使用面向服务体系结构和基于组件的开发来构建 Web 服务应用程序

Rational Software 白皮书

Alan Brown,
Simon Johnston,
Kevin Kelly

Rational®
the software development company

内容

简介

构建企业级软件系统是一项复杂的任务。尽管公司拥有数十年的先进技术，但如今的信息系统往往要求公司在设计、构建和改进关键任务软件解决方案方面有所突破。特别在于，很少有新系统是从头开始设计的。相反，软件设计工程师的任务一般都是延长现有解决方案的生命，这是通过描述现有数据储存库的新业务逻辑、通过新的渠道（如 Internet 浏览器或手持设备）展示现有数据和事务、集成先前断开连接的系统及支持覆盖业务活动等等来实现的。

为了协助软件开发人员，现在诸如 Microsoft 和 IBM 之类的供应商提供了商业软件基础结构产品。这些产品成为了他们分别在 .NET 和 WebSphere 产品线提倡的软件开发方法的主干部分。两种方法都强调根据分布式服务来组合系统。但是，根据服务来构建企业级解决方案有什么新颖之处呢？基于组件的系统的经验教训如何应用于构建基于服务的构架（SOA）呢？构建高质量系统以部署到新一代软件基础结构产品的最佳方法是什么？这些重要的问题是本文讨论的主题。

近年来，软件工程团队对支持大型软件系统可从独立、可复用的功能集组合而成这一概念的设计方法、流程和工具给予了很大关注。有些功能可能已可使用，可以在内部实现或者可以从第三方获取，而其它功能可能还需要创建。在这些情况下，整个系统的构思和设计必须将所有这些元素集合起来，形成单一而有条理的整体。今天，这一概念在*基于组件的开发（CBD）*中得到了例证 — 这是通过 Microsoft .NET 平台和 Java 2 Enterprise Edition（J2EE）标准这种受 IBM WebSphere 和 Sun 的 iPlanet 等产品支持的技术方法来实现的。

另一注意事项是操作系统 一般将分布在许多机器上，以提供性能、可用性和可伸缩性。企业解决方案必须协调硬件集合上的功能执行情况。构思这种系统的一个方法是将它看作由一组交互的*服务*构成。每个服务均使用户可使用一组明确定义的功能。就整体而言，系统是作为这些服务之间的一组交互作用来设计和实施的。将功能作为服务来展示是灵活性的关键。这允许其它功能（也许它们本身是作为服务来实施的）自然地利用其它服务，无论它们的实际位置如何。系统随着新服务的增

加而发展。生成的*面向服务体系结构（SOA）*定义系统的组成服务，描述为实现某个行为而进行的服务间交互，并将这些服务映射到特定技术中的一个或多个实施。

虽然服务包括了业务功能，但还是要求有某种形式的服务间基础结构，以方便服务交互和沟通。这种基础结构可能有不同形式，因为服务可以在单机上实施，分布在局域网中的一组计算机上，或者更广泛地分布在数个公司的网络中。一个特别有趣的情况是这些服务将 Internet 用做通信机制。产生的 *Web 服务*共享更普通服务的特征，但由于使用公开、不安全、低保真的机制进行服务间交互，它们要求注意特殊事项。

到目前为止，软件行业已对用于实施 Web 服务及其交互的底层技术给予了很大关注。但是，就设计 Web 服务以简单地组合为企业级解决方案而言，其最恰当方法又成为业内关注的问题。相反，令人惊讶的是，很少有人对设计由 Web 服务组成的企业级软件解决方案的适当做法和工具加以关注。与任何复杂结构的设计一样，由于许多得到很好了解的设计技巧、结构模式和风格支持早期构架决策，就产生了高质量的解决方案。这些模式针对的是常见的服务问题，如可伸缩性、可靠性和安全性。

本文为进一步理解企业级软件解决方案的服务和面向服务体系结构提供了环境。特别是，它探索了与更确定的软件组件概念相关的服务，并描述了当前基于组件的开发活动如何为实施面向服务体系结构提供经尝试和测试的基础。需要强调的是，基于接口的设计是服务和组件设计的关键，有人认为，服务和组件设计所展示的接口均存在某些限制和区分标准。统一建模语言（UML）[1] 用作一种工具，来描述逻辑和实施设计以及组件和服务设计的特定模式。

最后，本文还关注 Web 服务，通过 Web 服务来在一般意义上探讨与实施服务相关的问题。特别地，本文还研究如何将面向服务体系结构的一般指导信息解释为 Web 服务的具体设计模式。本文中所述的所有模式都已通过 Rational® XDE™ 的独特模式和代码模板功能来实施，并通过 Rational Developer Network[8] 发布。

什么是面向服务体系结构？

那么什么是面向服务体系结构（SOA）呢？就本质而言，它是设计软件系统的一种方法，目的是通过已发布和可发现的接口向最终用户应用程序或其它服务提供服务。在很多情况下，众多服务提供了展示各种业务功能的更佳方法，也就提供了开发支持业务流程的应用程序的极好方法。

面向服务体系结构并不是新概念；由于有了新兴的 Web 服务技术，面向服务体系结构在目前是很重要的。例如，以下是从 2000 年出版的一本书中摘录的内容，其中描述了面向服务体系结构的价值：

> *面向服务解决方案应用程序必须作为几组独立的交互服务来开发，向它们的潜在用户提供定义明确的接口。类似的，还必须提供支持技术，使应用程序开发人员能够浏览服务集，选择关注的服务集，并将它们组合起来创建所需的功能。[2]*

