵的，它们耗去了大量的时间和精力，而结果却往往是失败的。而且，由于各种专门用途的产品是专有的（它们之间不可互操作），因此如果一个公司购入了多种这样的产品，那么很多项目将遭遇额外的麻烦。

最近的经验表明，一个较好的解决方案是使用 Web 服务标准。通过使用一种 EAI 产品来增加一个开放的、基于标准的抽象层，可以更容易地实现与几乎任何现有环境的集成，而不必处理由不兼容的应用引入的复杂性。

BEA、IBM、IONA、Microsoft、SAP、SeeBeyond、Systinet、Tibco、WebMethods 等公司的新一代的、基于 Web 服务技术的集成产品（integration products）正围绕着面向服务的集成（service-oriented integration）的概念而诞生。

Web 服务与 SOA 的组合，提供了一种快速集成方案，它关注被共享的数据与可重用服务（而不是专有的集成产品），因此能够更快、更轻松地确保 IT 投入与企业战略保持一致。

下面通过一个例子来展示面向服务的集成（service-oriented integration）的优点。有三种较典型的金融业数据库应用，分别用于支持个人金融业务、公司金融业务和共同基金投资（mutual fund investment）等操作。用传统的三层架构（three-tier architecture）来开发应用，将区分表示逻辑（presentation logic）、业务逻辑（business logic）和数据库逻辑（database logic）。

如图 1-8 所示，传统的三层结构的应用可被重用为一个面向服务的应用（service-oriented application）：在业务逻辑层创建服务，利用服务总线（service bus）将上述应用与其他应用集成。面向服务的另一个优点是，如果业务逻辑层支持服务的话，将更易于实现表示逻辑与业务逻辑的分离。如果业务逻辑层支持服务的话，就容易实现各种 GUI 及移动设备与应用的连接，这比“为各个业务逻辑编写紧耦合的表示逻辑”要好。可以将表示逻辑置于一个独立的设备上，然后通过服务总线实现与应用的通信，而不必为同一种服务端使用与之紧耦合的表示逻辑层。

使用在业务逻辑层定义的 Web 服务，与使用各种不同的集成技术相比，更利于应用进行数据交换，因为 Web 服务体现了各种软件的一个公共标准。XML 可被用于独立地定义数据类型与结构。最后，在业务逻辑层进行面向服务入口点的开发，令业务流程管理引擎可以启动一个涉及多个服务的自动执行流程。

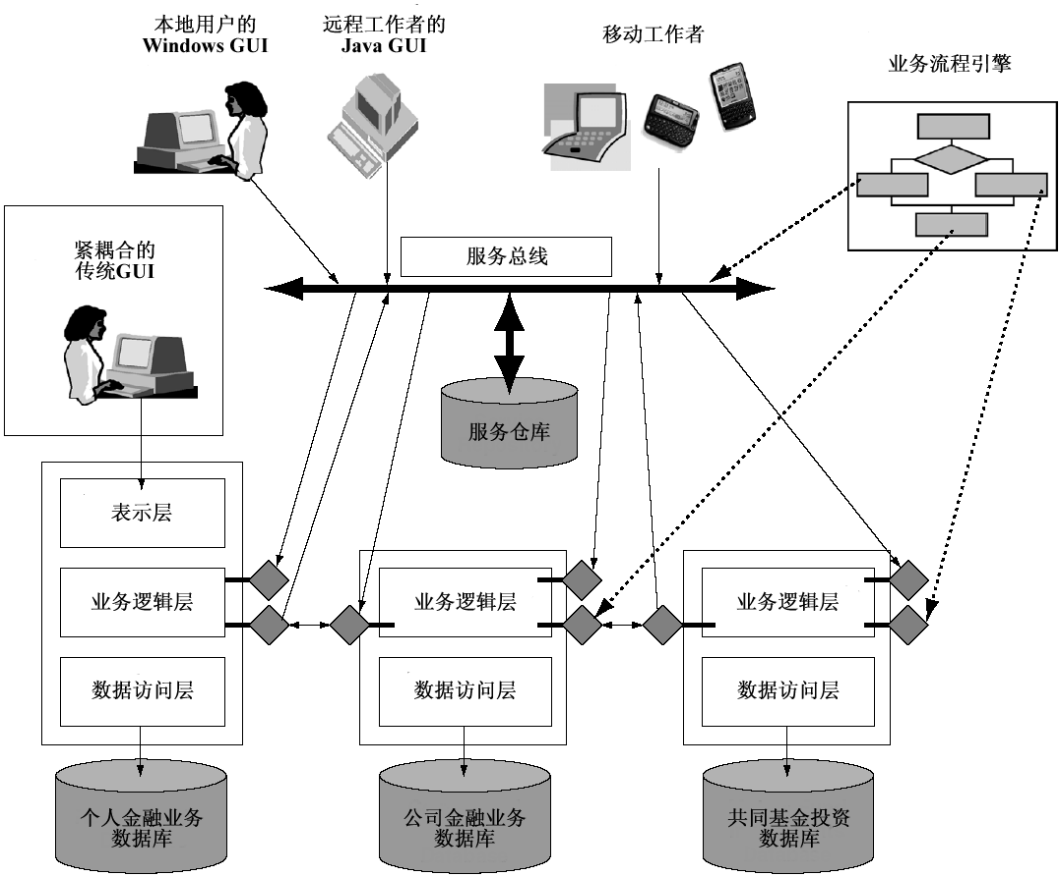

图 1-8 面向服务的集成

为应用的业务逻辑层创建一个公共的业务逻辑层（或称作“覆盖（overlay）”服务），使得我们可以通过公共的服务仓库（service repository）来存放和获取服务描述。如果一个新的应用要使用已有的服务，它可以通过查询服务仓库获得服务描述，然后利用服务描述生成与该服务进行交互的 SOAP 消息。

Multi-Channel Access
多渠道服务

大部分机构（商业的、政府的和非盈利的）的主要目标是为它们的顾客、客户、合作伙伴、公民或其他机构提供服务。这些机构通常通过多种渠道（channels）为其客户提供服务，以保证良好的服务和保持客户忠诚度。银行为了方便客户，提供了各种使用银行服务的方式，比如通过 Web、ATM 柜员机、出纳员窗口等。同样地，如果其他机构能让其客户通过多种服务渠道使用其服务（直接的或间接的）的话，将会受益于此。

