基于 SCA 的面向服务的设计与实现

SOA Design and Implementation Based – on Service Component Architecture

尧飘海　张云华　刘　巅　鲍　陈　（浙江理工大学 信息与电子学院 浙江 杭州 310018）

摘　要：SOA 架构已成为现代应用开发领域最重要的课题，其服务的平台无关性、语言独立性和松散耦合性极大的方便了各系统之间的集成和扩展。基于 SCA 构件技术是实现 SOA 架构的最佳实践，构件具备的完备性、实用性、稳定性和独立性涵盖了整个软件生命周期，也将成为占有绝对优势的软件工程实践方法。

关键词：面向构件 面向服务架构 组件模型 Java

1　引言

无论是在过去的传统行业，还是在现代的信息产业中，其发展的过程都是把复杂的问题通过应用抽象、分解、迭代和细化后，再进行分类归并集成，逐渐的组成一个功能完善的系统或产品。其中的分解技术就是把每种功能分解成较小、独立和具有可插拔式功能的单元，例如家用电器或汽车的零部件等等，每个部件都有相对独立的功能。当然，在传统的电器或汽车行业中，各部件或许在整个系统中的作用相互依赖性强；而在软件工程中，如何把一个复杂的系统通过抽象，分解成一个个可单独处理的可管理单元，提高软件的复用技术和软件开发效率，并保持系统中各部件的松散耦合性，并具有可插拔性，也是日前软件开发中所面临的最主要的问题。

随着软件工程不断进步，人们通过借鉴传统行业中各种产品的生产和发展过程，提出并应用了面向组件 CBD（Component – Based Development，CBD）的软件开发模型和面向构件 SCA（Service Component Architecture）的开发模型，它们是指通过有计划、有目的集成现有的软件和服务或它们的各种部分独立功能来进行系统的集成开发，实现现有的各种软件或服务等资源无缝的结合和利用，提高软件开发效率，加速软件开发过程，提前实现软件或服务等产品的上线。

2　技术背景

软件行业的发展非常的迅速，经历了从以算法和数据结构为基础的面向过程阶段、以实体或抽象的对象为核心的面向对象技术的发展过程，以及早期的以 CORBA（Common Object Request Broker Architecture，公共对象请求代理体系结构）、DCOM（Distributed Component Object Mode，分布式组件对象模型）和 EJB（Enterprise JavaBeans）等为代表的构件技术发展过程。在以前的技术架构下，开发人员不但要考虑具体的业务需求和业务逻辑，同时还要把注意力放在技术细节上，如线程的同步、事务锁的控制与管理等等，致使应用软件的开发变得越来越复杂。而我们现在提出的面向构件的技术是在综合前三个阶段发展的基础上，但是它绝不是模块化编程方法中的子程序、面向对象方法中的对象或类、或系统模型中的包等。构件是粗粒度、松耦合、及更高层次上的抽象，具有独立发布的功能部分，可以通过其接口访问它的服务。所以面向构件技术并没有采用某个代表性语言，而是借助于系统的任何语言，比如用 XML 来描述和组装构件。因此，实现的各构件是一个可交付的独立软件单元，通过接口或服务提供粗粒度的功能，遵循软件技术的标准规范和准则，并且构件之间可以无缝的组装、协同的工作，实现组件分发和复用的功能，具有完备性、实用性、稳定性和独立性。面向构件的技术架构就像计算机主板一样对外提供的各种数据接口，如我们常用的 USB，PCI 等各种串行或并行接口。也可以说像 JAVA 采用 JVM 的技术架构，只要提供构件虚拟机或中间件的形式就可以很容易的实现系统的跨平台、数据库、网络和跨语言的特性。面向构件涵盖了从软件构思、个体构件开发、构件组装或联合，及它们的演变，采用了建立在过去所有的技术、原理和实践基础上，不仅保留其优点，而且还在包括面向对象手段和分布式对象手段的

