ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1201−1211
DOI: 10.3724/SP.J.1001.2008.01201

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于OSGi的服务动态演化*

张 仕[1,2+], 黄林鹏[2]

[1](福建师范大学 数学与计算机科学学院,福建 福州 350007)
[2](上海交通大学 计算机科学与工程系,上海 200240)

Dynamic Service Evolving Based on OSGi

ZHANG Shi[1,2+], HUANG Lin-Peng[2]

[1](School of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350007, China)
[2](Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200240, China)
+ Corresponding author: E-mail: shi@fjnu.edu.cn

Zhang S, Huang LP. Dynamic service evolving based on OSGi. *Journal of Software*, 2008,19(5):1201−1211.
http://www.jos.org.cn/1000-9825/19/1201.htm

Abstract: A method is proposed in this paper for resolving dynamic service evolving based on OSGi. First, use indirect method for updating service definition based-on OSGi. The method can make dynamic evolving transparent. Then, divide service definition into implementation and data, which makes fields in service instance consistent and evolvable. The adding/deleting service during evolving is also discussed. A project is implemented, which can direct how to design, execute and evolve updatable service. The methods mentioned in the paper may be helpful to evolving components, service and object-oriented software.
Key words: dynamic service evolving; Internetware; OSGi; indirect service; program designing method

摘 要: 提出一种解决 OSGi 平台上服务动态演化的方法.针对 OSGi 平台的服务动态演化提出了重定向方法,解决了服务类定义的动态更新,较好地满足了演化中服务的透明性问题;提出了实现和数据相分离的方法,解决了服务动态演化中公共数据的一致性问题和动态演化问题;探讨了服务动态演化中服务增、减等问题.对所提出的方法均通过实例说明了其设计、运行和更新的可行性.所提出的方法可以用于指导解决组件、服务和面向对象软件动态演化中的相关问题.
关键词: 服务动态演化;网构软件;OSGi;重定向服务;程序设计方法
中图法分类号: TP393 **文献标识码**: A

为了适应开放、动态、难控的网络环境的需求,软件系统开始呈现出一种柔性可演化、连续反应式、多目标适应的新系统形态[1].从技术角度来看,网构软件在 Internet 上展现为一种与当前信息 Web 类似的软件 Web.

* Supported by the National Natural Science Foundation of China under Grant No.60673116 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z166 (国家高技术研究发展计划(863)); the Natural Science Foundation of Fujian Province of China under Grant No.2007J0315 (福建省自然科学基金); the Technology Plan for Provincial University of Fujian Province of China under Grant No.2007F5037 (福建省科技计划支柱省属高校项目)

Received 2007-11-13; Accepted 2008-03-11

以软件构件等技术支持的软件实体将以开放、自主的方式存在于 Internet 的各个节点之上,任何一个软件实体可在开放的环境下通过某种方式加以发布,并以各种协同方式与其他软件实体进行跨网络互连、互通、协作和联盟.这样的软件结构也就决定了其开发主要是依赖于组成部分的各个构件.

构件的开发集成了面向对象机制,具有聚合、重用,支持并发性、持久性和分布性等特点.用户能够连接已经实现的来自不同软件开发商的构件功能来构成相应的应用,从而缩短开发时间和开发成本.实践证明,基于构件的中间件平台的互操作、分布性、开发方便性、可移植性特别适合于一些大型分布式系统的需求[2].因此,基于构件的软件复用作为一种提高软件生产率和软件质量的有效途径,是近几年软件工程界研究的重点之一,被认为是继面向对象方法之后的一个新的技术热潮[3].作为传统软件结构的自然延伸,网构软件具有区别于传统软件形态的独有基本特征,包括自主性、演化性、协同性、多态性、反应性等[4,5].

由于 Internet 的开放、动态和多变,以及用户使用方式的个性化要求,决定了网构软件在发布之后,其演化过程也在长时间内持续不断.同时,各个不同节点之间运行、调整和演化并不是独立的,它们需要相互协作.而网构软件系统本身往往由散布在 Internet 上的其他软件组成,作为其组成部分的节点需要提供不间断运行,即使对节点进行改错、优化、增加新功能等活动也需在线执行,所以,如何进行节点服务的动态演化就成了软件维护中的重要问题.

OSGi(open services gateway initiative)服务平台(service platform)规范提供了开放和通用的架构,可以作为部署软件服务的基础平台,OSGi 的 Bundle 实质上就是能够对外提供服务的构件.目前,OSGi 已经得到了 Eclipse 等相关大型软件机构的认可,又有其扩展——R-OSGi 提供远程调用支持,可以作为网构软件的节点而存在,因此具有较好的发展前景.本文通过对 OSGi 上服务的动态演化进行研究,提出重定向方法,解决了服务方法动态演化问题,保证了服务更新的透明性、服务的持续性和服务提供的连续性.其次,利用实现和数据相分离的方法,解决了服务并行执行中演化的难点问题——公共数据演化问题.对相应方法都通过具体实例用于指导服务的设计、演化,最后对相关问题进行讨论.对分布在 Internet 上网构软件的动态演化而言,本文实现了服务的动态更新透明性,保证了对该服务应用透明性.如果所有相关的节点都能实现这种透明性,那么网构软件的动态更新也就能够实现.

