维普资讯 http://www.cqvip.com

第18卷 第10期
2008年10月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol.18 No.10
Oct. 2008

Elastos上AOP编程模型的研究

左 鹏,陈 榕
(同济大学 基础软件中心,上海 200092)

摘 要:AOP作为一个基于构件技术的软件开发模型,是对OOP的补充和完善。如何把AOP和现有的构件技术结合起来成为问题的关键。Elastos是基于构件的操作系统,提供了一种基于二进制的AOP的实现,能够灵活地实现基于构件级别的代码的动态插入、拦截,从而提供构件的动态组合以及实现各种功能。介绍了在Elastos上利用CAR构件技术实现AOP编程模型的方法。相比于其它的静态聚合,基于CAR构件系统的AOP模型具有随时聚合、随时拆卸的特点,是真正的面向方面的编程模型。

关键词:Elastos; 面向方面编程; CAR构件
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2008)10-0009-04

Research of AOP Programming Model on Elastos

ZUO Peng, CHEN Rong
(System Software Engineering Centre of Tongji University, Shanghai 200092, China)

Abstract: As a component-based software development model, AOP supplements and improves OOP. How to combine AOP and current component technique becomes the key problem. Elastos, a component-based operating system, provides a binary-based implementation of AOP. It can insert and remove codes on component level, accordingly providing dynamical combination and some other functions. Introduces how to implement AOP programming model using CAR component technique on Elastos. Compared with other static aggregation, the AOP model based on CAR component system that can aggregate and disaggregate at runtime is the real aspect-oriented programming model.

Key words: Elastos; AOP; CAR component

0 引 言

AOP(Aspect Oriented Programming)是在OOP(Object Oriented Programming)的基础上引入关注点(Concern)概念的一种编程方法。所谓关注点是指系统中的一些横向逻辑代码,如调试、权限管理、缓存、记录跟踪等。这些代码一般不代表系统的主要逻辑,但会分散在许多模块当中与其他代码缠结在一起。在AOP中称这些代码为横切关注点,也就是方面(Aspect)。AOP通过封装关注点,把这些操作与业务逻辑分离,使程序员在编写程序时可以只专注于业务逻辑的处理。因此,AOP技术是实现关注点分离、改善系统逻辑、减低软件开发难度、提高软件开发质量和软件重用性的良好方法。

收稿日期:2008-01-11
基金项目:国家"863"计划资助项目(2001AA113400)
作者简介:左 鹏(1985-),男,安徽池州人,硕士研究生,研究方向为嵌入式操作系统、系统软件支撑技术;陈 榕,博士生导师,教授,研究方向为嵌入式系统、构件技术。

Elastos[1]是自主知识产权的先进的操作系统。其上的CAR[2]构件技术定义了一套网络编程时代的面向构件的编程规范,实现了软件目标代码级的应用。它提供了一种基于二进制的AOP的实现,能够灵活地实现基于构件级别代码的动态插入、拦截,从而能提供构件的动态组合、扩展,以及实现各种功能。CAR的AOP机制使用户能够在完全不用修改源代码的情况下简单,方便地将一个一般CAR构件类与一个或多个Aspect方面构件类动态聚合起来,从而生成一个具有两个或多个CAR构件类所有接口的新构件类。同一般的AOP技术相比,极大地降低了系统设计的复杂性,提高了软件的灵活性。

1 Elastos中AOP的编程规范

1.1 Aspect方面构件

aspect构件是一种特殊的构件类实现,aspect对象的特征是可以被普通构件对象聚合,在实例化时由上下文环境构件负责聚合到功能构件。在Elastos AOP编程模型中,aspect可以作为一个关键字用来定义方

维普资讯 http://www.cqvip.com

面构件类;也可以作为一个属性用来修饰一个 context 语境,说明此 context 语境具有某个或某几个 aspect 方面特性。一个可被聚合的 aspect 对象必须实现两个接口,一个用于委托 IObject ,一个用于实现真正的 IAspect 接口,也就是委托和非委托接口,且非委托接口并不从 IObject 接口继承[3]。在 Elastos 中, aspect 构件对象还可以通过 Attach 和 Detach 等方法灵活地实现聚合任务。

由于"aspect"在问题域中是属于"属性"的抽象的概念,并不是一个可以独立存在的实体,因此它只能作为一个主语(某个构件对象)的谓词存在。因此,在 Elastos AOP 中,禁止用户使用非 aspect 与非 aspect 对象的聚合、aspect 与 aspect 对象的聚合、aspect 对象的创建等不合理做法。

