面向服务体系结构及其系统构建研究*

叶 钰, 应 时, 李伟斋, 张 韬
(武汉大学 软件工程国家重点实验室, 湖北 武汉 430072)

摘 要: 面向服务的体系结构(SOA)是一种新兴的软件体系结构,详细分析了 SOA 的基本结构和特点,比较了 SOA 同面向对象体系结构的不同之处,并结合一个实例给出了架构 SOA 系统的方法,描述了从基于组件设计方法过渡到面向服务设计方法的过程。

关键词: 面向服务的体系结构; 服务; 面向对象体系结构
中图法分类号: TP311.5 文献标识码: A 文章编号: 1001-3695(2005)02-0032-03

Research of SOA and Its System Building

YE Yu, YING Shi, LI Wei-zhai, ZHANG Tao
(State Key Laboratory of Software Engineering, Wuhan University, Wuhan Hubei 430072, China)

Abstract: Service-oriented architecture is a new architecture. Analyze the feature and basic structure of SOA detailedly and compare the differences between SOA and object-oriented architecture. With an example, this paper gives the approach architecting a SOA System and describe the process from the component-based design method to the service-oriented design method.
Key words: Service-Oriented Architecture (SOA); Service; Object-Oriented Architecture

1 引言

现代企业 IT 系统的开发与应用存在着许多困惑和难题: 如何聚集开发的各个模块? 如何提高代码模块的灵活性? 如何让开发出的系统更易维护? 如何提高系统模块的可重用性? 诸如此类企业在应用中遇到的难题,关键在于能否设计出一种先进的体系结构。这种体系结构就是面向服务的体系结构(SOA)。SOA 来源于早期的基于组件的分布式计算方式,在 OMG 和 IONA 的推动下,成为一个大家所广泛认可的规范。20 世纪 90 年代,CORBA 和微软的 COM 编程模式,促进了 SOA 的发展。随着 Java 编程语言、EJB 构件模式的发布以及 J2EE 应用服务市场的成熟,SOA 得到了进一步发展。现在,随着 Web Service 技术的兴起与逐渐成熟,为 SOA 的研究与发展提供了新的契机。SOA 由一系列相互交互的服务组成,每种服务都能提供对该定义功能集的访问。SOA 整体上设计和实现为一系列相互交互的服务,这种将业务功能实现为服务的方法可以增强系统的灵活性,系统通过增加新的服务来实现演化。SOA 定义了系统由哪些服务组成,描述了服务之间的交互,并将服务映射到一个或多个具体技术的实现[1]。总体来讲,SOA 的优势在于它的高可复用性、灵活性,以及更好的扩展性和可用性。

服务封装了业务功能,为了使服务之间能交互和通信,需要借助一些通信媒介,因为服务可能在单机上实现,也可能在局域网内实现,当采用因特网作为通信机制的时候,就是流行的 Web Service 技术了。Web Service 代表了 SOA 的一种实现,并且 Web Service 是 SOA 中最流行的一种实现。它是一种新型的 Web 应用程序,它具有自包含(Self-contained)、自描述(Self-describing)以及模块化的特点,可以通过 Web 发布、查找和调用。Web Service 实现的功能可以是响应客户一个简单的请求,也可以是完成一个复杂的业务流程。一旦一个 Web Service 配置好后,其他应用程序和其他 Web Service 就可以直接发现和调用该服务。

2 SOA 模型

简单来讲,SOA 是设计和构建松散耦合软件的解决方案,能够以程序化的,可访问软件服务的形式公开业务功能,以使其他应用程序可以通过已发布和可发现的接口来使用这些服务。通过应用 SOA,一个企业可以使用一组分布式服务来构成并组织应用程序。这样,企业就能通过重用他们自己的资源和他们的伙伴的业务功能来构造新的应用程序。

2.1 服务

服务是一个粗粒度的、可发现的软件实体。它以一个单独的实例存在,并通过一组松散耦合和基于消息的模型与其他的应用或服务交互。通常,服务具有以下几个特征:

(1) 粗粒度。相对于组件而言,服务上的操作被实现为封装了更多的功能,并且依赖于更大的数据集。

(2) 基于接口的设计。服务实现为单独定义的接口。这样带来的好处是单个服务可以实现多个接口,而多个服务也可以实现统一的接口。

(3) 可发现。不论是在设计时还是在运行时,服务都应该能被发现。这种发现可能是通过接口标志,也可以是通过服务标志。

收稿日期: 2004-01-18; 修返日期: 2004-03-21
基金项目: 湖北省青年杰出人才基金资助项目(2003ABB003); 武汉市科技攻关计划项目;武汉大学科技创新基金资助项目