出于本文档起见，我们将考虑服务的以下定义：

> *服务一般作为粗分型、可发现的软件实体来实施，该实体作为单个实例而存在，并通过松散连接（通常为异步）的、基于消息的通信模型与应用程序和其它服务进行交互。*

在许多方面，服务的术语都与用来描述基于组件的开发的术语大致相同；但是，如下面的图 1 所示，可使用特殊术语来定义 Web 服务内的元素。

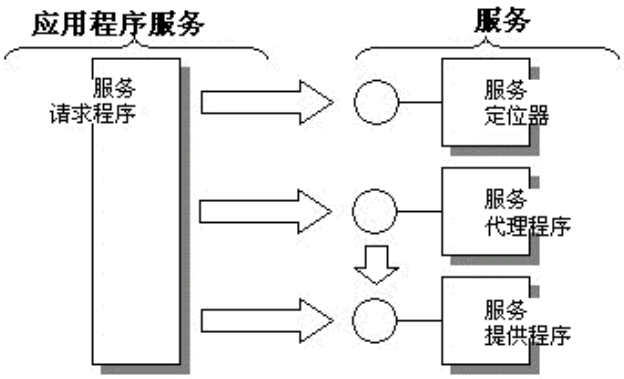

图 1 — 服务术语

- **服务** — 一个逻辑实体；由一个或多个已发布接口定义的合同。
- **服务提供程序** — 实施服务规范的软件实体。
- **服务请求程序** — 调用服务提供程序的软件实体。以往，这被称为“客户端”；但是服务请求程序可以是最终用户应用程序，也可以是其它服务。
- **服务定位器** — 一种特定的服务提供程序，它充当注册表并考虑到查找服务提供程序接口和服务位置。
- **服务代理程序** — 一种特定的服务提供程序，它可以将服务请求传递给一个或多个其它的服务提供程序。

这种服务描述以及服务的使用环境提出了一系列限制条件。而且，有效使用服务就意味着一些高级别的最佳做法。以下是有效使用服务的一些主要特征：

- **粗分型** — 频繁对服务实施操作，以包括更多的功能，并对较大的数据集（与组件接口设计相比）执行操作。
- **基于接口的设计** — 众多服务实施独立定义的接口。这样做的好处在于，多个服务可以实施公共接口，而且单个服务可以实施多个接口。
- **可发现** — 服务需要在设计和运行这两个时期内查找，不仅按独特标识来查找，而且按接口标识和服务种类来查找。
- **单个实例** — 基于组件的开发根据需要将组件实例化，与它不同，每个服务总是与多个客户端进行通信的单个运行中的实例。
- **松散连接** — 众多服务使用降低依赖关系的、断开连接的、基于消息的标准方法（如 XML 文档交换）连接其它服务和客户端。
- **异步** — 一般情况下，众多服务使用异步消息传递方法；但这并不是必需的。实际上，许多服务将不时地使用同步消息传递。

这些条件中有一些（如基于接口的设计和可发现性）也在基于组件的开发中使用；但是，所有这些属性加起来才能将基于服务的应用程序与使用组件构架（如 J2EE 或 .NET）开发的应用程序区分开来。

基于接口的设计

在组件和服务开发中，接口设计的完成使软件实体能够实施并展示其定义的主要部分。因此，“接口”的想法和概念对于基于组件的系统和面向服务系统中的成功设计都很关键。以下是一些与接口相关的主要定义：

- **接口** — 定义一组经过逻辑组织但不提供任何实施的公共方法特征符。接口定义了服务请求程序与提供程序之间的合同。接口的任何实施都必须提供所有方法。
- **发布的接口** — 可独特标识的接口，可通过注册表由客户端动态地发现。[3]
- **公共接口** — 可供客户端使用但不发布的接口，因此要求具备客户端部分的静态知识。
- **双接口** — 接口经常是成对开发的，这样一个接口就依赖于另一个接口；例如，客户端必须实施接口才能调用请求程序，因为客户端接口提供某种回调机制。这个概念是按 Web 服务提出的。

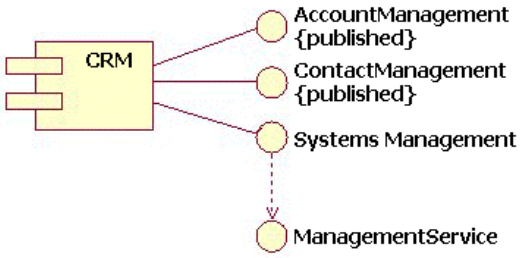

图 2 — 实施的服务

图 2 显示客户关系管理（CRM）服务的 UML 定义（表示为 UML 组件），客户关系管理实施接口 AccountManagement、ContactManagement 和 SystemsManagement。这些接口中只有前两个是已发布的接口，虽然最后一个是公共接口。请注意，SystemsManagement 接口和 ManagementService 接口形成双接口。CRM 服务可以实施任何数量的这类接口，而且正是服务（或组件）的这种根据客户端以多种方式行动的能力，允许很灵活地实施行为。甚至可能向特定类的客户端提供不同服务或其它服务。在某些运行时环境中，这样的功能还用于支持单一组件或服务上同一接口的不同版本。

接口行为

采用诸如 Java 或 C# 之类语言或 IDL 之类语言的接口定义仅提供一组方法特征符。这种定义提供“内容”，而不提供关于“方法”的任何指导信息。例如，考虑图 3 中的“安全性”接口。很明显，调用该接口的实施的客户端能够调用三个公共方法中的任一方法 — 它们能吗？