本文第 1 节主要介绍 OSGi 结构和动态演化中存在的问题,并对本文中用到的一些名词进行解释.第 2 节提出一种 OSGi 服务实现动态演化的解决方法,并通过实例说明如何设计可动态演化的服务类.第 3 节解决服务类数据域的共享、一致性及动态演化方面的问题.第 4 节通过实验对本文的动态演化方法进行分析,并对相关问题展开讨论.第 5 节对相关方面的研究工作进行小结.第 6 节对本文工作进行总结.

1 相关技术

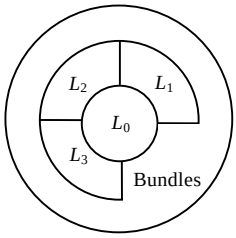
1.1 OSGi及基本概念

OSGi 服务平台规范提供了开放和通用的架构,使服务提供商、开发人员、软件提供商、网关操作者和设备提供商以统一的方式开发、部署和管理服务.OSGi 提供的灵活的服务部署机制和强大的管理功能增强了设备的智能性,通过增加远程调用 Bundle 的支持,使 OSGi 成为网构软件的一部分.

OSGi [6]R4 规范由基础框架(framework)、标准服务(standard services)、框架服务(framework services)、系统服务(system services)、协议服务(protocol services)及其他相关服务(miscellaneous services)组成.其核心框架(core framework)是 OSGi 最核心的部分,其组成如图 1 所示.

为了本文叙述更加清晰,这里首先给出相关名词的定义.服务类是指 OSGi 对外发布的服务所在类;服务对象/服务实例是指服务类的对象/实例;服务方法是指服务类中对外发布的服务对应的方法;服务方法运行是指服务方法的一个运行.公共数据特指服务类中的数据域,该数据域被该服务类中的所有服务方法共同使用,所以称为公共数据,在服务实例运行并对外提供服务时,实例中数据域对各个服务是共同使用的.每个服务类可以有多个服务方法,每种服务方法可以有多种服务方法运行.在 OSGi 的 Bundle 中,所有同一个服务类中的服务方法运行都是基于同一服务对象/实例的.Bundle 是 OSGi 加载、启动的基本单元,它可以由多个服务类组成.本文所

称服务动态演化是指在不中断现有服务方法运行的情况下,对现有服务实例进行更新,达到软件演化的目的.



L_0: Running environment (standard Java running environment)

L_1: Definition of class loader mechanism

L_2: Life Cycle support load, start, stop, update and unload for bundles

L_3: Service registry

Bundle: A Java program running on OSGi, a bundle can provide several service

Fig.1 The core framework of OSGi

图 1 OSGi 核心框架

1.2 服务演化相关问题

文献[7]中指出,软件组件演化看似非常简单,似乎只要把组件的实现进行替换即可,而实际上却涉及到许多问题,包括状态转换、上下文保持、文件访问、网络连接的保持等等.对于组件的动态更新,文献[8]总结了组件动态更新所必须具备的 3 个公共特点:请求缓冲(request buffering)、请求重定向(request redirection)和状态转换(state transfer).对于 OSGi 上的服务,在进行动态演化时也碰到类似的问题.就目前的技术现状而言,总结出 OSGi 的在线演化需要解决的主要问题有如下两个方面:

(1) 服务调用透明性问题.该问题是所有软件动态演化所需要考虑的重点,只有尽可能地实现客户端调用的透明性,才能真正达到动态演化的目的.请求缓冲、请求重定向的目的也就是为了尽量不影响客户端的使用,以达到透明演化.

具体到 OSGi 中,从 OSGi 对外提供服务和客户端调用机制对其进行分析,如图 2(a)所示.客户端对平台所提供的服务调用步骤主要分为:1) 查找,在 OSGi 平台上找到符合要求的服务;2) 取得服务引用(service reference),通过 getServiceReference()和 getService()函数得到相关服务引用;3) 调用,调用相关服务.OSGi 平台上的服务如果要进行替换,那么需要首先停止现有的该服务实例,注销后才可以注册同样的新服务实现.而且在调用中,服务方法通常都可以进行多次调用,如果在两次调用之间对相关服务实例进行了替换操作,停止了旧服务,那么服务引用将会指向错误位置,从而导致调用错误,具体如图 2(b)所示.

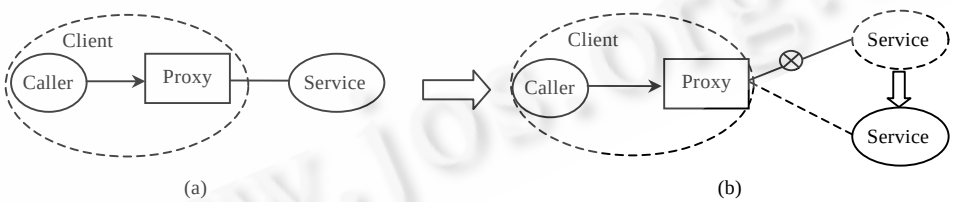


Fig.2 Dynamic service evolving of OSGi: Service replacement

图 2 OSGi 服务的动态演化:服务替换

(2) 状态转换问题,服务进行演化时需要对已有服务实例的状态进行保存,然后在新服务实例启动后利用这些信息进行恢复.对于新、旧服务实例共存的情况,还需要考虑如何在两个服务实例之间作好协调,保证它们之间是相容的,共享信息是一致的.