一般而言，业务服务（business services）不如提供服务的渠道变化得频繁。业务服务体现了运营功能（比如账户管理、订单管理、编制账单等），而客户设备和服务渠道是基于新技术的，因此变化更为频繁。

由于 Web 服务可被各种客户端（比如 Web、Java、C#及移动设备等）使用，因此 Web 服务接口有利于提供多渠道服务（multiple-channel access）。所以，利用 SOA 与 Web 服务，机构可以容易地完成提供多种服务渠道的任务。

图 1-9 展示了多渠道服务的例子。其中，一家机构的客户服务应用发布了各种服务，用于报告问题、跟踪问题报告的状态、发现新的补丁和向用户发布的错误报告等。一位客户经理需要通过移动电话使用服务，以及时追踪客户的问题报告列表。如果产品是通过代理商销售的，代理商可以提供它自己的客户服务。代理商的客户服务可以与供货商的应用连接，以分别提供一级和二级客户服务支持。负责某重要客户的客户服务经理可以直接使用这些特性、功能以及存放在客户服务应用中的信息。客户自己也可以通过移

动 PDA 设备查询某个已提交问题的状态。最后，呼叫中心接线员需要访问应用所提供的服务来开展他们的工作（也就是与有问题的客户进行互动）。

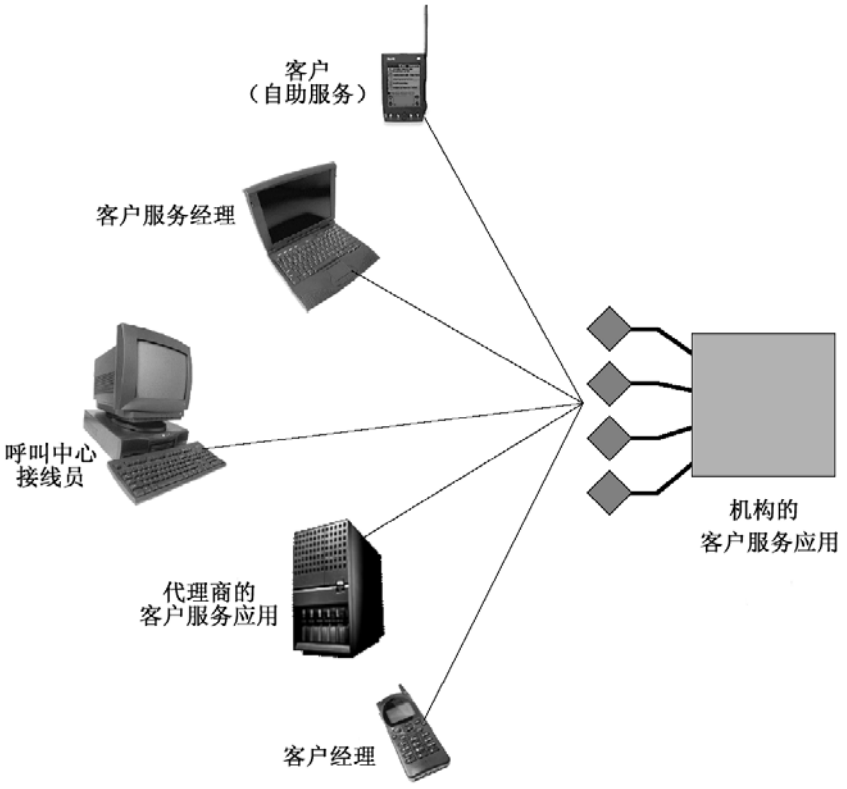

图 1-9　为客户提供多渠道服务

过去，机构开发的解决方案通常要为各种服务渠道分别提供一套完整的应用，比如 3270 终端、PC 接口或浏览器。服务渠道的增加（比如新的最终用户设备），令面向服务的企业可以随时随地为客户、供货商及合作伙伴提供更好的服务。但是，这也是对 IT 部门的重大挑战，因为需要将现有的单片式应用（monolithic applications）转换为可以提供多渠道服务的应用。当然，基本的方案是通过使用 SOA 与 Web 服务来提供这些应用。

Occasionally Connected Computing
偶发连接计算[5]

在把移动设备集成到 SOA 中时存在一个特有的问题，即移动设备不能保证与网络的持久连接。另外，移动设备会跨越不同的网络连接区域，并在重新连接时取得新的 IP 地址。目前营运环境中的多数应用都不能处理这种网络连接上的变化。正在兴起的新一代“移动（mobilized）”软件将快速而轻松地把移动设备集成到面向服务的企业（service-oriented enterprise）中。

在图 1-10 所示的移动软件方案中，从移动客户端到服务端的 SOAP 消息，是通过一种存储转发式的（store-and-forward）异步协议传输的，这种协议令移动客户端可以在网络连接不可用的情况下继续工作。在连接可用的情况下，事务（transactions）通过消息传输协议流向服务端的基于 SOA 的应用。在连接不可用的情况下，事务被保存在本地，待网络连接可用时再发送出去。

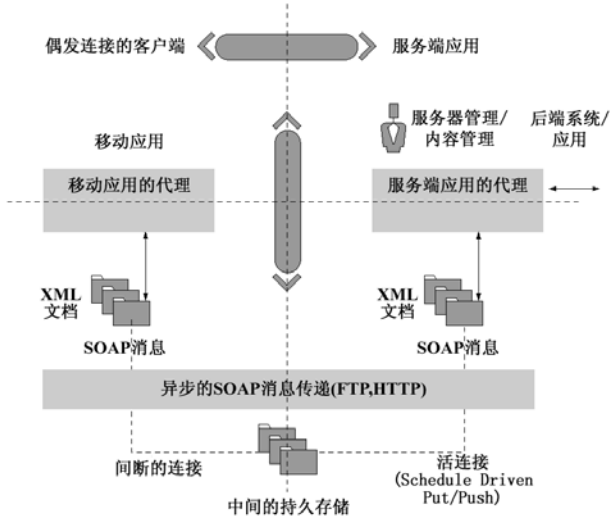

图 1-10 支持移动设备的偶发连接架构

[5] 译注：偶发连接计算（Occasionally Connected Computing，简称 OCC）是 Intel 公司提出的一种技术。详见 *http://www.intel.com/cd/ids/developer/asmo-na/eng/57606.htm?page=4*。