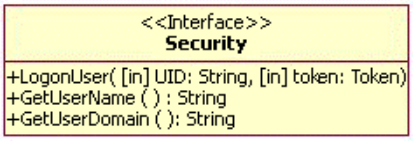

图 3 — UML 中的接口

通过简单地定义“内容”，我们不能说明客户端要等到用户登录后才能调用 GetUserName( ) 或 GetUserDomain( )这一事实。以下状态机说明了这一依赖关系或行为。这种限制条件经常包括在有关基于接口的设计的文献中；但是它在任何编程语言中都不受支持，所以无人能确保接口的实施者遵守任何行为规范。

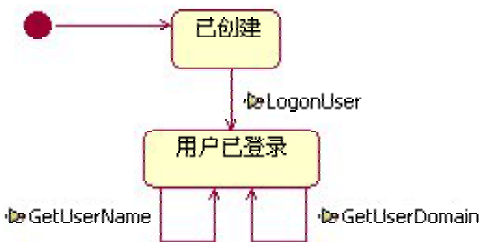

图 4 — 接口行为

但是，众多公司倾向于使用面向服务系统，寄希望于这样的系统能更容易地集成和编排，可通过服务的协作来实现业务流程。结果，定义接口行为的概念以及更重要的各组相关接口行为越来越受到业界的关注。不幸的是，当前很少有这样的标准方法。

可能的一种方法是，使用以标准化语言（如 UML）定义的设计模型（例如通过本文介绍的设计模型）来记录服务接口之间的相互依赖关系。这样的模型可以进行共享、推广，并用来带动新兴的特定标准。

此外，Rational 倡议了 Reusable Asset Specification（RAS），后者为可能应用于该问题的资产提供封装和共享机制。例如，使用 RAS 机制为某一服务分发详细信息时，还可以封装描述其行为的模型。在这样的模型中，就可以使用序列图来显示接口上的众多调用之间的必需的交互。

设计面向服务系统

在软件工程的任何新的开发行为中，很容易假设：可以应用在先前的项目中起过作用的相同技巧和工具。我们已经提到，组件和服务虽然类似，但并不相同；它们的设计标准和设计模式都是不同的。本节将讨论重要的实际结果。*底线在于，并非每个好的组件在转换成服务后都能成为好的服务。*

将应用程序设计分层

这种用旧解决方案解决新问题的倾向并不新奇。类似的，当开发人员开始创建基于组件的系统时，他们尝试让自己拥有面向对象开发的经验（具有类似的问题）。有了更多的经验，就可以理解，面向对象技术和语言是实施组件的极佳方法，可是我们必须知道通过决策和实施所作出的折衷。折衷涉及到实施多形态行为的继承和聚合或者重新设计类库，做到能在各组组件中使用，而不是作为单一 C++ 应用程序的基础。

类似的，我们将组件视为*实施服务的最佳方法*，然而必须知道，基于组件的示范应用程序不一定能用作面向服务的示范应用程序。我们一旦了解服务在应用程序构架中扮演的角色，就很可能充分利用贵公司的组件开发人员和现有组件。作出这一转变的关键，就是要意识到面向服务方法意味着另一个应用程序构架层。下面的图 5 说明，当更倾向于作为应用程序的消费者时，技术层可如何应用于应用程序构架来提供更粗分型的实施。用来指代这一部分系统的术语是“应用程序边缘”，反映了通过服务可极好地展示系统外部视图这一事实，以及内部复用和组合使用传统组件设计。

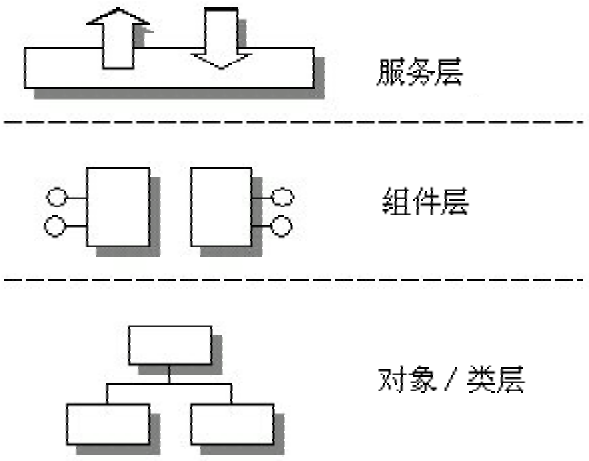

图 5 – 应用程序实施层

一般情况下，从面向对象的思维方式转变为基于组件的思维方式花费 6 到 18 个月时间，在这段时间里开发人员了解这一新技术以及对他们的要求。理想的情况是，转变为面向服务的系统能够更快地进行。此时，开发人员就必须了解开发和复用组件以支持面向服务应用程序所涉及的难题、折衷和设计决策。

客户模型示例

为了说明组件和服务如何交互来实现应用程序，请考虑管理某些客户关系信息的示例（如下图 6 的 UML 类图中所定义）。

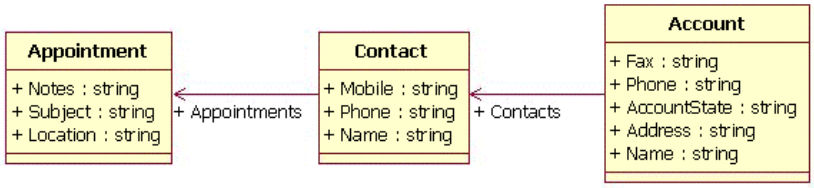

图 6 — 逻辑客户模型

在下面的几节中，我们将描述开发人员如何将这样的逻辑模型转变成实施模型，用于基于组件的应用程序，以及随后用于基于服务的应用程序（请记住，此模型不以任何方式描述系统行为）。将来就会清楚，这些转变步骤中有许多是可以自动生成的。Rational Software 拥有用于对应用程序构架建模、接受和应用这类模式以及在整个开发生命周期内管理这些模型 / 代码工件的工具。

