文章编号:1001 - 9081(2006)03 - 0699 - 04

SOA 体系下细粒度组件服务整合的探讨

杜 攀,徐 进
(北京理工大学 计算机科学技术学院,北京 100083)
(dupan@bit.edu.cn)

摘 要:研究了服务粒度对 SOA 架构下系统效率的影响,提出了使服务粒度能够灵活变化的方法。在基于 Java 平台的应用中,通过对本地细粒度组件服务的编排性描述,提供执行该描述的机构,使它们呈现为粗粒度的服务组件,从而加强系统的灵活性,提高系统的运行效率。最后结合具体应用给出了该方法的设计和部分实现。

关键词:SOA; 组件服务; 服务粒度; 服务整合
中图分类号: TP311.5 **文献标识码**:A

Discussion about the integration of component services with finer granularity in SOA applications

DU Pan, XU Jin
(*School of Computer Science and Technology, Beijing Institute of Technology, Beijing* 100083, *China*)

Abstract: To make the granularity of component services more flexible, an approach was presented after an investigation about the influence of service granularity on the efficiency of SOA application. In the applications based on Java platform, this approach changed the finer grained services into coarsely grained to improve its performance with describing those finer grained services on schedule and providing an engine to execute this description. The design and part implementation of this approach were given.

Key words: SOA; component service; service granularity; service integration

0 引言

在面向服务的体系结构(Service Oriented Architecture, SOA)的实施过程中,对于系统设计者来讲,设计服务组件时如何确定服务的粒度是一个非常重要的问题。

从组件复用的角度考虑,组件的粒度越细,未来该组件被直接复用的可能性就越大。在系统规模一定的情况下,组件粒度越细,意味着独立组件的个数就越多;从组件间通信量的角度考虑,通信量将随着独立组件个数的增长而产生组合爆炸。同时,现阶段面向服务的体系结构主要依靠 Web Service 技术作支撑,而 Web Service 技术主要通过 SOAP 协议进行网络通信,SOAP 协议中的消息格式又是基于 XML 语言的,这种消息格式的好处是消息能够自描述,而自描述的代价是添加大量的标记数据,从而导致网络通信量的大幅增加。也就是说,系统内部的网络通信方式、复杂的消息格式以及大量的独立服务组件,都将成为降低系统的运行速度的重要因素;灵活性是当今商业活动的一个非常显著的特点,而这种业务上的灵活性,恰恰是传统软件系统的软肋。当企业的业务发生变化时,应用系统必须进行相应的改变以适应这种变化。这时,很难保证系统的变化不会波及到服务组件内部,从而使得服务组件的优势消失殆尽,系统维护的复杂度也随之骤升。

综上所述,服务组件粒度的粗细,直接影响服务组件的复用性和系统的整体运行效率。灵活多变的业务需求对服务粒度又提出了更为苛刻的要求。

本文通过一种可配置的细粒度组件服务的整合方法,寻找上述问题的解决方案。目标是在 SOA 的设计和实现过程中,使用细粒度的组件服务提供基本的功能单元,通过某种可配置的方法,动态组装这些细粒度组件为粗粒度组件服务;在业务变化涉及到组件服务的内部服务时,又能够通过修改配置,重新组装细粒度服务组件,来重构粗粒度服务。

1 SOA 及服务组件概述

随着企业业务的不断扩展,在企业的信息基础设施建设和业务发展之间出现了不同步的现象。SOA 通过服务的概念,使得企业的应用系统同其合作伙伴的应用系统之间建立联系,从而实现业务级别的复用。企业的应用系统可以根据业务发展的需要实现随需整合,从而为两者之间的同步提供保障。

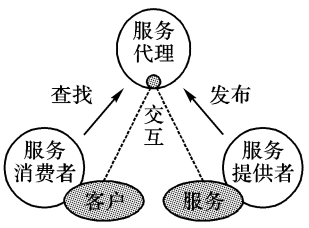

图 1 面向服务的体系结构

关于面向服务的体系结构的研究,正在逐步走向完善。

收稿日期:2005 - 10 - 13;**修订日期**:2005 - 12 - 19

作者简介:杜攀(1981 -),男,河南南阳人,硕士研究生,主要研究方向:软件体系结构、分布计算技术、人工智能软件; 徐进(1954 -),女,北京人,高级工程师,主要研究方向:软件体系结构、分布计算、人工智能.