Business Process Management
业务流程管理

业务流程（business process）是现实世界中的一种活动，它由一系列在逻辑上相关的任务（tasks）组成，若根据恰当的顺序和正确的业务规则（business rules）来执行这些任务，便可产生业务效果——比如“从下订单到收款”、“订单兑现”和“保险索赔处理”等都是典型的业务流程。

业务流程管理（Business Process Management，BPM）是一套软件系统、工具和方法的统称，它关注机构如何识别、建模、开发、部署和管理上述业务流程。业务流程也包括 IT 系统与人类的交互（除非是 IT 系统已完全实现自动化的情况，这样的话，业务流程则仅包含 IT 系统）。早已有各种 BPM 方案在使用了，从最初的工作流系统（workflow systems），到现在的 Web 服务编制（Web services orchestration），都在此列。

BPM 技术可以以 SOA 为基础，利用 SOA 在架构上的成果，更好地进行业务流程自动化。投资基于 Web 服务的 SOA 的重要原因之一就是：Web 服务有利于轻而易举地完成 BPM 的目标。

BPM 系统的目标是协助达到“业务流程与期望业务结果的一致”，并确保 IT 系统能够支持这些业务流程。BPM 系统令业务用户（business user）可以在图形化的界面下，以便于 IT 部门实现的方式为业务流程建模。所有 IT 系统都是以某种形式来支持和实现业务流程的。而 BPM 的独特之处在于，它显式地将业务流程逻辑（business process logic）从其他的应用程序代码中分离出来，这与其他形式的系统开发构成了鲜明的对比：在那些系统中，业务流程逻辑是深嵌在应用程序代码中的。

将业务流程逻辑从其他的应用程序代码中分离出来，有利于提高生产力、降低运营成本和增加机动性。如果能够正确实现业务流程逻辑与其他的应用程序代码的分离（比如，将 BPM 作为 SOA 与 Web 服务的消费者），机构将能更快地对不断变化的市场环境作出响应，以把握获取竞争优势的机会。如果能够根据业务流程的图形化描述生成一个可执行的流程规约，那么效率可以得到进一步提高。

Business Operational Changes
业务运营的改变

各种业务（businesses）由于进入业务的原因不同而具有特殊的运营特征（operational characteristics）。举个例子来说，一家比萨店有“30 分钟内送达”的承诺。要确保做到这一点，就必须考虑到各种运营特征，比如烘烤比萨的时间、取订单的时间以及规定区域内的运送时间等。显然，运送是一种需要依靠人来进行、不能完全自动化的操作，但是流程中的许多其他部分是可被自动化的，比如可以在收到 Web 预定后自动提醒将物料运送至比萨餐馆，然后利用机器进行比萨的制作。就理想上来说，你希望尽可能在更多方面提高运营效率，保持最少的 IT 系统花费。基于 SOA 的基础设施能够帮助你实现这一点。

BPM 有助于简化“如何将‘解决不同业务问题的 Web 服务’组合起来执行”这一难题。如果把服务（service）看成 IT 系统与业务功能（比如处理订单）的对应，那么 BPM 层可被看成一种将多个服务联合为一个可以完成一定功能（比如验证订单、对客户的信用历史进行检查、计算库存量是否能够满足订单、最终运送订单和发送发票等）的流程流（process flow）。通过执行由应用层代码组成的流程流（process flow），业务流程将更易于针对新的应用特征与功能（比如供货商、库存管理或运送过程的变化等）作出变化与更新。

图 1-11 所显示的是一种用于自动化订单处理的流程流（process flow），该流程流以订单的录入为起点。第一步是接受订单文档，检查安全凭证，并向发送者提供“文档已收到”的确认。

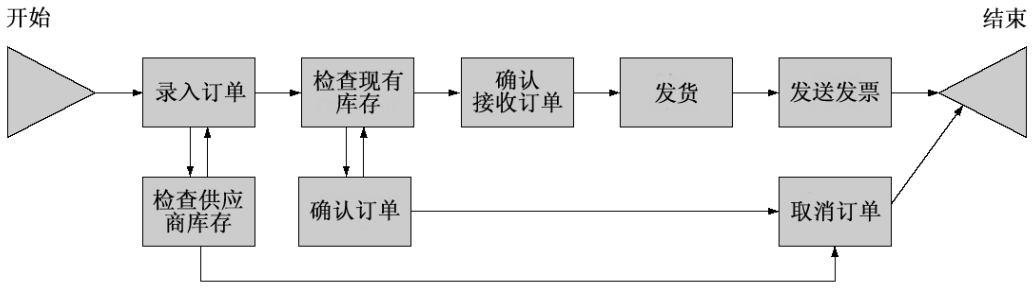

图 1-11　处理订单的业务流程流

通常，流程引擎（process engine）会保存输入文档，以便在后续步骤中使用。在文档经过验证之后，其位置（或者在数据库中，或者文件系统中）的引用（reference）将被发送给下一步骤，以便可以随时监测现有库存是否能够满足订单所需的产品数量。如果现有库存量足够的话，在流程流（process flow）的下一步将确认并接受订单，并通知客户订单可被满足。该确认通知可以用 Email 也可以用 Web 服务消息发送。

此时，客户可以再一次确认订单，以确认报价及运送时间。如果客户不进行再次确认，流程（process）将被取消，并进入“取消订单”步骤（主要是清除前面业务流程的工作，也可能包括一些补偿事务操作（compensating transactions）以取消为该订单预留的库存量，并取消相应的递送安排）。如果客户再次确认的话，下一步将是准备发货。收到发货确认后，流程流（process flow）将进入到最后一步，即将发票发送给客户。

通常流程流（process flows）由多个独立的任务（tasks）组成，其中每个任务都是一个 Web 服务。流程流会根据一项任务的执行结果来选择不同的分支以继续执行。可以在流程流的分支处进行错误处理。例如，如果一种产品在某一供货商处缺货的话，那么流