基于组件的设计

我们的逻辑模型示例将如何转变成组件设计（例如，用于 COM 或 J2EE）？ 下图 7 中展示了模型的基于组件的设计。

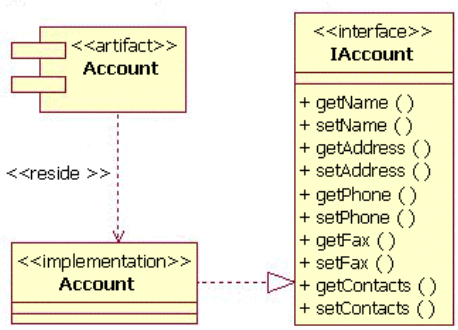

图 7 — 普通组件图

上面的接口有一个*关键*设计特性。关键在于，对于现有的组件平台，有一个通用模式；对于分析类中的每个属性，我们必须提供两个操作，一个操作设置值，另一个操作返回值。对于本地组件，方法调用的开销可以忽略；而对于远程对象，远程过程调用（RPC）机制得到优化，以将开销降到最低。在很多情况下，在应用程序中，客户端仅需要一小部分属性，这样就可以在需要时访问它们。

面向服务设计

上面的设计模型说明了在每个组件实例代表单个对象的情况下考虑组件实施的正确方式；例如，我们的 CRM 数据库中的单个联系人在逻辑上成为了独立组件。这样组件标识就与联系人的标识联系在一起。

但是，对于服务，有着管理一组资源的单个实例，所以它们几乎完全是无状态的。这意味着，我们需要将服务视为可以创建和管理一种或几种实例的*管理器*对象。这将产生一个设计模式，该模式利用了代表实例状态的*值对象*（状态在组件间转换时持续不变的分布式系统中的通用模式），这些对象实际上只处于串行化的状态。这样就得出一个很有趣的结果，如果我们可以定义采用组件定义（如图 7 中的模型）以及将其转换为服务的规则，我们就可以将这种串行化实施为一种模式。使用 Rational XDE，就可能创建和复用这样的模式。

这种从供应程序到请求程序的状态传递，意味着使用单个大操作，而不是使用大量小操作来检索组件状态。现在，这对远程服务（而大多数服务是远程的）的网络使用有着某些涵义，而且在处理较大的值对象时对请求程序的行为有着某些涵义。这还有另一层涵义；请求程序是随某个实体的状态副本提供的，但这个副本是旧的吗？在检索库存报价或天气预报时，我们就知道这些都是过期的，但是我们必须接受这一情况。我们还受到数据类型的制约；库存报价数据的老化速度比天气数据更快。在这里所述的构架中，请求程序必须接受状态副本。有关详细信息，请参阅本文后面的“粗分型交互的影响”部分。

考虑到这些要求，服务的预期外观如何？图 8 中的模型片段显示这样的模式可如何按设计级进行描述，说明组件所发布的接口以及接口操纵的值对象。

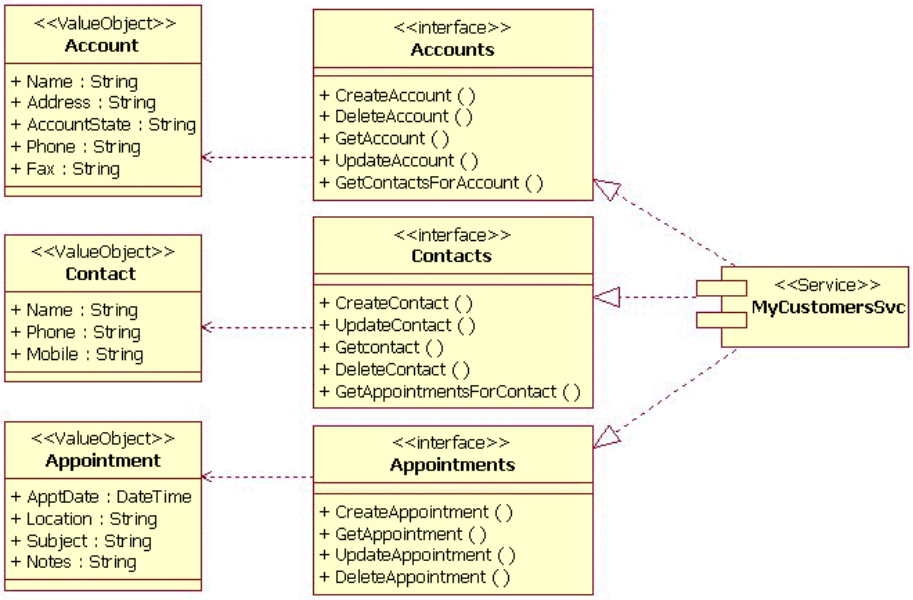

图 8 — 普通的面向服务设计

既然我们知道了可如何设计服务，那么就让我们看看这一特殊设计的一些属性。首先，要清楚，对于给定的交互，值对象传递到请求程序的信息比简单地从供应程序 MyCustomerSvc“获取”或“设置”到请求程序的信息多得多，而且这可能会影响网络带宽。但是，考虑到 Web 服务的本质，实施服务时所使用的协议与组件实施时使用的协议有很大区别，这一点是很清楚的。这样的平台使设计工程师或信息工程师多了一个负担，即要小心选择值对象以及它们的组合来使每个值对象的内容最大化，同时又不会对网络不利。