(4) 单实例。这一点与基于组件的开发不同,组件的实例可以是随意的,但每个服务只有一个唯一的,一直运行的实例,这个唯一的实例与多个客户端进行通信。

(5) 松散耦合。服务通过松耦合的,基于消息的方法(如 XML 文档交换)连接到其他的服务或客户端。

2.2　SOA 的基本结构[2,3]

SOA 结构(图 1)中共有三种角色:

(1) 服务提供者(Service Provider)。发布自己的服务,并且对使用自身服务的请求进行响应。

(2) 服务代理(Service Broker)。注册已经发布的服务提供者,对其进行分类并提供搜索服务。

(3) 服务请求者(Service Requester)。利用服务代理查找所需的服务,然后使用该服务。

SOA 体系结构中的组件必须具有上述一种或多种角色。在这些角色之间使用了三种操作:

(1) 发布(Publish)。使服务提供者可以向服务代理注册自己的功能及访问接口

(2) 查找(Find)。使服务请求者可以通过服务代理查找特定种类的服务。

(3) 绑定(Bind)。使服务请求者能够真正使用服务提供者。

为支持结构中的三种操作,SOA 需要对服务进行一定的描述,这种服务描述(Service Description)具有以下两个特点:

①它要声明服务提供者的语义特征。服务代理使用语义特征将服务提供者进行分类,以帮助具体服务的查找。服务请求者根据语义特征来匹配那些满足要求的服务提供者。服务描述应该声明接口特征,以访问特定的服务。

②服务描述还应声明各种非功能特征,如安全要求、事务要求、使用服务提供者的费用等。接口特征和非功能特征也可以用来帮助服务请求者对服务提供者的查找。

理论上,面向服务的体系结构这种思想,在其简易性上,十分吸引人。如果能够用定义很好的机制封装应用,就有可能将一个单一的应用加入到一个服务的集合中。封装的过程创建了一个抽象层,屏蔽了应用中复杂的细节(你将不必关心用的是哪一种编程语言,什么操作系统,应用程序用的是什么数据库产品)。唯一相关的就是服务所描述的接口。

2.3　与面向对象体系结构的比较

面向对象的体系结构就是应用面向对象的方法建立系统的体系结构。其主要思想是:对问题域中客观存在的各项事物建立相应的对象,对象的属性和方法分别描述事物的静态特征与动态行为,对象间的交互通过对其方法的调用进行。面向对象的特点是它封装了对象的属性和行为,实现了信息隐蔽。同时对象内部行为的修改不影响外部对它的调用。但面向对象体系结构的一个明显的缺点是:当一个对象通过过程调用与其他对象交互时,它必须知道其他对象的标志,而当一个对象的标志改变时,需要对所有调用这一方法的对象进行修改[4]。

面向对象的体系结构应用在分布式系统中,所有的状态信息都放在服务器端,对对象的访问包含网络调用和往返通信。因为每个对象都是与事先已创建好了的对象进行通信,所以对象间的通信可以理解为有状态的。同时,服务器也可以采用一些智能管理机制(如 EJB 钝化策略)来更好地管理在服务器上的加载。并且面向对象的体系结构通常采用特定的非因特网友好的协议,如 IIOP DCOM, RMI 等[5]。

在面向服务的体系结构中,应用端通过发送请求信息来与一个特定的应用程序进行通信,因为所有的请求都被发送到同一个服务终端,所以这种通信是无状态的。只要请求能被分发到不同的服务操作来处理,这种已描述的服务终端可以独立于一个服务,也可以被多个服务所共享。

3　构建 SOA 系统

3.1　应用层次设计

面向对象技术和语言是实现组件的伟大杠杆,在架构 SOA 系统的时候,虽然一个面向组件的设计并不能完全演化为一个合乎要求的面向服务的设计,但组件是实现服务的有效途径。对于一个 SOA 系统,可将其应用实现层划分为三层:①服务在最上层,是最粗粒度的实现,它作为应用系统的外部视图;而内部的重用和组合仍然使用传统的组件设计。②组件层。③类/对象层。其结构图如图 2 所示。

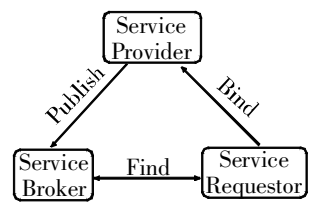

图 1　SOA 的基本结构

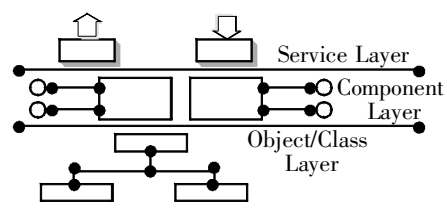

图 2　SOA 的应用实现层结构

在构建一个具体的 SOA 系统的时候,首先给出系统的逻辑模型,这个逻辑模型用类/对象图来进行建模;然后依据具体的需求给出相应的组件模型,组件的设计主要在于设计其功能接口;最后就是服务模型,为了使设计更加合理,要使用到具体的组件平台或相关的设计模式。