同时，解决了其局限性。

SOA（Service Oriented Architecture，面向服务的架构）是一种新的面向服务架构的编程模型，号称"下一代软件架构"。它是一系列服务的集合，从软件和业务功能两方面的观点看，服务就是定义良好的软件组件，它不依赖于任何调用它的应用程序的上下文或状态。随着 SOA 标准规范的发布，它作为未来的发展趋势已无可争议，服务和构件是相互统一的完美结合体。因此面向构件的技术越来越备受关注，因此其相关的软件开发原理和实践指导活动就变得十分重要，如何高效的完成从构件系统的定义、设计、开发、组装、部署和维护是所有开发人员面临的问题。2005 年 11 月 30 日，BEA 宣布将与 IBM、Oracle、SAP、Iona、Siebel 和 Sybase 一起，支持一种构建和包装应用程序的新规范，即 Service Component Architecture（服务组件架构，SCA）。SCA 作为一种标准规范，将注意力集中在业务逻辑上，专门为 SOA 设计的，以面向构件为对象而涵盖整个软件生产过程和生命周期。

3　SCA 模型

SCA 组件模型由一系列的构件组成，这些构件采用 XML 文件中的元素来定义。SCA 运行时可以以非标准的其他方式表示这些构件，并且可允许动态修改系统的配置，由 XML 文件也是实现 SCA 构件的可以移植的表示方式。基本的工件是模块（Module），它是 SCA 的部署单位，包含了可供远程访问的服务。一个模块包含一个或者多个构件（Components），这些构件实现了模块提供的业务功能。构件以服务的方式向外提供它们的功能，这些服务可以被同一个模块中的其他构件访问，模块之外的构件也可以借助入口点（Entry Point）来访问这些服务。构件在实现自身的功能的时候可能会依赖于其他构件提供的服务，这种依赖关系被成为引用（Reference）。引用可以被关联到同一个模块中的服务，也可以关联到由其他模块提供的服务。被关联到的模块之外的服务，包括由其他模块提供的服务，在模块中被定义为外部服务（External Services）。在模块中也包含了引用和服务之间的关联关系，并且用连线（Wire）来表示。一个构件至少包含了一个经过配置的实现（Implementation）。实现就是一段程序代码，用来实现业务功能。构件为实现中可配置的属性配置了具体的值，这些可配置的属性是通过实现来声明的。构件也可以为实现所声明的引用配置连线，指向特定的目标服务。

模块通过 XML 文件部署在 SCA 系统（SCA System）中。一个 SCA 系统代表一系列的服务，它们提供了由某个组织所控制的一组业务功能。例如，对于一个企业中的会计部门而言，SCA 系统可能涵盖所有财务相关的功能。它可能包含一系列模块，用来处理特定领域的会计核算，一个用于客户账户，另一个处理应付账户。为帮助构建并配置 SCA 系统，子系统（Subsystem）被用来进行分组和配置相关的模块。子系统包含模块构件（Module Component），模块构件是配置后的实例。子系统，如同模块一样，也有入口点和外部服务。入口点和外部服务声明了存在于系统之外的外部服务和引用。子系统也可以包括连线，用来将模块构件、入口点及外部服务连接起来。