面向服务设计中的高速缓存

让我们重新考虑在上一节中提出的将“旧”副本的信息从供应程序传递到请求程序的想法。例如，如果我在开发库存组合包管理应用程序，我就不希望为了每个安全性的当前代价而一再要求 Web 服务；传递 3 到 5 个字符的数据可获得安全性，传递 5 到 7 个字符可补偿该代价。这可能会对网络和服务提供程序造成不可接受的负担。请求程序应该做的就是，请求整个组合包的内容，方法是将一系列符号或组合包标识传递到服务并为每个安全性检索所有信息。现在，如果用户只是要求更新单个符号，这似乎就有点小题大作了；但是，现在请求程序可以对结果进行高速缓存，而且如果用户要求更新另一个符号，则可以从高速缓存满足该请求。现在请求程序方面的工作是确定数据的“租用”持续时间。所以，对于组合包，如果我知道库存报价有 20 分钟时间的延误，我就可能希望努力得到 25% 的差额，并将结果高速缓存 5 分钟。

这一模式在信息系统中会一再看到。无论用户何时从订单管理系统检索订单，该用户都会有效地得到订单的副本，因为另一用户可能在同时更新该订单（除非系统封锁对该订单的其它访问权）。现在，如果 Web 服务供应商能够作为其与请求程序交互的一部分实际地确定高速缓存或租用持续时间，那就好了。这些问题在 Microsoft Message Queue Server（MSMQ）和 IBM MQSeries 这样的消息传递系统中得到很好的了解，在这种情况下消息超时和过期时间是得到定期管理的。

我们将在本文档的后面部分重新讨论这个问题，并为请求程序和供应程序的开发提供一些具体的指导信息来处理这个问题。

XML Web 服务应用程序设计

XML Web 服务现在是我们的词汇表的一部分，而且现在有广泛的供应商支持以及越来越多的平台和工具来部署并开发 Web 服务。Web 服务堆栈的元素已经在别的地方有了很好的描述，因此本文档将不再进行完整的回顾。以下定义由 World Wide Web Consortium（W3C）Web 服务体系结构工作小组提供：

> *Web 服务是由 URI 确定的一种软件应用程序，其接口和绑定可由 XML 工件来定义、描述和发现，并支持使用基于 XML 的消息、通过基于 Internet 的协议与其它软件应用程序直接交互。*[4]

让我们简单地考虑 Web 服务中的部分技术如何应用于我们描述面向服务体系结构术语时在图 1 中描绘的图片。

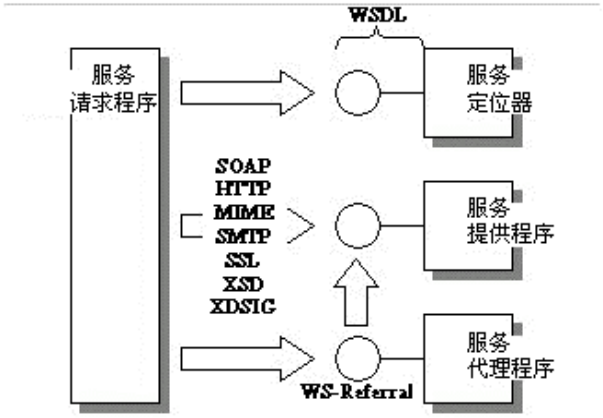

图 9 — XML Web 服务标准

首先说明一个错误概念，即认为所有 Web 服务都使用 XML 消息，具有简单对象访问协议（SOAP），而不是 HTTP － Web 的实际协议。这是不正确的。Web 服务消息可以使用 XML，但它可能会传输二进制数据；它一般使用 SOAP 标题，但不要求将 SOAP 编码用于消息正文；它可以将 HTTP 用于传输，但也可以使用 SMTP 或其它方法。对于 Web 服务的描述和发现，有两个明确的标准：WSDL（Web Services Definition Language）和 UDDI（Universal Discovery, Description and Integration）。

这种格式和传输协议方面的灵活性是 Web 服务的当前问题之一 － 互操作性。考虑到 SOAP 格式、信封、传输协议等等方面的选择，两种实施可如何传送信息呢？跨入这一步的是 Web Services Interoperability（WS-I）组，该组将尝试向行业提供关于使用当前标准和新兴标准的指导信息。供应商也在这方面给予帮助，他们提供灵活的 Web 服务开发环境，如 IBM WebSphere Studio Application Developer Integration Edition，它用多种格式和传输协议创建 Web 服务，这样就可按要求使用最快的组合或正确组合。

Web 服务设计和实施模式

Web 服务实际上并不改变关于应用程序的功能要求的分析和设计流程 － 保险索赔处理应用程序仍然必须处理保险索赔！我们提出的是非功能性要求领域的一套限制条件和潜在问题。以下几部分描述这些可能的缺陷中的一部分。

性能和可靠性

常见的问题是：Web 服务性能、可靠性和可伸缩性所要求的功能是否可由基于 HTTP 和 SOAP（一向很慢而不可靠）的构架提供。首先必须定义“慢和不可靠”，然后必须意识到，即使可靠的传输最终还是要依赖于不可靠的方法。构建和设计企业级解决方案时，我们必须始终记住功能和非功能需求，并确保作出正确的折衷和决策来支持业务目标。