同时,由于 Web Service 技术相关的标准的确立和 Web Service 的应用普及,基于 Web Service 技术的 SOA 的构建,也渐渐进入实际应用阶段。

SOA 的具体架构如图 1 所示。图 1 中显示的是 SOA 的三个基本角色,服务提供者、消费者和代理以及它们的一些工件和操作。

服务是网络上可用的软件资源。服务提供者通过标准机制来提供服务,同时服务消费者通过网络有计划地消费服务。服务代理发布服务的所处位置,并且在消费者发出服务请求时,对这些服务进行定位。

一个服务由两部分组成:接口和实现。接口提供了消费者和提供者之间程序性的访问约定。一个服务接口必须含有该服务的身份,该服务的输入和输出数据的细节,以及和该服务的功能和目标有关的元数据。服务的实现包含该服务的功能性或业务性的逻辑。

服务包括以下五个类型:

数据服务 允许用户访问、集成、翻译和转换整个企业的各种关系型和非关系型数据源。

组件服务 提供对打包应用程序的访问。

业务服务 处理具体应用业务的复杂的服务,这些服务使用多个打包或应用程序的功能。

合成服务 使用上述三种类型合成新的服务。

共享的或基础架构服务 低级服务,如日志记录等。

本文的关注点为组件服务及组件服务的粒度。

组件服务 是粗粒度服务,无论组件服务是否是打包应用程序,他们都可以从一个单一的企业资源来发布。组件服务实现的方法是通过各个应用程序 API 来提供可复用的功能,可使用分布式计算技术来实现。

服务粒度 可以按照基于服务的功能和服务发送、接收的数据数量来定义服务,如细粒度服务,粗粒度服务和组合服务。细粒度服务执行了最小的功能,发送和接收少量的数据;粗粒度服务执行了较大的业务功能,并交换了更多的数据。

2 设计与实现

借鉴基于 J2EE 的 Web Service 技术以及基于 Web Service 的工作流技术的设计思想,寻求一种细粒度组件服务的整合方法,以便将细粒度组件服务整合成为粗粒度的服务,并发布为 Web 服务,从而减少系统的数据通信量。在必要的时候,又能够通过更改配制文件的方式,对原有的粗粒度服务进行重组,以满足业务多变的需求。

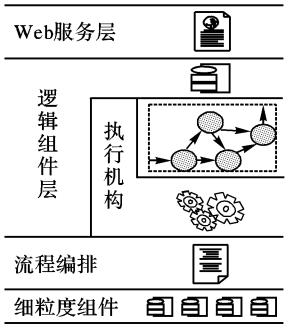

图 2 基本架构

1)架构的最底层为细粒度组件,即细粒度组件层,它们工作于传统的组件容器之内。

2)次第层为流程编排层,通过对细粒度组件进行一个流程的编排,从而使它们成为一个整体。这个流程编排文件(流程描述文件)实际上就是该架构设计目标,也就是前面论述中提到的可配置方法中的配置文件。

3)逻辑组件层工作在流程编排层之上,它负责使编排好的细粒度组件服务协同工作,以粗粒度组件服务的形式对外展现。

4)Web 服务层负责将逻辑组件层生成的粗粒度组件服务发布出去。

该架构设计的核心是逻辑组件层,它承担的任务为:以细粒度组件和工作流程描述文件为原料,向 Web 服务层提供服务,从 Web 服务层的角度看,逻辑组件层提供的是粗粒度的服务组件。

业务变化可能导致 Web 服务的功能发生变化。由于粗粒度服务组件只是基于细粒度服务组件和流程编排的逻辑服务组件,因此,就有可能只需要修改流程配制文件,而无需深入到源代码级的功能修改,来满足功能变更的需求。