2 服务实例的动态演化

2.1 OSGi服务动态演化总体模型

本文采用重定向服务的方法实现服务的动态演化,其总体模型如图 3 所示.该方法仅针对服务方法实现变

化的情况,对于更加复杂的情况将在后面加以讨论.OSGi 的服务实现动态演化步骤描述如下:

(1) 通知重定向服务实例动态演化开始.

(2) 建立 New_Service,并把 IndirectService 的重定向变量指向 New_Service 服务实例.

(3) 对 IndirectService 的服务请求被重定向到新服务实例上.

(4) 更新操作结束.

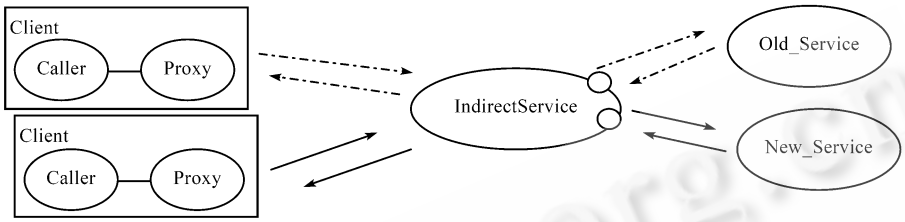


Fig.3 The model of OSGi service implementation evolving

图 3 OSGi 服务实现演化模型

2.2 OSGi服务动态更新

OSGi 中把服务的提供和消费理解为生产者和消费者.消费者通过注册的方式和生产者建立起 wire 关系,为简化起见,忽略了客户端从 OSGi 平台得到服务引用的过程,OSGi 客户/服务调用如图 2(a)所示.

为了实现对客户端透明的服务演化,总体而言有 3 种方法:

(1) 对 OSGi 平台进行修改,以得到系统平台的支持.由于 OSGi 对外提供同一服务的服务实例不能够同时存在多个,如果要进行演化、替换,首先就要把旧服务注销.旧服务的注销必然会影响到已有客户端的调用,也即破坏了更新的透明性要求.为了修改该限制,只有通过对 OSGi 平台进行相应的修改,而这破坏了 OSGi 的通用性,服务演化也就变成了平台相关.

(2) 在服务内部设定断点、状态收集等,再通过利用 Java 语言反射机制和 JVM 的深入分析实现服务的动态演化.该方法需要对原来的程序做大量工作,同时也很难找到通用的算法来实现批量式转换.

(3) 通过对相关服务进行包装,实现重定向功能.为了保证演化的透明性,就必须要保持已有服务实例不受影响,同时为了保证能够实现更新功能,所以在实际服务实例和客户端之间加入重定向服务实例 IndirectService,如图 3 所示.重定向服务实例主要实现把服务请求重定向到实际服务实例的服务方法上.这样,对客户端而言只能看到 IndirectService 上发布的服务,而不必关心服务内部的实现.在演化时,真正提供服务的对象属于非服务对象,其实现可以改变,并且可以多个实现共存,从而实现新、旧替换.以动态演化点为准,保持现有的实际服务实例及方法运行不变,但是在这之后的所有新创建服务方法运行则基于新定义的服务实例.

通过对以上 3 种可能的方法的分析,且考虑到服务更新实现难易等方面的问题,本文采用方法(3).该方法很好地解决了服务的并行性问题,保持了现有服务不中断,实现了服务提供的连续性.下面将通过一个具体的服务设计来说明如何设计、实现和利用基于重定向方法的动态演化.

2.3 实 现

图 4 中以类图的方式定义了重定向可动态演化 OSGi 服务类.该定义中把原来的实际服务类定义为重定向服务类中的一个变量,建立重定向类接口的主要目的是为了对外发布,隐藏一些不便对外提供服务的方法.而实际服务类 FooImpl 在 IndirectFooImpl 中作为接口声明,是为了方便实现类的演化.

系统需要对外发布的服务方法在重定向接口中加以定义,所以服务的注册也就是利用重定向类及其接口作为参数.Bundle 对服务的发布,就是在 Activator 类的 start 中注册该服务.对 IndirectFooInterface 而言,其接口中定义的方法 indirectFoo_1 被发布,并且可供其他 Bundle 调用,具体实现如下:

```
public void start(BundleContext context) throws Exception {
```

```
serviceReg=context.registerService(IndirectFooInterface.class.getName(),
                                   new IndirectFooImpl(), null);}
```