例如，在使用 SOAP over HTTP 时，总有可能构建应用程序级的协议和交互，这些协议和交互提供消息受理和安全性的其它功能。但是，如果要考虑某些服务在相同的安全性和应用程序环境中进行通信，我们可能会考虑使用非 HTTP 方法。请考虑图 10 中的示例。

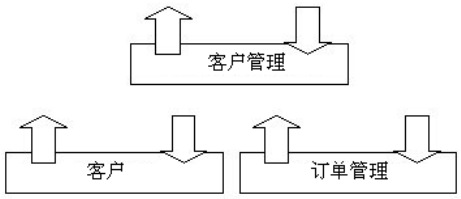

图 10 － 服务依赖关系

基本上，所有外部客户端都与“客户管理”服务进行交互；但它是与两个内部服务进行交互的。这里要作出的决定是：对于这些内部服务通信，我们为什么需要 HTTP 和 SOAP 的灵活性？让我们假设，性能是对“客户管理”和客户之间交互的关键要求。如果这样的话，我们可能会决定使用提供二进制编码格式和更高性能特征的组件 RPC 通信（如 Microsoft .NET Remoting 或者 Java 的 RMI）。另一方面，从“客户管理”到“订单管理”下订单的关键要求是为了保证交付，这样我们就可以使用某种排队技术（如 IBM MQSeries 或 MSMQ）来交付消息，在这种情况下，为获得更高程度的可靠性而牺牲了性能。

很重要的一点是要意识到，即使 Web 服务会展示简单的模型和一套简单、灵活的协议，您也不一定要使用这些选项。正如 WSDL 已经有针对 SOAP 和 HTTP GET/PUT 的绑定一样，向请求程序提供其它选项也是很重要的。例如，单个服务可能会使用消息队列绑定*和* SOAP 绑定来展示消息，这

样请求程序就可选择使用更适当的绑定。在这种情况下，供应程序可能还会提供激励因素，例如，如果使用了消息队列但没有对 HTTP 对话保证任何服务，就会提供有保证的服务级别。

另一个需要及早作出的设计决策是消息交换是等幂的还是可交换的，或者两者都是。*等幂*意味着，如果同一条消息多次到达，并*受到操作*，那么每种情况下都不会有不良后果。*可交换*意味着，两条相关的消息可以按照任何顺序到达，不会有任何不良后果。如果服务的设计可以使消息交换至少确定为等幂，那么不太可靠的传输就是更有吸引力（也更便宜）的选择。

安全性（本文没有讨论）实际是一组从简单便宜到复杂昂贵的选项，同样，性能、可靠性和可伸缩性的设计目标最终也是一组决策：您有多大需要？您可以负担多大的费用？众多服务提供的解决方案和选项与现有开发方法一样多。

通过异步行为和排队实现的可伸缩性

如面向服务体系结构简介中所提到的，使您的 Web 服务在本质上异步是有好处的。由于与 Web 服务相关联的其它传输开销以及期望服务在本质上将是远程的，所以减少请求程序等待响应的时间是很重要的。通过异步调用服务，以及独立的返回消息，我们可以让请求程序在供应程序有机会响应时继续执行。这并不是说同步服务行为不正确，只是经验已经表明异步服务行为更可取，特别是在通信成本很高或者网络延迟时间不可预测的情况下。

请考虑图 11 中的示例。

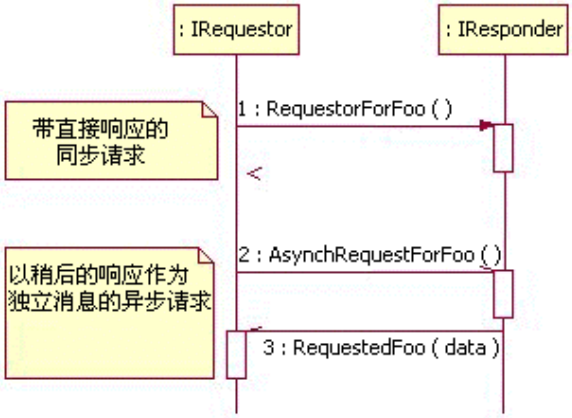

图 11 — 同步对异步

图 11 中描述的行为是实施高度可伸缩 Web 服务的一大步。通过异步调用服务，供应程序就可以使用多个角色线程来处理多个客户端请求。为支持操作的异步模式，所要做的远远不止快速返回到客户端那么简单。首先，您需要指定双接口；请求程序需要将返回地址传递给某个实施接口的服务，该接口可接受返回的消息。这意味着需要管理各方之间对话的状态。要了解实现这一点的各种方法，请考虑非基于 Web 服务的 Web 会话的设计。

但是，这仅在某种程度上是可伸缩的。对于期望极高负荷的服务，我们需要将侦听请求程序的部分与为请求本身服务的部分断开连接。这已经是一个众所周知的模式，其中消息队列用于将服务本身与服务实施断开连接。图 12 中的图显示如何实施提供程序，方式是使用队列将请求与实施断开连接。