从图 2 也可以看出,要达到上述目标,需要解决的关键问题集中分布在流程编排层和逻辑组件层。

1)流程编排层需要解决的问题有:细粒度组件服务的接口描述语言和接口描述;基于接口描述的流程描述语言问题和流程编排。

2)逻辑组件层需要解决的问题有:执行机构的工作方式、流程实例化和逻辑粗粒度组件的接口呈现。

3)Web 服务层主要问题在于如何将一个服务组件发布为 Web 服务;细粒度组件层的主要问题在于组件的实现,这两个问题都属于常规问题,本文将不予讨论。

2.1 接口描述相关问题

可以利用 Java 编译后的 class 文件来实现对接口的动态描述。Java 反射 API 显示二进制类格式中包含的元数据的 Java 接口。使用发射可以动态显示 Java 类的特性:基类、超接口、方法签名和字段类型。还可用反射动态实例化对象,调用方法以及转化字段。Web Service 使用 Web 服务描述语言 WSDL(Web Service Description Language)描述 Web 服务的接口。WSDL 基于 XML 语言,具有自描述的特性,因此适合于分布式应用。但本文讨论的问题是局限于本地细粒度组件服务的整合,因此,可以使用接口本身的 class 文件作为接口自己的精确描述。这对于组件开发人员来说,减少了接口描述语言相关的工作量,从而提高工作效率。

2.2 配置文件相关问题

配置文件实际上就是一个流程描述文件,它负责完成基于细粒度组件服务的工作流程编排。当组件功能变化时,为了减少功能改变引起的工作量,引入了流程的概念。通过对细粒度组件进行流程的编排,使其不但能够协作产生粗粒度组件的功能,还能够根据需要,通过改变流程的编排,适应业务变化引起的组件功能的改变。

为了便于流程的编排,选择 XML 语言来描述流程。在基于 Web Service 的流程编排实现中,BPEL4WS 是一个标准的流程执行语言,它基于 WSDL 的服务接口描述来定义相应的流程执行过程。基于组件接口文件本身对接口的描述,制定一个用于定义流程的流程执行语言。同 BPEL 一样,使用 XML 语言的优势,来实现流程的编排。

一个流程所包含的基本实体有:活动、事件、联系和消息。其中活动是指流程执行过程中的一个环节,它完成一个具体的功能;事件标识流程执行过程中的状态改变;联系标识活动之间的同步关系;消息表示活动之间的通信实体。

安全身份管理系统中的 key 解锁功能的实现流程如图 3 所示。在解锁流程中,共有四个活动,分别是申请、审批、授权和回复。活动之间的联系指明了流程执行过程中各个活动之间的同步关系。比如申请完成后,可能根据用户的种类,选择让流程进入审批活动或者跳过审批活动,直接进入授权活动。

数据流以活动之间传送的消息为载体进行流通。比如要执行解锁流程,首先必须向申请活动传入需要解锁的用户信息和钥匙信息,这两个信息作为一个消息传入申请解锁活动,然后申请解锁活动对传人的信息进行处理(比如校验合法性)后,决定引发哪个事件。

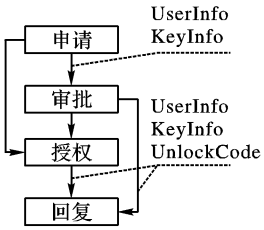

图 3　解锁流程

每一个活动的开始、结束、挂起、就绪、执行、异常等都对应着该活动状态的改变,因此都有一个事件来表明这种改变。

为了定义(或者说描述、配置)上述流程,使以后的执行机构能够按照流程描述,生成相应的流程实例,需要把流程中所涉及到的上述实体,映射为描述文件中的 XML 元素。此外,由于流程是一个动态的概念,在流程执行过程中,还会产生一些中间数据以辅助流程运转,这些中间数据也必须是执行机构所能够理解的,所以,需要把这些中间数据也映射为描述文件中的 XML 元素。

具体地讲,这些实体应该包括流程 < processes > ,临时变量 < variables > ,异常处理器 < faultHandlers > ,事件处理器 < eventHandlers > 和流程中的各种活动 < activity > 以及活动中的流程控制事件,比如 < receive > , < reply > , < sequence > , < flow > , < invoke > , < assign > , < while > , < switch > , < throw > , < catch > 等。

在上述拟使用的标签范围内,实现解锁流程的流程描述片断如下:

```
< process name = 'unlock' >
  …
  < activity name = 'apply' >
    < operation > apply ( UserInfo, KeyInfo) </operation > </
  activity >
  …
  < activity name = 'response' >
    < operation > response( ) </operation >
  </activity >
  < sequence >
    < receive >
      < variable > UserInfo </variable >
      < variable > KeyInfo </variable >
      < activity > apply </activity >
    </receive >
    …
    < invoke > apply </invoke >
    …
    < reply >
      < variable > userInfo </variable >
      < variable > keyInfo </variable >
      < variable > unlockCode </variable >
      < activity > response </activity >
    </reply >
  </sequence >
</process >
```