3.2　一个实例

为了说明组件和服务在实现一个应用时是如何交互的,以及如何从面向对象的设计方法过渡到面向服务的设计方法,下面结合一个订单管理关系来描述这个过程。图 3 是用 UML 给出的该系统的逻辑模型。该类图由三个类组成,分别表示客户的客户(Customer)、订单(Order)、产品(Item)。

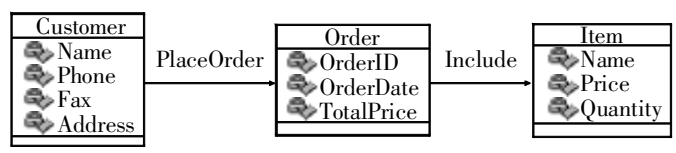

图 3　逻辑模型

3.2.1　基于组件的设计

基于组件设计的关键在于接口,在现行的组件平台中,接口的设计有其固定的模式,就是对每一个属性都提供设置值和返回相关值的操作。如果用面向组件的设计方法(COM 或 J2EE),其类图可以描述为如下形式,图 4 将逻辑模型转换成面向组件的实现模型。ICustomer 接口对所有的属性提供了设置和返回相关值的操作,类 Customer 实现了 ICustomer 接口,而组件 CCustomer 则依赖于 Customer 类。

3.2.2　基于服务的设计

对组件而言,每一个组件实例都表示一个独有的对象;但对于服务而言,它必须为所有的请求者提供服务,也就是说,它

必须管理所有的资源,这要求服务的构建者将服务作为一个管理器对象。该对象负责创建和管理所有服务类型的实例。

此时可以应用 J2EE 的值对象模式来表示实例状态和对象, Value Object(值对象)模式支持分布式组件高效信息交换。设想一个具有多个属性(包括客户 ID、姓名、地址、社保号码等)的分布式组件,因为通过网络执行远程调用开销巨大,所以简单地为每个属性提供远程 set 和 get 操作是低效的。而 Value Object 模式则提供了返回 Value Object 的远程操作,这使我们可以通过一次远程调用取得所有属性值[6,7]。

采用面向服务的设计方法,系统的类图可以设计为图 5 所示,将服务以组件的形式封装。该组件实现了三个接口 ICustomer, IOrder 和 IItem, 而这三个接口又依赖于相应的值对象类。在这三个接口中提供的方法不再是简单的设置和返回相关属性的操作,而是在值对象间传递了更多的信息。

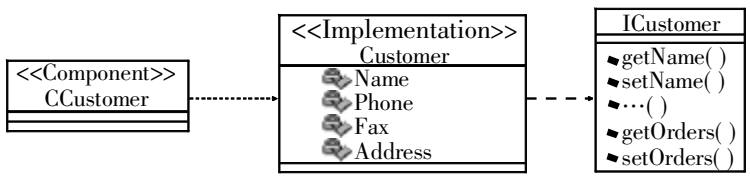

图 4 组件模型

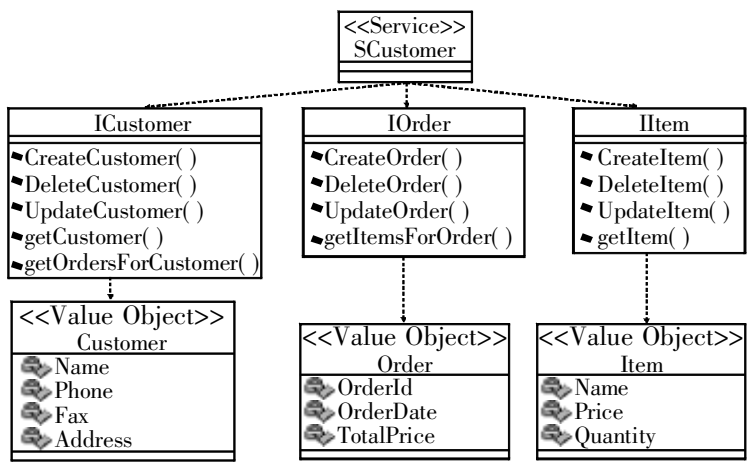

图 5 服务模型

4 结束语

面向服务体系结构是新型的软件体系结构,它实现了在 Web 上分发并集成应用程序逻辑,解决了 Web 数据集中、冗余、无法共享的缺陷,屏蔽了后台技术的复杂性。该体系结构提出了服务提供方、服务代理方、服务请求方三个角色,通过发布、搜索、绑定三种基本操作完成了 Web 应用程序的实时整合。该体系结构实现了 Web 集中计算模式向分布式计算模式的转变,并克服了原有分布式对象模型的紧密耦合的缺点。