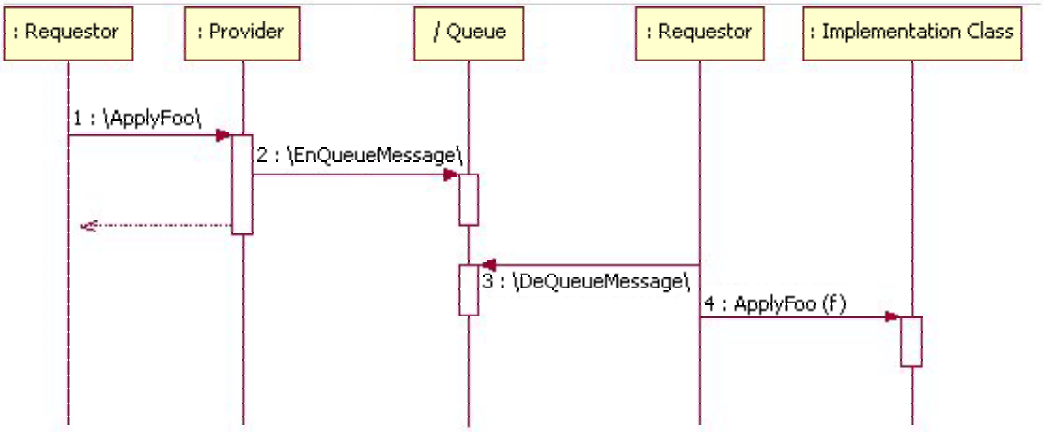

图 12 — 排队的实施

这样的模式可以用以下平台提供的服务在 .NET 和 J2EE 中很容易地实施：MSMQ for .NET 和 Java Message Queue Service（JMS）或用于 J2EE 的消息驱动 Bean。这向开发人员提供了更简单的可伸缩性模型；实施不用处理所请求的一组同步线程，只要添加其它队列监听程序就可从队列中（甚至跨多个机器）提取消息。

信息出租重访

当我们考虑租借信息时，我们更多的是根据从图书馆借书的角度而不是租赁房子或车子之类财产的角度来考虑这个问题的。隐式的，无论请求程序何时发出服务请求，它都是请求某些信息的副本；提供它时总是只带有给定时间的状态快照。现在这可能成了一个问题，除非它得到明确的了解和说明。一种策略是让提供程序给出*到期*时间以及信息。或者，请求程序可能会随着租借物（如图书馆的书）获得一张“票”，这将允许它有可能通过询问信息是否仍然有效来扩展租借对象，然后让服务器重新设置租借信息，而不必再次检索数据。

现在，因为我们可能期望 HTTP、SOAP 或某一传输协议会为我们处理这个问题，所以这似乎已经是个基本问题了。我们可以复用 HTTP 高速缓存语义（使浏览器和防火墙能将页面高速缓存），但它并非真正受提供程序的控制，而且请求程序可能也没有将 HTTP 用于传输。一个选择是将合用的支持构建到您的文档交换中，这样请求程序和提供程序之间的消息就对客户端的租借信息编码，如图 13 所示。

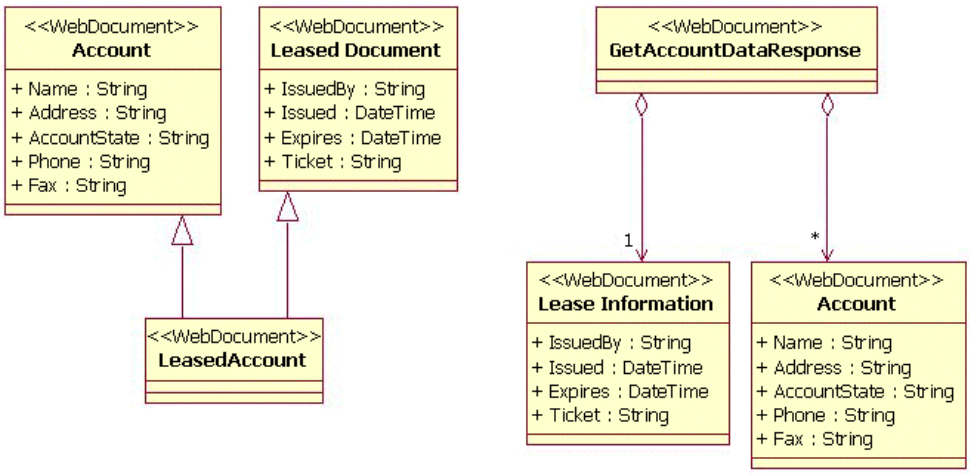

图 13 – 信息租借的两种实施

图 13 说明了信息租借模式的两种备用实施。第一种实施说明通过继承来将“帐户 XML”文档专门化为特殊的格式，这样，文档不仅仅是“文档”，而且是“租借的文档”，因此可包含其它信息。第二种实施返回了租借信息，以及作为响应消息的独立部分的帐户。这两种方法都同样有效，但它们生成结构不同的数据，区别在于风格的选择，即继承还是聚合。

得到的 Web 服务设计模型

图 14 说明图 7 中的普通服务设计可如何用特定于 Web 服务设计和开发的 UML 概要文件进行建模。概要文件非常简单，仅提出两个新的*构造型*（现有 UML 语言的扩展），用于《WebService》和《WebDocument》。通过复用 UML 中已存在的接口语义，我们可以简单地将服务的各发布方面可视化，一如 WSDL 中的定义。