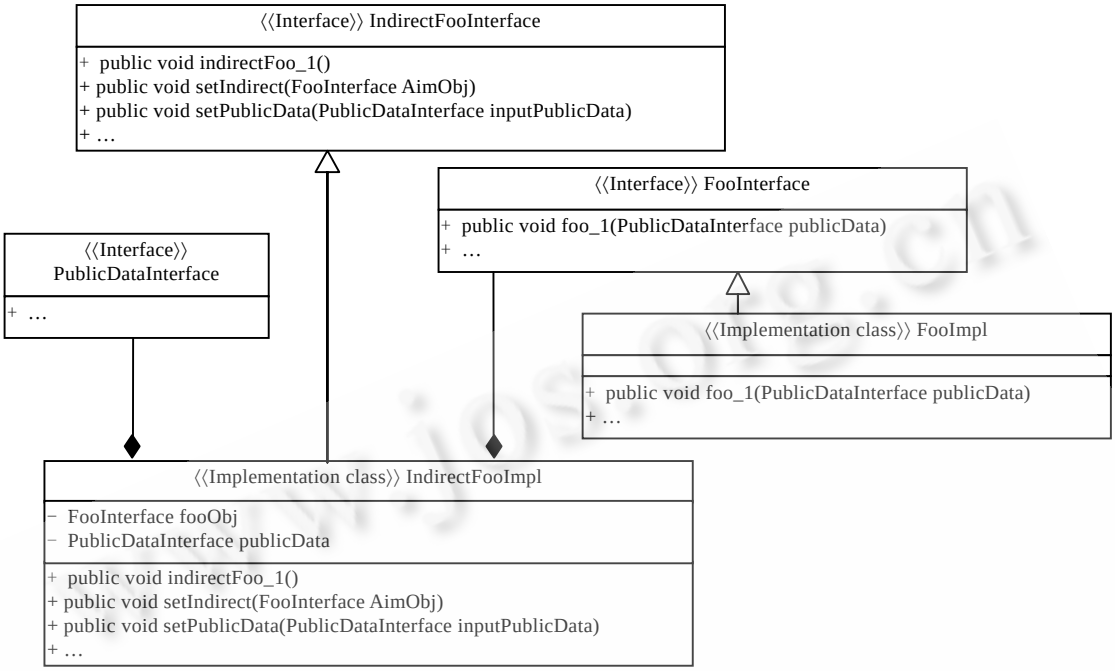


Fig.4 The Class figure for dynamically updatable service

图 4 可动态演化服务类类图

对客户端用户而言,其使用方法并没有发生任何变化,调用顺序仍旧为获取服务实例引用、利用接口进行变量转化、调用,其实现如下:

```
ServiceReference serviceRef=
        context.getServiceReference(IndirectFooInterface.class.getName());
Object service=context.getService(serviceRef);
IndirectFooInterface tmp=(IndirectFooInterface)service;
tmp.indirectFoo_1();
```

实际服务类接口的建立主要有助于在动态演化时定义同一接口但类名不同的类,可以很方便地实现替换操作,当然,也可以通过自定义类装载器加载同名类的新定义.如此,在载入新的服务实例时,无须对 JVM 或者类装载作出修改,其定义如下:

```
public interface FooInterface {
    public void foo_1(PublicDataInterface publicData) throws Exception;
     //实际服务方法接口声明
…}
```

实际服务类定义,主要定义了相关的实现.

```
public class FooImpl implements FooInterface {
    public void foo_1(PublicDataInterface publicData)throws Exception{…}
                                                    //实际服务方法实现
…}
```

重定向服务类接口主要用于对外提供服务接口.

```
public interface IndirectFooInterface {
    public void indirectFoo_1()throws Exception;
    public void setIndirect(FooInterface AimObj);
    public void setPublicData(PublicDataInterface inputPublicData);
… }
```

重定向服务类实现服务请求的重定向,并且通过两个 set 方法可以实现服务实例和公共变量的替换.

```
public class IndirectFooImpl implements IndirectFooInterface {
    FooInterface fooObj= new FooImpl();   //指向实际服务实例
    PublicDataInterface publicData=new PublicDataInterface();
                                                    //把实际服务实例中数据域封装、分离
    public void indirectFoo_1()throws Exception
        {FooObj.foo_1(publicData);}   //通过调用实际服务类方法实现重定向
    public void setIndirect(FooInterface AimObj){fooObj=AimObj;}
                                        //动态演化设置,通过重新设置重定向变量指向新的服务实例
    public void setPublicData(PublicDataInterface inputPublicData) {…}
                                        //可用于对共享变量进行更新,这里未实现该功能
…}
```

以上对 OSGi 服务实现方法上的变化使服务实现可以进行动态演化,并且能够保持现有的服务不中断.对实际服务实例上数据域的演化,特别是演化时一致性的维护将在第 3 节提出的方法中实现.

3 公共数据的处理

在 OSGi 的服务中,虽然多个客户端可以同时调用一个服务,但实际上只有 1 个类对象实例在运行(服务实例).多个服务方法对应 1 个服务实例,所以服务实例中的所有变量就成为服务方法的公共变量.

在服务动态演化时,实际服务实例和对外发布服务实例相分离,通过对 IndirectService 设置指向不同的实际服务实例(Old_Service 或 New_Service),从而实现服务的动态演化,如图 3 所示.当新、旧服务实例同时运行时,它们有各自的实例变量定义,但是不同服务实例的内部变量却不再能够相互共享,新、旧服务方法之间对公共变量的操作互不可见,容易引起新、旧服务实例数据的不一致性,所以,如何能够找到简单的方法解决好新、旧服务实例变量的不一致性和动态演化是服务动态演化的难点.通常情况下对该问题的解决是利用一些复杂的算法来进行协同,这显然增加了程序设计的难度.为此,本文提出了服务实现和公共变量相分离的方法.该方法能够较好地解决上述问题.