2.3　执行机构的设计和部分实现

确定了流程的描述方法后,下面讨论如何将上述精确描述的流程实例化,并寻找一种合适的执行方式。

执行机构需要实现的基本功能可概括为三个层次的管理功能:流程管理 PM、活动管理 AM 以及向它们提供支持的队列 QM 和事件管理 ALM。执行机构的体系结构如图 4 所示。

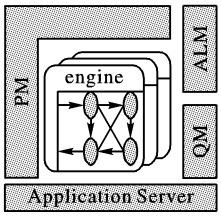

图 4　流程执行机构

整个执行机构的实现建立在应用服务器之上,组件服务的实际生命周期的管理跟执行机构无关,执行机构依靠应用服务器来管理实际的组件服务的行为。执行机构建立逻辑上的流程实例,并管理流程和流程中包含的各种逻辑实体。

流程管理实现了基本的流程生命周期的管理,包括流程实例的创建、销毁和流程状态的转换。它实际上就是引擎工厂(engine factory),由它产生的每一个引擎对应一个流程实例,管理一个流程实例中每一个活动的状态转换,执行活动管理的功能。

活动管理实现了基本的活动生命周期的管理,包括活动实例的创建、销毁和活动的状态转换。

队列管理为流程和活动的状态转转提供支持,为处于不同状态的流程和实例分别建立不同的队列。

事件管理也是为流程和活动的状态转换提供支持,整个执行机构依靠各种实现定义好的事件作为触发器,改变流程或者实例所处的状态,驱动流程的流转。

下面的伪码描述了流程管理器如何生成流程的执行引擎:

```
engine = new Engine ( getEngineConfiguration( ),
  createQueueManager( ), createAlarmManager( ),
  createActivityManager( ));
```

其中,getEngineConfiguration( )用于获得执行引擎的初始化参数信息。当 engine 到达一个消息或者是发生了某个特定的事件时,引擎就会读取指定流程的定义(配置)文件,生成流程中的活动定义实体来建模流程。

活动定义实体中包含了实例化一个活动实体所需要的所有信息,从这个意义上说,活动定义实体和活动的关系就好比类和对象之间的关系一样。引擎根据特定的访问模式访问建模后的流程,再根据流程模型,逐个实例化流程中的每一个活动。

流程的执行过程,主要完成的活动实例的生命周期相关的管理。流程的状态包含以下几种:

不活动:当流程建立时,流程的所有活动都是不活动的。

就绪态:活动被放进了队列,而且活动等待的条件已经成熟。

执行态:活动正在执行。

完成态:活动已完成。

错误态:活动执行过程因发生错误而被结束。

等待:活动被放进了等待队列。

未知:活动状态为空。

根据以上活动相关的状态,可以对活动所必需包含的通用操作进行定义,如下:

Execute():活动的功能性操作。比如,需要在这里调用组件服务的实际操作,或者服务组件之间传递消息等。

isReady():判读活动当前是否可以进入执行态。

finish():当活动执行完成时,执行一些必要的扫尾工作。

2.4 粗粒度服务接口呈现

细粒度的组件服务在经过流程编排以后,协作产生更强大的功能,处理更多的数据,从而表现为粗粒度的组件服务。服务如果不发布出去,或者不能够遵从 WSDL 国际标准,从而为消费者所使用,就不能称之为真正的服务。

如果为流程取一个服务名称,然后整理出服务所需要的所有输入信息和输出信息,就能够像一个普通的组件一样生成一个服务接口。或者以细粒度服务为单位组织每一个细粒度服务所需要的输入输出参数,从形式上构成粗粒度服务的内部操作,并依据这些信息构造粗粒度服务的服务接口。

基于上述服务接口,就可以模仿现有的 Web Service 技术中服务发布的方法,建立对应的 WSDL 文件对外发布。但是由于这里的服务接口只是逻辑概念上的接口,必须在调用之前将调用形式转化为逻辑概念上的粗粒度服务在执行过程中能够实际使用的信息形式,然后再转交给执行机构。

3 结语

目前网络传输同机内传输相比,效率不理想。在使用 Web Service 技术实现 SOA 时,必须考虑到低效率的网络传输造成的负面影响。通过提供一种对企业细粒度组件服务的整合方法,降低基于 SOA 架构的企业应用中的网络传输量,从而减轻设计复杂度,提高系统开发、运行效率。

面向服务的架构的最大优点在于为企业的应用整合提供了有力的支持,但由于面向服务的体系结构考虑问题的出发点是企业业务的整合,是从宏观的角度探讨系统的复用。因此,大都从 Web Service 的角度讨论服务的组合问题,这个探讨属于业务复用的范畴,它的着眼点是业务;而本文讨论的问题的着眼点是业务内部的服务构件,讨论服务构件的粒度的转化方法,使服务构件在不同的粒度层次都能够被复用。这两个问题在特定条件下能够相互转化。

参考文献:

[1] 王千祥. 应用服务器的原理和实现[M]. 北京:电子工业出版社,2003.