SOA 的强大和灵活性将给企业应用带来巨大的好处。如果某组织将其 IT 体系结构抽象出来,并将其功能以粗粒度的服务形式表示出来,每种服务都清晰地表示其业务价值,那么,这些服务的顾客(可能在公司内部,也可能是公司的某个业务伙伴)就可以得到这些服务,而不必考虑其后台实现的具体技术。更进一步,如果顾客能够发现并绑定可用的服务,那么在这些服务背后的 IT 系统能够提供更大的灵活性[8]。

本文详细介绍了面向服务体系结构的特点及其基本结构,比较了它与面向对象体系结构的区别,并结合一个简单的订单管理关系来说明了面向服务体系结构系统的构建过程。在构建的过程中,将应用实现分成三层,最底层是对象/类层;其次是组件层,组件是实现服务的最有力的工具;最上层是服务层,是应用最粗粒度的实现。

面向服务体系结构的构建要依托于具体的组件平台及其相关技术,本文应用了 J2EE 的值对象模式,该模式在本文的实例应用中可以节省远程调用的开销。

参考文献:

[1] Ian Brown, *et al*. Using Service-Oriented Architecture and Component-based Development to Build Web Service Applications[R]. A Rational Software WhitePaper from IBM 04/03, 2002. 11-15.

[2] Mikep Papazoglou. Tilburg University INFOLAB, Dept. of Information Systems and Management, Service-Oriented Computating: Concepts, Characteristics and Directions[C]. Roma, Italy: 4th International Conference on Web Information Systems Engineering, 2003.

[3] Luciano Baresi, *et al*. Modeling and Analysis of Architectural Styles Based on Graph Transformation[C]. ESEC/FSE, Helsinki, Finland, 2003 ACM 1-58113-743-5/03/0009, 2003. 72-74.

[4] Li Xiaolong, *et al*. Pipe-Filter Software Architecture and Its Design [J]. Computer Engineer and Application, 2003, (1): 114-116.

[5] Jorgen Thelin. Chief Scentist Cape Clear Software Inc. A Comparision of Service-Oriented, Resourse-Oriented, and Object-Oriented Architecture Styles[R]. A WhitePaper of Jorgen Thelin / Cape Clear Software Inc., 2003.

[6] Paul J Perrone, *et al*. J2EE 构建企业系统(专家级解决方案)[M]. 张志伟,谭郁松,张明杰. 北京:清华大学出版社, 2001.

[7] Vlada Matena, Beth Stearns. J2EE 平台上的 EJB 组件开发[M]. 瞿裕忠,陆海涛,等. 北京:机械工业出版社, 2001.

[8] Jason Bloomberg. Senior Analyst LLC, The Role of the Service-Oriented Architecture[R]. A Rational Software WhitePaper, 2003.

作者简介:

叶钰,男,硕士研究生,主要研究方向为软件体系结构、Web Service、工作流等;应时,教授,博士生导师,主要研究方向为基于组件的软件工程方法、软件体系结构、软件模式等;李伟斋,硕士研究生,主要研究方向为软件体系结构、软件模式等。

(上接第 31 页)

[9] Yau S S, Pao-Sheng Chang. A Metric of Modifiability for Software Maintenance[C]. Software Maintenance, Proceedings of the Conference on, 1988. 374-381.

[10] Allen E B, Khoshgoftaar T M. Measuring Coupling and Cohesion: An Information-theory Approach [C]. Software Metrics Symposium, 1999. Proceedings. Sixth International, 1999. 119-127.

[11] S S Yau, S Liu. Some Approaches to Logical Ripple Effect Analysiy [R]. SERC-TR-24-F, Software Engineering Research Center, University of Florida, Gainesville. FL, 1988.

[12] Rome Air Development Center, Automated Life Cycle Impact Analysis System [R]. Rome Air Development Center, Air Force Systems Command, Griffiss Air force Base, RADC-TR-86-197, Rome, NY, 1986.

[13] Pfleeger S L, *et al*. A Framework for Software Maintenance Metrics [C]. Software Maintenance, Proc. Conference, 1990. 320-327.

[14] S S Yau, *et al*. Knowledge Representation of Software Component[J]. Interconnection Information for Large-Scale Software Modifications, IEEE Transactions of Software Engineering, 1987, 13(3): 545-552.

[15] L Cleveland. An Environment for Understanding Programs [C]. Proceedings of Conference on Software Maintenance, 1988. 500-509.

作者简介:

何进(1978-),男,硕士研究生,主要研究方向为软件工程、质量管理和控制等;苏秦(1963-),女,教授,博士生导师,工学博士,主要研究方向为质量管理与质量认证、生产与运作管理等。