程流会进入这样一个分支：它向其他供货商发送 Web 服务请求，看它是否有足够的库存。如果所有可用的供货商都缺货，流程流将返回一个错误。

市场和制度的不断变化，要求机构具有相当的灵活性（flexibility）。为 IT 经理、架构师及开发人员提供一种通用的、基于服务的（services-based）解决方案，将有利于实现机构的灵活性，并易于实现生产力的提高。尤其是在为业务流程管理（BPM）奠定了 SOA 的基础之后，企业便可集中精力于更高层次的问题（比如设计最佳的业务流程，而不是关心应用实现的技术细节）。

Extended Web Services Specification
补充的 Web 服务规范

随着基本 Web 服务规范（SOAP、WSDL 等）被广泛接受和使用，对增加补充技术（比如安全性、事务性及可靠性等）的需求越来越大，这些补充技术对于任务关键型（mission-critical）应用是必需的。这些补充的特性有时也被称作服务质量（qualities of service），因为它们有助于解决一些 SOA 环境中的较难处理的 IT 问题，并使 Web 服务更适于在各种支持 SOA 的应用中使用。

对某些应用而言，核心的 Web 服务规范就已经足够了；但对另一些应用来说，缺少补充 Web 服务规范中的某些特性将有所妨碍。比如，有些公司希望它们发布的 Web 服务具有相当的安全性，并要求只接受通过可靠的消息保证机制发送的订单。

在定义核心的 Web 服务规范时，已经预留了扩展点（比如 SOAP 报头），以备添加补充特性的需求。

Standardization
标准化

各种 Web 服务规范正在通过不同的方式（比如，通过一小群厂商，或者通过被正式授权的技术委员会）走向标准化。从一般的经验来看，很多规范都是先从一小群厂商共

同努力开始，然后被提交给标准化组织以寻求被进一步接受。由 Microsoft、IBM 以及它们各自的合作伙伴制订的那些规范，往往能够获得较大的市场牵引力。 Microsoft 和 IBM 是 Web 服务规范运动的事实领袖它们已经定义或帮助定义了所有主要的 Web 服务规范。在本书编写之时，一些“WS-*”规范仍处于个别机构的控制之下，但是我们希望它们将来能被提交给标准化组织。

目前活跃在 Web 服务领域的标准化组织有：

- W3C（World Wide Web Consortium，万维网联盟）。致力于 Web 标准（如 HTTP、HTML 及 XML 等）的制定。W3C 是 SOAP、WSDL、WS-Choreography、WS-Addressing、WS-Policy、XML Encryption 和 XML Signature 的大本营。

- OASIS（Organization for the Advancement of Structured Information Standards，结构化信息标准促进组织）。成立之初致力于促进 SGML（Structured Generic Markup Language）[6]间的互操作性，自 1998 年开始使用现在的 OASIS 这个名称[7]，以强调其对 XML 的关注。目前，OASIS 主要推动的规范有 UDDI、WS-Security、WS-BPEL、WS-Composite Application Framework、WS-Notification、WS-Reliability、Web Services Policy Language（Extensible Access Control Markup Language TC 的一部分）（本书将会介绍以上规范）以及 Web Services for Remote Portlets、Web Services Distributed Management 和 Web Services Resource Framework（WSRF）等。

- WS-I（Web Services Interoperability，Web 服务互操作组织）。创建于 2002 年，专门致力于确保各种 Web 服务实现的互操作性。WS-I 发起了若干工作组（working groups）以解决 Web 服务规范之间的不兼容性。WS-I 提出了一些称作*概要（profile）*的规范（这些规范为其他规范提供了公共的解释），并为 Web 服务厂商提供了用以确保符合 WS-I 规范的测试工具。

[6] HTML 和 XML 都是由 SGML 派生而来的。（译注：SGML 是一个较为复杂的标记语言，为便于在 Web 上应用，HTML 和 XML 各自取了 SGML 的子集。）

[7] 译注：之前的名称为 SGML-Open，于 1993 年成立。详见 OASIS 官方主页的介绍：*http://www.oasis-open.org/who/*。

- IETF（Internet Engineering Task Force，因特网工程任务组）——负责定义和维护基本的 Internet 规范，如 TCP/IP、SSL/TLS 等。这些规范与 Web 服务的关系不是那么直接。TCP/IP 是用于 HTTP 传输的最常见的通信协议，Web 服务中会用到基本的 IETF 安全机制。IETF 与 W3C 共同协作进行 XML Signature 规范的制订。

- JCP（Java Community Process，Java 社团过程）——由 Sun 公司创建，致力于促进 Java 的普及并控制其发展。JCP 主导着多个 Java Specification Requests（JSRs），JSRs 为 Web 服务定义各种 Java APIs，例如用于 SOAP Java 绑定的 JAX-RPC、用于 XML 数据绑定的 JAX-B 以及用于 WSDL 的 Java APIs 等。

- OMG（Object Management Group，对象管理组织）——最初致力于创建和促进 CORBA（Common Object Request Broker Architecture，通用对象请求代理体系结构）规范，OMG 主导着定义 WSDL 到 C++以及 CORBA 到 WSDL 的映射的规范。

Web 服务的标准化进程自 SOAP 1.1 规范于 2000 年年中提交给 W3C 开始拉开序幕。之后，带附件的 SOAP（SOAP with Attachmens）、XKMS、WSDL 等也被提交给 W3C。与此同时，一个非公开组织提出了 UDDI，而后 UDDI 被提交给 OASIS。其他提交给 OASIS 的主要规范包括：WS-Security、WS-BPEL、WS-CAF 及 WS-Notification 等。最近，WS-Addressing 和 WS-Policy 也被提交给了 W3C，这表明 W3C 又恢复了成为多数主要规范的大本营的势头。

ebXML 系列规范也出自 OASIS，这些规范与 Web 服务规范栈有很大程度的重合。Web 服务和 ebXML 都用到了 SOAP，但除此以外，它们的规范栈别无相似之处。ebXML 有自己的注册库（registry）和自己的 Web 服务编制（orchestration）语言。