[2] SHALLOWAY A, TROTT JR. 设计模式精解[M]. 北京:清华大学出版社,2004.

[3] VENNERS B. 深入 Java 虚拟机[M]. 第 2 版. 北京:机械工业出版社, 2003.

[4] NAGAPPAN R, SKOCZYLAS R, SRIGANESH RP. Java Web 服务开发[M]. 北京:清华大学出版社,2004.

[5] RAMANATHAN R. Make Application Integration Easy[EB/OL]. http://www.fawcette.com/weblogicpro/2004_09/magazine/features/rramanathan/, 2004-08-20.

[6] HOLLINSWORTH D. The Workflow Reference[R/OL]. www.wfmc.org/standards/docs/tc003v11.pdf, 1995-01-09.

[7] Web Services Description Language (Version 2.0)[EB/OL]. http://www.w3.org/TR/2005/WD-wsdl20-20050803, 2005-08-03.

[8] ANDREWS T, CURBERA F, DHOLAKIA H, *et al*. Business Process Execution Language for Web Service(Version 1.1)[EB/OL]. http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp, 2003-05-05.

(上接第 698 页)

和 Matlab 的文件读写功能传递参数,在实时性要求很高或参数很多的情况下并不适合。

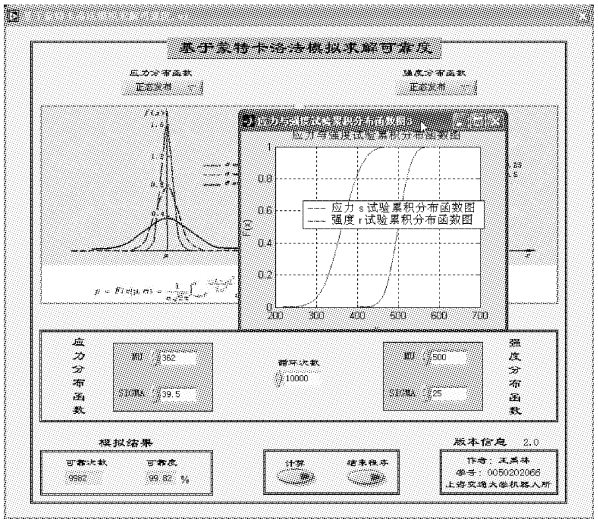

图 4 程序运行效果图

VC++ 参数类型转化法实现起来最为复杂,对开发者在设置和编程上有一定要求。但它可直接传递参数而不需要中间文件,可移植性很高。

另外 Matlab 的 Compiler 具有一定的局限性,在混编的过程中只能使用 Matlab 数学库中的函数和图形库中的部分函数,对于一些工具箱中的函数,如用到一些数学库之外的函数,就有可能出现错误。即使编译通过,在运行的时候也有可能出错。

利用上述的三种方法均实现了 LabVIEW 与 Matlab 的无缝集成。程序的运行效果图如图 4 所示。这三种方法各有优缺点,用户可根据实际情况选择合适自己的方法。

参考文献:

[1] 裴锋,汪翠英. 利用 COM 技术的 LabVIEW 与 Matlab 的无缝集成[J]. 仪器仪表用户,2005,12(2):97-98.

[2] 苏金明,黄国明,刘波. Matlab 与外部程序接口[M]. 北京:电子工业出版社,2004.

[3] National Instruments Corporation. LabVIEW Application Builder User Guide[Z], 2003.

[4] 雷振山. LabVIEW 7 Express 实用技术教程[M]. 北京:中国铁道出版社,2004.

[5] 刘建伟. VC++ 与 Matlab 混合编程的快速实现[EB/OL]. http://kenbeyond.blogchina.com/kenbeyond/925404.html, 2005-06-01.