从前面的叙述中可以注意到,图 3 中实际演化并进行替换的是 Old_Service 而非对外的服务实例 IndirectService.所以,可以把实际服务实例中的变量进行封装加以定义,并放在 IndirectService 中.由此,即使 Old_Service 进行演化也不会影响到共享变量.在重定向服务方法调用实际服务实例方法时,只需把该封装后的共享变量作为参数进行传递.对新、旧服务实例采用这种方式可以很好地处理共享数据变量的不一致性问题.图 5(a)和图 5(b)为数据域与实现相分离的示意图.公共变量的更新主要是运用 Java 的子类和接口的方法,具体而言就是,首先把公共数据变量进行封装,然后增加一层接口定义取值操作.这样,只要是具有相同接口的对象均可被该变量所指.

封装后的数据域(公共变量)作为如下接口的对象.该接口主要包含一个取值操作,用于取封装在内的各个具体实例.

```
public interface PublicDataInterface {
```

```
    public Object getValue(String fieldName);
}
```

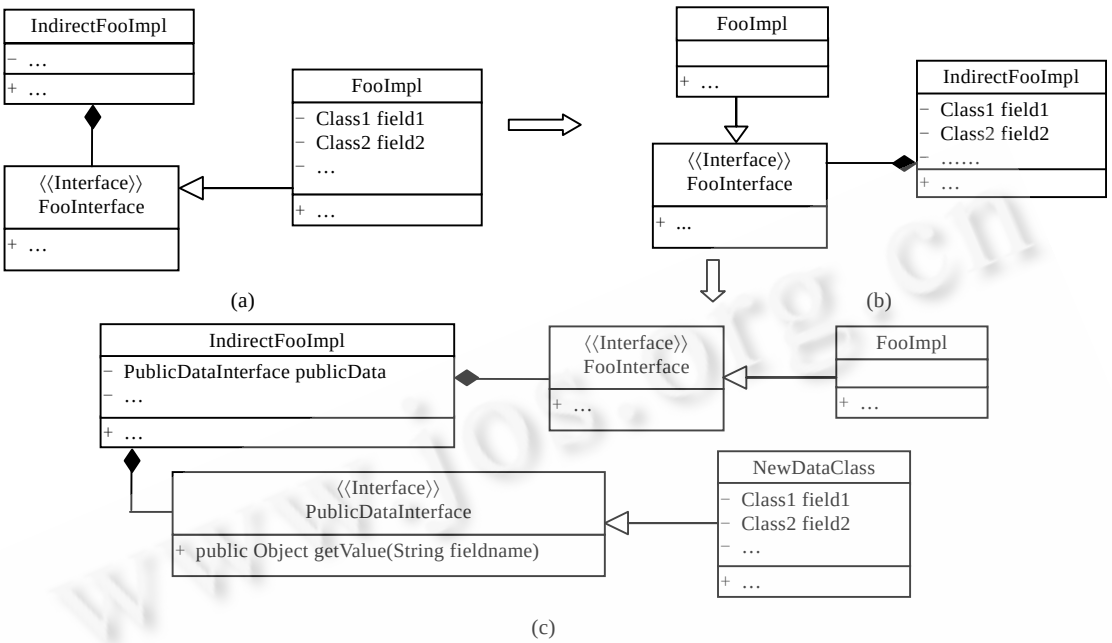


Fig.5 The change of fields' definition for service class

图 5 服务类数据域设计变换

在新、旧数据类定义中,同时继承了 OldDataClass 和实现了 PublicDataInterface,其中前者主要是为了兼顾新、旧服务实例的使用,后者则定义了一个统一的调用接口,以便变量可以在重定向服务类中声明后持续使用.新的数据类定义中可以增加相应的数据域,对于减少可能会引起异常.如果在 DataClass 中存在 int 等非对象变量,则需要利用对应的类型进行替换,例如,将 int 替换为 Integer.其具体类实现如下,类图设计如图 5(c)所示.

```
public class NewDataClass extends OldDataClass implements PublicDataInterface {
    Class1 field1=new Class1();
    public Object getValue(String fieldName) {
        if (fieldName.equalsIgnoreCase("field1")) return field1; else …}
}
```

对相关对象的引用则首先通过其对象名称作为参数,返回的值是 Object,然后进行强制类型转换,之后便可以像一般对象那样使用.于是,在实际服务实例的方法实现中,可以按如下示例方法访问需要的变量,并可以进行修改.其修改值可以在其他调用中反映,也就解决了公共数据一致性问题.

```
PublicDataInterface v1=new NewDataClass ();
Class1 tmp=(Class1)v1.getValue("field1");
```

在需要对公共数据对象进行演化时,可用来替换的对象的类定义形式如 NewDataClass 定义,同时在类定义中需要重新实现 getValue 函数.在替换过程中为了能够保证公共数据的一致性,需要定义锁机制,限制服务方法在状态转换过程中对相关变量操作.该方法只能实现数据的增量式动态演化,对于删除一些数据域的情况,可能会由于找不到数据域而引起程序产生运行异常.