Standardization and Intellectual Property Rights
标准化与知识产权

Web 服务的标准化面临着一个困难，即没有哪个单独的标准化组织真正处于明显的

领导地位。W3C、OASIS、WS-I、IETF 和 OMG 等瓜分了 Web 服务规范的制订。这样的话，规范何以成为一个被采纳的标准？如果有人能为此提出一种有效的解决办法，无疑他将会成为百万富翁。市场的接受和采纳将最终导致差异，这表明：在厂商的投入程度和赢得用户方面，经济因素是至关重要的。Web 服务厂商们常常先在小规模的团队中非正式地启动规范制订工作，以快速完成规范的制订，然后自行发布规范，接着再将规范提交给某个标准化组织。Microsoft 的 SOAP 1.1 就是这样炮制出来的，其他厂商（诸如 BEA、HP、IBM、IONA、Oracle、SAP、Sun、Tibco 和 WebMethods 等）也都通过这种方式推出了很多规范。然而，在向标准化组织提交规范的时候，这种规范制定方式存在的一个重要问题就突显出来了。具体来说就是：必需要解决知识产权相关的问题（包括版权和专利等）。这是规范制定过程中的重要步骤，即使标准化组织未对被提交的规范做多少改动。规范只有被提交给某个开放的标准化组织，并且解决了知识产权问题（至少在公开声明上看起来是这样），这个规范才有可能被真正地广泛采纳。不经过这一步，非最初参与规范制订的软件厂商们将难以获知规范的变动，这会令它们在产品（如果是基于该规范的话）中的投入作废，并且还要面对可能产生的知识产权许可费（使用费或其他形式的费用）的问题。

Specification Composability
规范的可组合性

前面提到过，SOAP 和 WSDL 规范在设计时就预留了扩展点，以备添加补充特性的需求。多个补充规范可以在一个 SOAP 报头（SOAP header）中组合。下面是一个例子：

```
<S:Header>
  <wsse:Security>
  ...
  </wsse:Security>
  <wsrm:Sequence>
```

```
...
 </wsrm:Sequence>
</S:Header>
```

在这个例子中，SOAP 报头中增加了对安全性（security）和可靠性（reliability）的扩展。安全性报头（security header）中一般包含安全令牌（security token）（可用于确保消息来自受信源（trusted source））等信息。可靠性报头（reliability header）中通常包含一个消息 ID 和顺序号，用以确保消息（或一组消息）被可靠地收到[8]。

可以注意到，上面的示例代码中为安全性和可靠性报头分别使用了不同的命名空间（namespace）[9]：`wsse:` 和 `wsrm:`。这样做，是为了避免可能产生的命名冲突。在一个 XML 文档中，不允许同一名称的元素或属性具有两种类型，而 SOAP 消息是一个 XML 文档，因此必须遵从这一规定。命名空间前缀（namespace prefixes）为元素名称（element names）和属性名称（attribute names）提供了唯一的限定符（unique qualifier），因此避免了名称冲突。命名空间保证了各种对 SOAP 报头的扩展不会相互影响。

增加补充特性不一定需要对现有 Web 服务进行修改——更确切地说，在许多情况下，可以把补充特性加入到 SOAP 报头（SOAP header）中，而不必修改 SOAP 主体（SOAP body）。不过，对执行环境和映射层（mapping layers）的修改可能是必要的。当然，在对 SOAP 报头进行扩展时，必须确定执行环境是否支持、以及如何支持该扩展；否则，对 SOAP 报头的扩展将不起作用。

对 Web 服务的扩展增加了执行环境中 SOAP 处理器（SOAP processors）的职责。在生成 SOAP 消息的过程中，可以利用与 WSDL 契约（WSDL contracts）关联的策略断言（policy assertions）来确定是否要在 SOAP 报头中加入某些信息，以帮助执行环境选用恰当的传输协议或约定诸如事务协调（transaction coordination）之类的特性。

如图 1-12 所示，为 Web 服务调用增添各种补充特性的结果，是增加了在发出消息

8 译注：可靠的消息传递的含义是，确保消息能够收到且只收到一次，并且必须按序收到。
9 译注：此处的“namespace”是 XML 术语，常见的译法有“命名空间”、“名称空间”、“名字空间”等。这里采用“命名空间”的译法。

之前和收到消息之后的额外处理。后面章节中将对这些补充特性作更详细的介绍。这些补充特性可能会与某个业务流程引擎的需求相关，而且还可能需要注册库（registry）的支持。

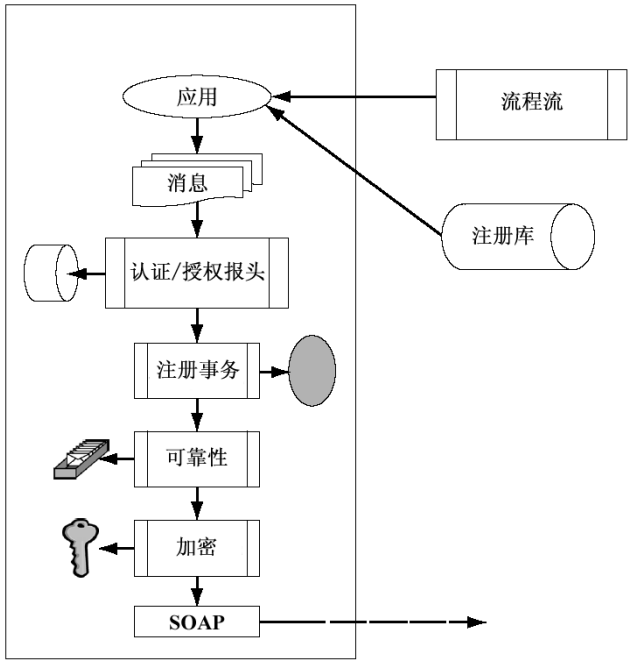

图 1-12 为 SOAP 增加补充特性

Composibility and Complexity

可组合性与复杂性

可以想象，补充的 Web 服务规范之间的可组合性（composability），令它们可以逐渐使用或发现新的概念与特性。作为 Web 服务规范开发的事实领袖，IBM 和 Microsoft 显然希望避免 Web 服务规范过于庞杂。Web 服务的价值很大程度上来自于其相对简单性。CORBA 就是因为过于复杂和难于使用，而常受到批评。当然，CORBA 比它之前的技术