SCA 构件表示图用来可视化地描述在一个具体的组装模型中工件之间的关系。本文使用如下图表来辅助说明 SCA 构件的一些示例：

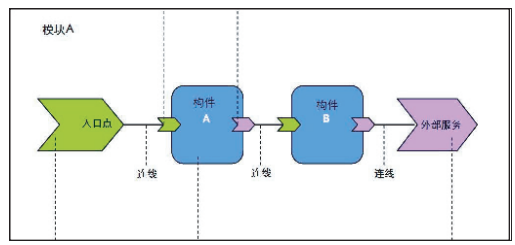

图 1　组装模型图

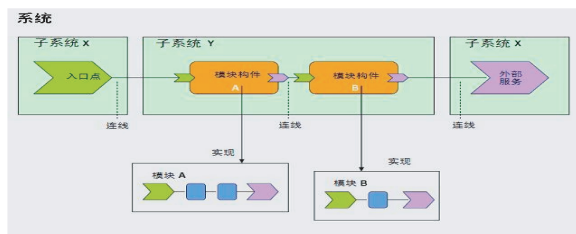

图 2　SCA 系统图

SCA 服务组件架构是最新发布的组件集成架构。SCA 体现的是一种利用通用组件定义方式来集成分散商业功能的思想。SCA 提供了一种统一的调用方式，从而使得客户可以把不同的软件模块通过服务构件的标准化而统一地封装起来和被调用访问。这种面向服务构件的编程模型可以大大简化客户的编程，提高应

用的灵活性。更直接地说,它是一种大大改进了的部署描述符,因此它不仅可以直接使用在过去传统的语言上,比如:Java,C + +,BPEL 等等,还可以运用在脚本语言上,例如:PHP 和 JAVASCRIPT 脚本,甚至声明式的语言上,比如:XQUERY 和 SQL 等。

SCA 通过连接线(Wire)连接和组装构件的各部件,比如构件类型(Component type),接口(Interface),组件(Composites),实现(Implement),引用(References)和服务(Service)等,实现各系统间的集成架构。其部署以和平常的部署文件类似,是用.composite 文件名结尾的 XML 文件来表示,其定义格式可以如下表示:

```
< ? xml version = " 1. 0"  encoding = " ASCII" ? >
  < composite  xmlns  = "  http :// www. osoa. org/
xmlns/sca/1. 0"  name = " xs :NCName" >
< include name = " xs :NCName" / > *
< service name = " xs :NCName"  multiplicity = " 0. . 1 or
1. . 1 or 0. . n or 1. . n" ? > *  < interface/ >
  < binding uri = " xs :anyURI" ? / > *  < reference >
wire – target – URI < / reference > +  < / service >
< reference  name = " xs :NCName"  override = " sca :
OverrideOptions" ?
  multiplicity = " 0. . 1 or 1. . 1 or 0. . n or 1. . n" ? > *
< interface/ >
  < binding uri = " xs :anyURI" ? / > *  < / reference >
< property  name  = "  xs : NCName "  type  = "  xs :
QName"  many  = "  xs : boolean "  ? override  = "  sca :
OverrideOptions" ? > *  default – property – value  < /
property >
< component  name = " xs :NCName"  > *  < imple-
mentation/ >
  < property  name = " xs :NCName"  source = " xs :
string"  ? file = " xs :anyURI"  ? > *  property – value  < /
property >
  < reference name = " xs :NCName" / > *  wire – tar-
get – URI  < / reference >
< / component >
  < wire > *
< source. uri > wire – source – URI < / source. uri >
   < target. uri > wire – target – URI < / target. uri >
  < / wire >
  < / composite >
```

显然,在定义中各构件的名字必须唯一,组件可以包括零至多种服务、构件、引用、连线和子组件或构件。构件包括了组件的业务逻辑,提供服务和参考引用等功能的描述与实现等等。服务通过组件对其它构件或系统提供和发布服务、引用和外部构件联系起来,最后通过连线把服务和引用连接,实现系统的集成和组合。

4　构件实现

下面以 Java 语言为例,具体说明怎样把现有的传统功能代码组装成 SCA 构件或引用来向客户提供服务,最后详细展示了采用非 SCA 代码发现和调用此 SCA 服务。

随着 JDK1. 5 的发布,Java 类可以通过服务声明式的语言(@ Service annotation)来发布对外提供的服务,其服务端的代码首先声明一个功能接口,然后再通过具体类来实现此接口,下面的代码实现天气预报服务的接口和具体类:

```
package zist. softlab. weather ;
public interface WeatherService  {//服务接口
String getWeather(String region) ;
}
package zist. softlab. weather ;
import org. osoa. sca. annotations. * ;
@ Service(WeatherService. class) //声明式服务
public  class  WeatherServiceImpl  implements  Weath-
erService  {
public String getWeather(String region)  {
. . . // 具体业务实现代码
}
}
```