4 实验及讨论

本节首先通过一个 OSGi 下服务动态更新实验对本文中所述方法进行验证,然后对本文中所述的动态更新方法作进一步讨论,分析其限制并提出相应的解决对策.

4.1 实 验

实验提供一个计算π值的服务,由客户端指定具体的精度要求,通过计算得到结果.实验所用计算机采用 CPU 为 AMD Sempron™ 处理器 2500+ 1.41GHz,内存为 512MB,带宽为 100Mbps 的局域网.运行操作系统为 Windows XP,所有代码均在 Eclipse 下实现.

实验首先对比了可动态更新程序(重定向服务)与原程序的运行效率,如图 6(a)所示.分别对两种情况的服务都运行两次,*X* 轴表示运算输入的精度要求,如 8×50.从实验数据上可以看出,两种情况下服务的运行效率并没有什么明显不同.我们还专门针对其中 3 个精度运行 10 次,然后按照总时间进行比较,3 次实验下两次直接服务方式慢 2ms,1 次间接服务方式慢 3ms,所以排除计算机系统的影响,可以认定在本文实验中,重定向服务方式对它们的性能没有产生影响.接着,图 6(b)中反映的是客户端连续调用服务 99 次,输入精度要求为 400,它们的响应时间曲线.在调用第 25 次后、第 52 次后、第 81 次后分别进行了 3 次类动态更新,为了能够对比它们的性能,仅仅调整一些标识数据.从实验中可以注意到,在第 1 次和第 3 次动态更新后响应时间为 111ms,第 2 次动态更新后服务响应时间为 110ms,除了两个点上响应时间为 94ms 以外,其他情况下服务响应时间有 109ms 和 110ms 两种情况.通过多次实验,结果类似,由此我们可以得出结论,服务动态更新对客户的调用产生的影响有限,可以满足应用要求.

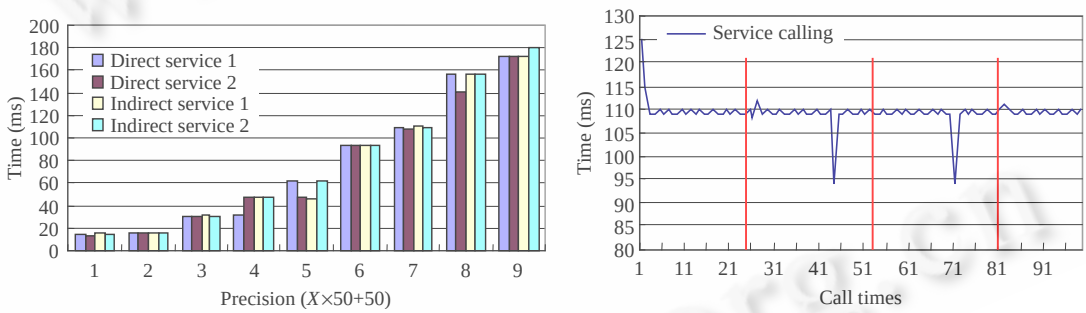


Fig.6 Comparison between raw and updatable program

图 6 原程序与可动态更新程序运行对比

4.2 讨 论

本文的服务动态演化过程中不考虑请求缓冲的主要原因在于,修改 IndirectService 从旧服务对象 Old_Service 到新服务对象 New_Service 时,实际对外的服务并没有停止,而且在变量指向变化时系统会自动维护,以保证不会出错.当然,在进行公共数据转换工作时则需要在转换过程中对相应对象加锁,以保证执行的原子性.OSGi 保持旧服务实例 Old_Service 继续执行,不需要演化过程干预,在所有已请求服务都执行完毕后,该服务实例及其定义会被 JVM 收集并销毁.

前面已经提出基于 OSGi 的动态演化方法并解决了公共变量共享问题.这里将对上文所提出的服务动态演化方法的深入问题进行讨论,并提出可行的方法指定相应的实现.

(1) 重定向类的演化问题.本文为了能够实现服务的动态演化,增加了一层重定向类,根据前面的分析,如果要实现重定向类的演化就需要再增加一层,而这无法从根本上解决问题,所以限定重定向类是不可演化的.

(2) 类增加对外服务.这种情况是指服务定义所在类如果增加一些方法,并且要把这些方法也作为服务对外发布.在这种情况下,我们有两种方法可以选择,一种是直接把该类新增的方法作为服务进行注册;另有一种是再增加一个 indirectService,然后把新服务由其重定向进行发布.第 1 种方法产生的问题是直接发布在使用时会由系统自动产生一个新的服务实例,由此,OSGi 上同时存在两个该类实例,使它们难以协调公共变量;第 2 种方法也涉及到该问题,但是可以通过两个 indirectService 之间的交互来解决,然而其实现会受到较多的限制.总体而言,比较适用于无共享变量的情况.

(3) 类减少对外服务.如果全部取消,那么自然可以通过注销 IndirectService 来处理;如果要保持部分服务,那么只能在实现上把 Service 类该方法设置为空方法.该方法没有从根本上去除该服务接口.