要简单一些，但相对于 Web 服务来说，则仍显得复杂。复杂性（complexity）的解决办法在于: 需要具有足够本领来有效地利用技术的人。而 IT 成本的最大部分在于人力方面，因此复杂性通常意味着需要更大的项目开支。为了让 Web 服务遵守承诺，IBM 和 Microsoft 希望在为基本的 Web 服务规范增添复杂特性时，能够保持其简单性。有人认为 Web 服务就是用于支持聚合各种新特性的，因此可以扩展现有的应用，而不必修改它。然而，这只是一个未经验证的断言，因为完全实现了各种补充规范的产品尚未问世，而且还根本不清楚是不是在产品中将以同样的方式来实现这些补充特性。另外，还没有一个完整定义了这些补充特性如何共同工作的 Web 服务架构。还有，如果你收到了一个 SOAP 消息，其中包含有安全性、可靠性及事务性等报头，你会先处理哪一个？安全性报头，还是可靠性报头？或者事务性报头？确实没人知道。如果觉得这些过于复杂，人们可以回过头来使用 XML over HTTP 这种最简单的方式，并对这些扩展进行手工编码（hand code）。

Metadata Management
元数据管理

Web 服务中的元数据（metadata）是关于 Web 服务的描述信息，这些描述信息用于构造消息主体（body）（包括数据类型和结构）和报头（header），服务请求者需要根据这些描述来调用服务。服务提供者发布元数据（metadata）；服务请求者发现元数据，并据此构造可被服务提供者正确处理的消息。

在进行服务调用时，不仅要发送数据类型和数据结构，还要发送关于补充服务质量的信息，比如安全性、可靠性、事务性等信息（如果有的话）。如果消息中缺少某些这类信息，消息处理可能会失败。

元数据规范包括：

- **XML Schema**——用于定义消息中的数据类型与数据结构，用于表达策略信息。
- **WSDL**[10]——用于将消息和消息交换模式（Message Exchange Patterns，MEPs）与服务名称及网络地址相关联。
- **WS-Addressing**——用于包含与端点相关的端点寻址属性（endpoint addressing properties）和端点引用属性（endpoint reference properties）。许多其他补充规范需要 WS-Addressing 来支持定义通信模式中的端点寻址与端点引用属性。
- **WS-Policy**——用于将服务质量（quality of service）需求与 WSDL 的定义相关联。WS-Policy 是一个包含安全性、可靠性、事务性等策略断言的框架（framework）。
- **WS-MetadataExchange**——用于查询和发现与 Web 服务关联的元数据，包括获取 WSDL 文件及相关 WS-Policy 定义的能力。

基于 Web 服务的 SOA 在服务绑定（service binding）上与基于 J2EE 或 CORBA 的 SOA 有所不同。Web 服务是通过服务发现（可以是动态的），而不是通过引用指针或名称进行绑定的。如果服务请求者能够理解服务提供者给出的 WSDL 及相关联的策略文件，那么服务请求者便可动态生成 SOAP 消息，以执行提供者的服务。所以，各种元数据规范，对于基于 Web 服务的 SOA 的正确执行来说，是至关重要的。

寻址（Addressing）

寻址（Addressing）是补充的 Web 服务规范的重要需求之一，因为 Web 上不存在 Web 服务端点地址（endpoint address）的目录。在几乎所有的消息交换模式（MEPs）中（除去那些最简单的），SOAP 消息中都必须包含端点地址信息。WS-Addressing 取代了之前提出的 WS-Routing 和 WS-Referral。

如果没有寻址方案，在服务提供者收到你发送的 Web 服务请求时，一般只知道一个地址，即请求者的返回地址，而该地址是仅在当前会话（session）期间有效的。回复过

[10] 译注：WSDL 读作“wizdle”。

程中一旦发生错误，将无法进行重试——基本上，一旦发生通信错误，请求者的地址就丢失了。

并且，也没有一种用于指定“除请求者地址以外的其他返回地址”的好方案。最终的结果是，无法为复杂的消息交换模式（MEPs）或多传输（multi-transport）[11]定义地址规划（address schemes）或标识端点地址（endpoint addresses）。

策略（Policy）

策略（policy）在表达 Web 服务的补充特性时是必要的。有了策略，请求者便可对 Web 服务的安全性、事务性及可靠性作出规定，而不是仅在 WSDL 文档中提出关于消息的数据需求。

服务提供者可以用一组机器可读的（machine-readable）的断言（assertion）来表达策略,并向服务请求者提出：必须符合这些断言才能调用其服务。服务是否对安全性有要求？是否支持事务性？对于运行时间长而且复杂的交互，断定它是否支持事务、或者事务能否跨越多个 Web 服务是非常重要的。

WS-Policy 对于实现补充特性的互操作来说是必要的，因为策略断言是请求者唯一能够用以确定“提供者是否要求某些或全部补充特性”的方式。以安全性为例，各个服务提供者支持的安全令牌（security token）种类（比如 X.509 或 Kerberos 等）会有所不同。WS-Security 是一种能够携带任何令牌类型的开放框架。但是，如果提供者没有声明它所期望的令牌类型，请求者便无法确信提供者的要求是什么。

服务请求者在决定要调用提供者的服务时，确定提供者是否支持可靠性或事务性也是很重要的。比如，也许你想知道提供者的服务是否支持在提交失败后重试，以及是否会在成功接受消息后发送确认消息。最后，你可能还想知道是否要向服务提供者发送一个事务上下文（transaction context），以便提供者的服务加入事务。

[11] 译注：multi-transport 指的是可以通过多种传输协议（比如 HTTP/S、MQSeries、IIOP、JMS 等）来传输 SOAP 消息。

获取元数据（Acquiring Metadata）

请求者将用 WS-MetadataExchange 或其他类似机制，直接查询 WSDL 及其相联的策略文件，以获取它所需要的元数据（metadata）。WS-MetadataExchange 使用了 WS-Addressing 中的一个叫做“活动（action）”的特性来访问服务提供者发布的元数据。WS-MetadataExchange 用于提供关于 Web 服务描述的任何信息——在大多数应用中，它基本上可以代替 UDDI 了。

开发者是否使用 UDDI 是不一定的。补充 Web 服务规范之间的互操作，需要相关的元数据管理机制的支持，而公共的 UDDI 没有提供这样的机制，而且请求者可能需要 WS-MetadataExchange 来确保它们拥有与提供者进行补充特性的互操作所需的信息。