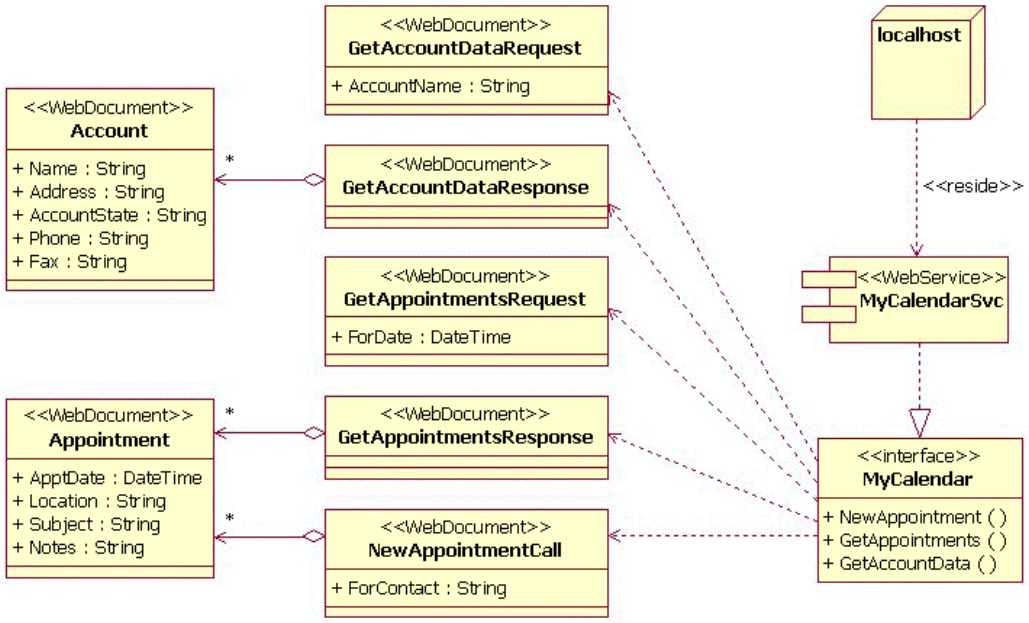

图 14 — XML Web 服务设计

下表说明图 14 中的概要表元素如何与 WSDL 中用于 MyCalendar 服务的工件相关。

WSDL 工件	UML 元素	注释
服务	《WebService》	服务表示为一个 UML 组件，该组件实现一个或多个接口并处在特定位置。《reside》 关系将记录实际的 URL 位置信息。
portType	接口	每个 portType 都表示为由一个或多个服务实现的 UML 接口。实现关系将记录绑定信息。
消息	《WebDocument》	每条消息都表示为一个 UML 类。XML 模式与 UML 的相互映射是消息和部分结构建模所必需的。
部分	属性或关联关系端	消息的每一部分都可以表示为《WebDocument》上的 UML 属性或者表示为与另一个《WebDocument》的关联。
地址位置	节点	节点表示服务所在的服务器。节点可以确定一组驻留服务，而服务可以驻留在多个节点上。

要强调的是，这种设计 Web 服务的方法提倡复用定义数据交换的文档以及受众多服务支持的接口。这是设计企业级解决方案时的关键功能。最好的情况是，与“帐户”文档交互的所有服务都基于同一个文档定义进行交互。我们还发现，操作性的、未发布的接口（如图 2 中的 SystemsManagement）可以由本领域的专家进行定义，然后启用，这样它们就可以在整个解决方案范围内共同实施。

结论

有些人期望 Web 服务能成为软件行业的一场革命，但也有人会告诉您，Web 服务只是个骗局。与所有新技术一样，真相介于两者之间。正如我们已经看到的，面向服务应用程序的构架中有一些基本区别；但是基于组件的现有开发技巧在实施这类服务时仍然有效。

有可能从单个领域模型同时得到组件和服务实施，如图 6 中的“帐户、联系、约定”模型所示。其它主要结论有：

- 可以从*现有的*基于组件的模型中得出服务模型。
- 可以应用通用的模式和设计，以协助转向面向服务模型。

这些通用模式和设计可以泛化、自动化，然后用模型到模型和模型到代码转换工具（如 Rational XDE）进行实例化（并在下图中说明）。

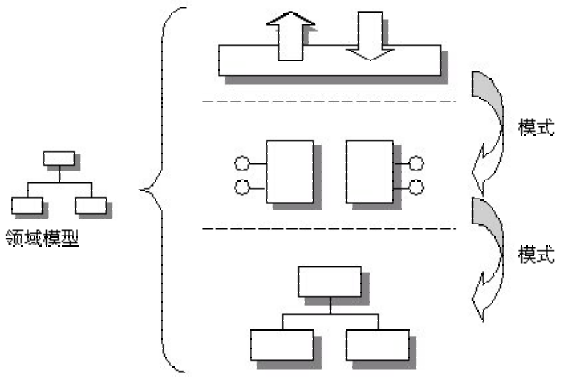

图 15 — 服务实施方法

采用这种方式有很多好处：它降低开发人员开销，增加实施的可预测性，并通过复用经过验证的模式来提高质量。将最佳做法（如“使用面向服务体系结构”）添加到您的专长并将部分最佳做法通过自动化工具（如 Rational XDE）进行编码，您将能够为更快速、更可靠地实施作出好的选择和折衷决策。

参考资料

[1] 有关更多信息，请访问 http://www.rational.com/uml/。
[2] Large-Scale, Component-Based Development，Alan W. Brown 著，Prentice Hall, 2000
[3] Public versus Published Interfaces，Martin Fowler 著，IEEE Software, March/April 2002 (Vol. 19, No. 2)
[4] W3C Web Services Architecture Requirements; http://www.w3.org/TR/2002/WD-wsa-reqs-20020429
[5] Web Services Security, Version 1,0; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp
[6] Web Services Referral Protocol, Draft; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[7] XML Web Services Basics; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[8] Rational Developer Network http://www.rational.net/