5 相关研究

网构软件是由分布的组件、服务等组成的,因此对网构软件的动态演化研究也就转化为对服务、组件等基本元素的动态演化.OSGi 的 Bundle 可以说是一种特殊的组件,它们的动态演化在很大程度上具有相似性.本节主要对组件更新的相关研究作一阐述.

对于组件的演化,有许多系统的方法,比如基于软件体系结构的方法、基于 Agent 的方法等.Plasil 在文献[9]中介绍了 SOFA 的动态组件更新(dynamic component updating,简称 DCUP)方法.DCUP 提供了一组相互正交的抽象,从而支持组件在运行的应用中进行更新.Postma 在文献[10]中介绍了一种 3RDBA 方法,可用于替换长时间运行系统中的组件,还可根据所作的演化取得各种必要的信息.OSGi 本身提供了较粗粒度(bundle)的演化,它允许在 OSGi 平台上运行不同版本的 Bundle,但却不能实现对用户的透明性.文献[11]对 OSGi 上的软件组件动态更新进行研究,提出了 bridge 方法,但是它对组件的定义有较多的限制,其中最主要的是对类数据域的定义上.文献[12]则针对整个组件的同时更新会对系统产生较大的影响,提出仅仅对组件中的某些部分进行更新的方法,同时该方法允许多个版本的组件同时运行,这样,多个组件上都提供相关的功能自然对于整个软件系统有更好的容错性.

基于 CORBA 的运用已有很长的时间,所以这方面的研究也较多.Tewksbury 等人[13]描述了在 Eternal 系统中的 CORBA 应用的更新.他们使用服务器对象的复制来提供在更新期间不被中断的服务.对象被逐个用中间过渡版本更新,这个版本成为新、老版本中间的一个过渡版本.Bidan 等人[14]为基于 CORBA 的分布式系统 Aster 描述了一个动态可重配置管理器,提出的更新机制不是要求对象在更新前是被动静止的,而是要求对象之间的连接是处在被动状态的.作者为重配置定义了形式化的一致性和效率的限制,论证了它们的重配置算法是效率优化的.Almeida 等人[15]描述了 CORBA 中的动态更新.它们和 Bidan 等人的工作的区别是:(1) 支持重新进入的调用;(2) 支持多个对象的原子更新;(3) 使用 ORB 扩展了的比较大的透明性.Balasubramanian[16]通过在中间件平台 SwapCIAO 上扩展轻量级 CORBA 组件模型的服务功能来实现组件的动态更新,解决一个物流帐目清单跟踪系统(ITS)的组件动态更新功能,但其动态更新机制相对简单.

在对动态组件体系结构演化的研究上,Bialek 将动态组件体系结构和动态软件更新[17]结合起来,提出了一个支持组件运行时重配置的更新体系结构.王晓鹏等人[18]借助 JAVA 平台的类装载机制提出了对组件化软件进行在线演化的方法.Shen 等人[8]基于进程演算 CSP,为动态组件更新机制构建了一个平台独立的统一形式化模型 DUC 来表示体系结构组件的更新机制,在一定程度上对开发通用的动态更新形式化方法作出了很好的尝试.Oreizy 等人在软件体系结构层次上提出了支持运行时软件更新的方法,高度抽象了运行时更新的处理操作,将有关应用具体的行为和动态更新策略分开[19].Magee 等人[20]的工作主要集中在动态更新的管理工作上,比如,管理组件的增加、删除和重新配置组件间的链接,其提出的方法分离了软件体系结构和相关应用组件功能的实现,允许改变软件结构但不考虑具体应用的状态在更新前后的一致性.

此外,在不同的应用领域,比如电子商务、嵌入式系统[21,22]也有基于组件的在线演化研究进展.Gardler 在文献[21]中就提出了一种在线电子商务支持系统的半自动化组件在线演化方法.Vardewoude 在文献[22]中对名为 SEESCOA 的基于 Java 的嵌入系统进行了探讨,使用端口的概念将消息在组件间进行重定向,已装载类也包含

版本信息,并可将所有组件的实例立刻更新.

6 结 论

软件的动态演化是近年来软件工程领域关注的重要问题,而网构软件则又是软件发展的趋势之一.基于此,本文主要致力于解决 OSGi 平台上服务实例的动态演化问题.首先分析了 OSGi 平台的基本结构和服务演化需要解决的透明性问题.然后针对 OSGi 平台的服务动态演化提出了重定向方法,解决了服务类定义的动态更新,较好地满足了演化中服务的透明性要求.接着提出了用实现和数据相分离的方法,解决了服务动态演化中公共数据的一致性问题和公共数据定义演化问题.最后对在服务动态演化中服务接口增、减等问题进行了初步探讨.文中对重定向方法和数据与实现相分离的方法分别用实例说明了它们的设计、运行和更新.本文所提出的相关方法可以用于指导组件、服务和面向对象软件动态演化中的相关问题.

对OSGi服务动态演化,我们还将从两个方面深入研究:(1) 将致力于自动实现对现有服务程序的改造,使本文提出的方法能够得到推广;(2) 把相关技术运用于 R-OSGi 上,从协调、互动更新方面对网构软件的演化问题进行研究.

References:

[1] Lü J, Ma XX, Tao XP, Xu F, Hu H. Review of Internetware. Science in China (Series E), 2006,36(10):1037−1080 (in Chinese with English abstract).