Security
安全性

在 Web 服务规范栈的每一层都涉及安全性（security）问题，安全性需要利用各种机制来预防分布式计算过程中的各种挑战与威胁。在某些特定威胁或多种威胁组合的情况下，需要将这些机制加以组合才能得以预防。在 Web 服务和 SOA 中，评估是否需要在网络层、消息层以及对消息中数据进行保护是特别重要的。

基本的安全性保护是建立在加密（encryption）、认证（authentication）和授权（authorization）机制之上的，而且通常还要进行问题跟踪的详细记录。业界已在 WS-Security 这一规范框架上取得了一致，虽然仍有一些工作正在进行之中。

WS-Security 用于为 Web 服务消息提供端到端（end-to-end）的消息层安全性。常见的基于 HTTP 的安全机制（比如 SSL）仅为传输层提供点对点（point-to-point）的保护。有时通过使用其他传输映射（比如 CORBA 的 IIOP 或 WebSphere MQ 等）也可以获得额外的安全性；但是正如其余的补充特性那样，Web 服务安全规范是针对 HTTP 这一缺省

的、用于大多数情况的传输协议而制订的，它可以很容易地应用于其他传输协议。

WS-Security 报头（headers）可以包含像 Kerberos 票据（tickets）和 X.509 证书（certificates）这样的强身份认证（strong authentication）格式，也能使用 XML Encryption 和 XML Signature 技术对消息内容作进一步保护。尽管用于安全断言标记语言（Security Assertion Markup Language，SAML）的 WS-Security 授权概要（authorization profile）还在开发之中，SAML 仍可独立用于授权信息的交换中。

IBM、Microsoft 及 Verisign 等公司另外提出了一些对 WS-Security 作进一步扩展和补充的规范，比如：

- **WS-SecurityPolicy**——通过定义安全断言（security assertions）对 Web 服务的需求作详细描述，以供服务请求者参照。
- **WS-Trust**——定义如何通过从受信源（trusted sources）处获取必要的安全令牌（security tokens）（例如 Kerberos 票据），对安全系统建立整体的信任。
- **WS-SecureConversation**——定义如何为一个安全会话（secure session）建立和维护持久的上下文，以便对 Web 服务进行多次调用、而不必每次都进行大开销的身份认证。
- **WS-Federation**——定义如何创建跨越多个安全区域（security domains）的联合会话（federated session），以便在经过单次身份认证后即可使用部署在多个安全区域内的 Web 服务。

由于 Web 服务都是 XML 应用，而 XML 自身存在着安全问题（XML 是以人类可读的文本在 Internet 上发送的），因此，用基于 XML 的安全技术来保护装入 SOAP 消息之前和之后的 XML 数据也是很重要的。这些技术包括：

- **XML Encryption**——通过使用各种受支持的加密算法，来提供部分或全部 XML 文档的机密性（confidentiality），以确保文档内容不会被窃听或被未授权的人读取。

- **XML Signature**——通过使用各种加密和签名机制来提供完整性（integrity），以确保服务提供者能够确定文档有没有在传输过程中被修改，并确保文档被收到且只收到一次。

XML Encryption 和 XML Signature 既用于保护 Web 服务的元数据，也用于保护 Web 服务的数据。

Reliability and Messaging
可靠性与消息传递

消息传递（messaging）包括 SOAP 以及它的各种消息交换模式（Message Exchange Patterns，MEPs）。在高级的消息传递方面，尚无统一的规范集合被业界认可，这里就不介绍了。而可靠性与消息通知方面的各个规范，本质上工作方式相同，因此这里将二者混合起来进行介绍。

一般来说，可靠的消息传递（reliable messaging）是保证一个或多个消息被收到正确次数的机制。可靠的消息传递规范包括：

- WS-Reliability
- WS-ReliableMessaging（由 IBM 和 Microsoft 等制订）

可靠的消息传递用于确保 SOAP 消息（SOAP messages）可以在不可靠的网络（比如基于 HTTP 的 Internet）上进行可靠的递送。

可靠的消息传递是一种用于进行有保证的递送（guaranteed delivery）、无重复的、消息次序正确的 SOAP 消息交换协议。可靠的消息传递的原理是：为一组消息设定相同的 ID，根据消息号将消息编组，并依据顺序号进行排序。

有了可靠的消息传递，便可自动从某些传输层的错误状态中恢复；如果没有可靠的消息传递，应用必须自己解决这类问题。可靠的消息传递还支持在 Internet 上衔接两种不同的私有消息传递协议。

在一般的消息传递领域也有一些扩展的消息交换模式（MEPs）规范，比如事件通知（event notification）、事件发布/订阅（event publish/subscribe）等，它们基本上扩展了 Web 服务的异步消息传递能力。这类规范包括：

- WS-Eventing
- WS-Notification

通知（notification）通过一个常被称作消息代理（message broker）或事件代理（event broker）的媒介（intermediary）进行消息递送。订阅者（subscriber）确定它们希望收到消息的通道（channels）或感兴趣的消息主题（topics），发布者（publisher）将发送消息到订阅者正在侦听的通道或主题。通知（notification）是一种可用来进行广播和发布/订阅的消息传递机制。

Transactions
事务

事务（transctions）指的是：对持久数据（persistent data）的多个操作，要么全部完成，要么什么都不做，就像处理订单或资金转移那样。事务处理技术中最重要的一个方面是：在遇到操作系统或硬件故障之后，将应用恢复到某个已知状态的能力。例如：如果在一次资金转移操作完成（即借记操作和贷记操作都已完成）前发生了一个错误，事务应确保银行账户的余额状态与该资金转移操作开始之前一样。

有些 Web 服务应用可能只需要下层执行环境的事务处理能力，比如应用服务器和数据库所提供的事务性。而另一些 Web 服务应用则要将多个 Web 服务调用（Web service invocations）及 SOAP 报头中的事务上下文（transactional context）组成为一个更大的事务性单元（transactional unit），以便进行跨越多个执行环境的事务协调。

Web 服务事务规范扩展了事务协调器（transaction coordinator）的概念，为 Web 服务修改了通用的两阶段提交协议（two-phase commit protocol），并为更加松耦合的 Web 服务及编制流（orchestration flows）定义了扩展的事务协议。目前大多数执行环境（比如 J2EE、.NET 框架、CORBA 等）中都有事务协调器。Web 服务规范为这些（及其他）协