下面的 XML 文件描述了服务端部署 WeatherService 类来实现构件的对外提供服务规范,它直接采用 WeatherService 类信息中进行转换,所以整个文件没有过多的冗余,非常的简洁明了,因此部署也非常的方便。

```
< ? xml version = " 1. 0"  encoding = " ASCII" ? >
< componentType  xmlns = "  http :// www. osoa. org/
xmlns/sca/0. 9" >
< service name = " WeatherService"  >
```

```
< interface. java interface = " zist. softlab. weather.
WeatherService" / >
</service >
</componentType >
```

其客户端调用服务过程可以采用引用注入和应用构件环境 API 来实现。引用注入的调用采用定义字段、设置方法参数或带参数的类虚构方法和@ Reference 声明式的语言实现,其包括引用名字和是否为必须注入服务二个属性。引用注入的客户端调用模型如下所示,应用构件环境 API 调用类似,在此再详述。

```
package zist. softlab. client ;
import zist. softlab. weather. WeatherService ;
import org. osoa. sca. annotations. * ;
@ Service(ClientService. class)
public class ClientServiceImpl implements ClientService {
private WeatherService weatherService ;
@ Reference ( name = " weatherService" , required =
true) //引用
public setWeatherService(WeatherService service) {
  weatherService = service ;
}
public void clientMethod() {
  String result =
weatherService. getWeather(" huanzhou" ) ;
}
}
```

客户端调用相应的部署 XML 文件如下:

```
< ?xml version = " 1. 0" encoding = " ASCII" ? >
< componentType xmlns = " http ://www. osoa. org/
xmlns/sca/0. 9" >
    < service name = " ClientService" >
      < interface. java interface = "
zist. softlab. client. ClientService" / >
    </service >
      < reference name = " weatherService " multiplicity
= " 1. . n" >
       < interface. java interface = "
zist. softlab. weather. WeatherService" / >
    </reference >
</componentType >
```

此外 SCA 组件模型还为 JAVA,C + + 语言提供了是否允许传递引用、调用返回、构件名称、构件元数据、上下文、对象销毁、初始化、会话、范围等声明语言来支持构件的发布和调用,加速了面向构件技术的有效开发,而其打包和部署过程完全采用了 J2EE 架构,可以以 JAR 或 EAR 的文件形式直接复制或上传到相关目录下即可实现发布和调用。

5 结束语

SOA 框架主要用于组合各项业务功能、流程和向外发布服务,以便实现复杂的业务应用程序和流程;它的可重用服务必须是松散耦合的,与平台和实现语言无关,通过其定义良好的接口和契约进行连接来实现服务的组装。无疑,面向构件的技术将是 SOA 中服务的组装和实现最佳实践。随着 SCA 规范的发布、越来越多的厂商的加入和用户的支持,SCA 组件模型简化了服务的构建与整合,在逻辑上也实现了把应用和中间件分开,在与数据编程架构的结合的情况下,将把软件过程实践方法带入一个新的阶段。

参考文献

1 黄柳青,王满红,等. 构件中国 – 面向构件的方法与实践. 清华大学出版社. 2006.

2 Dirk Krafzig, Karl Banke, Dirk Slama, etal. Enterprise SOA. Prentice Hall PTR. 2004.

3 Sreedevi Penugonda. 开发示例 SCA 应用程序. http ://www. ibm. com/. 2007. 01.

4 Techtarget . SCA/SD 加速 SOA 编程模型. http ://www. techtarget. com. cn. 2007. 01.

5 BEA 在线. JavaOne 大会上的服务组件架构. http ://dev2dev. bea. com. cn/. 2006. 6.

6 柴晓路,等. Web Services 技术、架构和应用. 电子工业出版社. 2003.

7 Service Component Architecture Specifications. http ://www. osoa. org/. 2007.