[2] Ma XX, Yu P, Tao XP, Lü J. A service-oriented dynamic coordination architecture and its supporting system. Chinese Journal of Computers, 2005,28(4):467−477 (in Chinese with English abstract).

[3] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Journal of Software, 2003,14(4):721−732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[4] Mei H, Huang G, Zhao HY, Jiao WP. An Internetware developing method centered on software structure. Science in China (Series E), 2006,36(10):1100−1126 (in Chinese with English abstract).

[5] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002,30(12):1901−1906 (in Chinese with English abstract).

[6] OSGi service platform release 4. 2006. http://www2.osgi.org/Specifications/HomePage

[7] Adamek J, Plasil F. Component composition errors and update atomicity: Static analysis. Journal of Software Maintenance and Evolution: Research and Practice, 2005,17(5):363−377.

[8] Shen JR, Sun X, Huang G, Jiao WP, Sun YC, Mei H. Towards a unified formal model for supporting mechanisms of dynamic component update. In: Michel W, Harald G, eds. Proc. of the ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 2005. 80−89.

[9] Plasil F, Balek D, Janecek R. SOFA/DCUP: Architecture for component trading and dynamic updating. In: Proc. of the Int'l Conf. on Configurable Distributed Systems. Annapolis: IEEE Computer Society, 1998. 35−42.

[10] Postma A, America P, Wijnstra JG. Component replacement in a long-living architecture: The 3RDBA approach. In: Proc. of the 4th Working IEEE/IFIP Conf. on Software Architecture (WICSA). Oslo: IEEE Computer Society, 2004. 89−100.

[11] Pohl HW, Gerlach J. Using the bridge design pattern for OSGi service update. In: Proc. of EuroPLoP 2003. http://www.hillside.net/europlop/europlop2003/papers.html#WorkshopD

[12] Cook JE, Dage JA. Highly reliable upgrading of components. In: Proc. of the Int'l Conf. on Software Engineering. IEEE Computer Society, 1999. 203−212.

[13] Tewksbury LA, Moser LE, Smith PMM. Live upgrades of CORBA applications using object replication. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM). Florence: IEEE Computer Society, 2001. 488−497.

[14] Bidan C, Issarny V, Saridakis T, Zarras A. A dynamic reconfiguration service for CORBA. In: Proc. of the Int'l Conf. on Configurable Distributed Systems. Annapolis: IEEE Computer Society, 1998. 35−42.

[15] Almeida JPA, Wegdam W, Sinderen M, Nieuwenhuis L. Transparent dynamic reconfiguration for CORBA. In: Proc. of the 3rd Int'l Symp. on Distributed Objects & Applications (DOA 2001). Rome: IEEE Computer Society, 2001. 197−207.

[16] Balasubramanian J, Natarajan B, Schmidt DC, Gokhale A, Parsons J, Deng G. Middleware support for dynamic component updating. In: Proc. of the Distributed Objects and Applications (DOA) 2005 Int'l Conf. LNCS 3761, 2005. 978−996.

[17] Hicks M, Nettles S. Dynamic software updating. ACM Trans. on Programming Languages and Systems, 2005,27(6):1049−1096.

[18] Wang XP, Wang QX, Mei H. An approach to online evolution of component based software. Chinese Journal of Computers, 2005, 28(11):1891−1897 (in Chinese with English abstract).

[19] Oreizy P, Medvidovic N, Taylor RN. Architecture-Based runtime software evolution. In: Proc. of the 20th Int'l Conf. on Software Engineering. Kyoto: IEEE Computer Society Press, 1998. 177−186.

[20] Magee J, Kramer J. Dynamic structure in software architectures. In: Proc. of the ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1996. 3−14.

[21] Gardler R, Mehandjiev N. Supporting component-based software evolution. In: Aksit M, ed. Proc. of the Int'l Conf. NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World. London: Springer-Verlag, 2002. 103−120.

[22] Vandewoude Y, Berbers Y. Run-Time evolution for embedded component-oriented systems. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM). Montreal: IEEE Computer Society, 2002. 242−245.

附中文参考文献:

[1] 吕建,马晓星,陶先平,徐锋,胡昊.网构软件的研究与进展.中国科学(E 辑),2006,36(10):1037−1080.

[2] 马晓星,余萍,陶先平,吕建.一种面向服务的动态协同架构及其支撑平台.计算机学报,2005,28(4):467−477.

[3] 梅宏,陈锋,冯耀东,杨杰.ABC:基于体系结构、面向构件的软件开发方法.软件学报.2003,14(4):721−732. http://www.jos.org.cn/1000-9825/14/721.htm

[4] 梅宏,黄罡,赵海燕,焦文品.一种以软件体系结构为中心的网构软件开发方法.中国科学(E 辑),2006,36(10):1100−1126.

[5] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12):1901−1906.

[18] 王晓鹏,王千祥,梅宏.一种面向构件化软件的在线演化方法.计算机学报,2005,28(11):1891−1897.

张仕(1977－),男,福建龙岩人,博士生,讲师,CCF 会员,主要研究领域为程序设计语言,数据集成.

黄林鹏(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式系统,程序设计语言.