调器定义了扩展，以便用于基于补偿的（compensation-based）协议和衔接不同软件领域的协调器间（coordinator-coordinator）协议。

协调（coordination）是一种为合成的 Web 服务应用确定一致的、预定义结果的通用机制。协调器模型（coordinator model）包括两个主要阶段：第一阶段为获取阶段，参与合成的 Web 服务各自向协调器登记一个特定的协议（比如两阶段的提交、补偿或业务流程等）；在第二阶段，当参与合成的各个服务分别执行结束后，协调器启动约定好的协议来完成事务。如果发生错误，协调器应启动恢复协议（recovery procotol）（如果有的话）。

Web 服务的事务规范包括：

- BEA、IBM 和 Microsoft 提出的 **WS-Transactions** 规范系列：
  - **WS-AtomicTransactions**——为短期执行的业务流程定义了两种两阶段提交协议（two-phase commit protocol，2PC）：易失的两阶段提交（Volatile 2PC）和持久的两阶段提交（Durable 2PC）。
  - **WS-BusinessActivity**——为长期执行的业务流程定义了基于试验提交和基于补偿（compensation-based）的撤销协议。
  - **WS-Coordination**——定义了用于上述两个可挂接的协议（pluggable protocol）的协调器。
- OASIS 的 WS-Composite Application Framework（WS-CAF）：
  - **WS-Context**——定义了一个独立的、用于一般上下文（即用于非事务协议上下文，如安全、设备、网络 ID 或数据库及文件 ID 等）的上下文管理系统。
  - **WS-CoordinationFramework**——为基本的上下文规范和 WS-TransactionManagement 中的可挂接事务协议（pluggable transaction protocol）定义协调器。
  - **WS-TransactionManagement**——定义了三个可挂接到 WS-CoordinationFramework 中的事务协议：ACID，Long-Running Actions（LRA）和业务流程管理（BPM）。

上述两个规范系列都以一种支持挂接的协调器为中心。它们都定义了原子事务（即两阶段提交）和基于补偿的事务。WS-CAF 将上下文管理作为单独的规范，并增加了一个专门用于业务流程管理（BPM）的事务协议。

Orchestration
Web 服务编制

Web 服务最终将被发布用于给定 IT 环境中的大多数软件系统和应用，并要跨越多个组织的 IT 环境。在具有异常处理、分支和并行执行的长期运行的（long-running）业务流程流中，可以用编制引擎（orchestration engine）创建较复杂的交互模式，而不是令 Web 服务用 SOAP 和 WSDL 支持下的一种或多种消息交换模式（MEPs）进行彼此调用。但是要实现这一点，编制引擎必须保持上下文，并提供多个服务之间的关联机制。

可以将 Web 服务编制本身作为 Web 服务来发布，通过它提供封装一组其他 Web 服务的接口。通过使用 MEPs 的组合以及编制机制，整套应用可以由不同封装层次的（从封装单个软件模块的，到封装复杂的 Web 服务流的）Web 服务建立起来。

业界已经对 OASIS 的 Web 服务编制规范 WS-BPEL 取得了一致的认同。WS-BPEL（Web Services Business Process Execution Language，Web 服务业务流程执行语言）假定 Web 服务是用 WSDL 与策略断言定义的。

通常，一个流程流因某个 XML 文档的到达而启动。所以，在对流程流的入口点进行建模时，往往采用面向文档式的 Web 服务（document-oriented Web services style）。通常，流程流中的各个任务需要取文档中的不同部分加以操作（比如检查不同供货商的库存量）。这意味着，流程流中的各个步骤可以通过组合请求/响应（request/response）与面向文档的 Web 服务来实现。

WS-BPEL 规范与其他扩展规范的不同之处在于，它所定义的是一种可执行的语言，而且能够与各种促使业务流程自动化的软件系统相兼容。Web 服务编制，通过说明性的（declarative）方式（而不是编程的方式）表达了进行 Web 服务合成的需求；而其他的 Web 服务规范，则大多是用 XML 来表示现有各种对 SOAP 报头进行扩展的分布式计算功能及特性。

W3C 的 Web 服务编排定义语言（Web Services Choreography Definition Language，WS-CDL）是 Web 服务编制领域的另一个规范。编排（choreography）被定义为在多个外部交易伙伴之间建立形式化关系。WS-CDL 与 WS-BPEL 的区别之一是，前者不要求所有被集成的端点（endpoints）都有 Web 服务基础设施（infrastructure）。

Strategic Value of Orchestration
Web 服务编制的战略价值

有人说，Web 服务编制即 Web 服务获取战略价值的所在。Web 服务有其内在价值，因为开发人员可以相对轻松地解决不同类型软件间的互操作问题。Web 服务编制可以各种不同的方式使用：从创建合成服务，到创建成熟的业务流程管理。为 BPM 使用 Web 服务编制显然是较困难的，因为它要求对企业的业务流程有较深入的理解。总之，人们总期望能够在编制层找到解决困难问题（比如数据类型与结构的不兼容，数据的语义匹配，及关联多个 Web 服务的执行结果等）的解决方案。要证明这些问题是否能够在编制层得到解决、自动化业务流程管理是否正是各机构所想要或需要在部门和企业级配置的，还需要时间来证明。

Summary
小结

现在我们可以看到，要新建基于 SOA 的企业应用，Web 服务标准（包括核心的和补充的 Web 服务标准）对于创建和维护这样的 SOA 是很有裨益的。这些应用通常被称作合成应用（composite application），因为它们要组合多个服务。

我们已经看到，目的并不是 SOA 本身，而只是起点——它是通向更优的 IT 环境的一套规划与指导；它是一种基础设施的规划，这种基础设施不但可以确保 IT 与业务一致，而且可以通过资产（assets）的重用、快速应用开发及多渠道服务减少开销。

Web 服务的核心标准已经取得了初步成功，下一步是要定义各种补充的特性与功能，以支持越来越多的应用种类。

面向服务（service orientation），提供了一种与面向对象不同的思考角度与方式。从面向过程的计算到面向对象的计算是一个意义重大的转变，到 SOA 的转变也同样具有重大意义。服务并不是要取代其他技术，而是一种补充技术。它尤其适合于解决多种执行环境（比如面向对象的、面向过程的及其他系统）间的互操作性。