1.2 Aggregate 聚合

真正的面向方面的聚合模型是可以做到随时聚合、随时拆卸的聚合。基于构件的 Elastos 操作系统利用 car 构件技术实现了对 AOP 动态聚合的支持。动态聚合是构件对象在运行时随着运行环境的改变聚合其它的构件对象。虽然现在的静态聚合能够满足一定的应用需求,但在现实模型中,更多的情况却是动态聚合[4]。如,一个人进入了学校,学校这个环境就会把学号、班级等属性与之聚合在一起;进入公司,公司就会把员工相应的属性与之聚合在一起。

1.3 Context 上下文语境

Elastos 中用 context 作为关键字来定义一个语境构件类。context 上下文语境是对象运行时的某种特定的状态,它是基于动态聚合实现的一种技术。一个 context 上下文语境至少包含一个 aspect 对象,所谓的具有环境特征或失去环境特征,就是语境会为进入此语境的对象动态地聚合或拆卸一个或多个 aspect 对象。这些 aspect 对象是语境在 car 文件中定义时指定的。一个普通构件对象如果进入了一个 context,那么该对象将具有此 context 的属性,即 aspect,一旦对象离开了该 context,环境属性就会失去(但该对象很有可能又进入了另外一个 context, 拥有新的环境属性)。

比如宠物店的猫(构件对象),具有宠物和商品两方面的特征(Aspects)。但猫刚生下来时却未必是宠物,更不会是商品;而一旦被售出,就不再是商品,但宠物特征却保留了下来;护士在医院里是护士,回家后可能就是贤惠的妻子等等。可以看出,随着构件对象的环境(context)的改变,构件对象所具有的特征(Aspects)也可能是不一样的。

1.4 Container 包容

在 Elastos 中,CAR 构件实现了包容技术,即封装一个或多个构件对象。它可以通过对象包容器构件类 AObjectContainer 来对被包容在一个构件对象里的其它多个构件对象进行管理并实现一些通用的接口。AObjectContainer 是 Elastos 系统提供的一个特殊的方面构件,它作为一个外部构件可以包容若干个同类的或不同类的其他构件,这些其他构件作为内部构件被包容在 AObjectContainer 构件之中。AObjectContainer 构件包含指向内部构件接口的指针,此时 AObjectContainer 构件仅仅是内部构件的一个客户而已。它一般通过对象枚举器来枚举内部构件对象。AObjectContainer 定义了一个 IObjectContainer 接口,包含了一组方法如 Add(IObject * pObject), Remove(IObject * pObject)等,实现向 container 中添加或删除构件对象等操作。

2 Elastos 中 AOP 的编程规范

AOP 用到的最主要的实现思想是通过建立一个对象代理,该对象代理提供给用户调用的接口,然后根据用户调用的接口方法以及用户指定的绑定形式,按照用户设定的次序调用绑定,替代方法或者真正的对象接口方法实现。实现逻辑如图 1 所示。

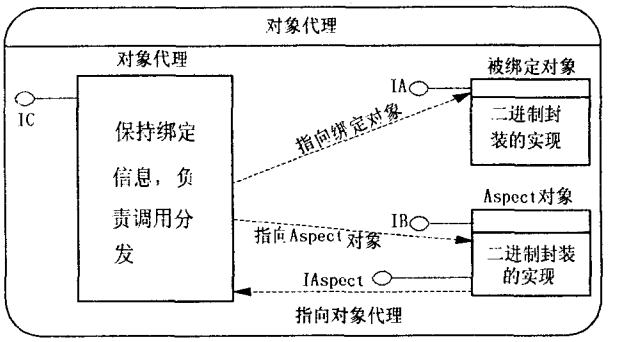

图 1 对象代理示意图

用户通过图 1 中对象代理暴露的 IC 接口(当然可以是多个接口)进行调用。对于 A,C 两种情况, IC 接口的定义则完全和被绑定对象暴露的接口 IA(当然可以是多个接口)相同。对于 B, IC 接口聚合了 IA 接口以及 Aspect 对象所暴露的接口 IB(当然可以是多个接口)。对象代理将根据用户事先设定的绑定策略依次调用相关方法或者函数[5]。在 Elastos 中,CAR 构件的每个接口都是由 IObject 继承而来的,CAR 中 IObject 接口的定义如下:

```
IObject {
  virtual CARAPI QueryInterface(//
    /* [in] */ RIID riid,
    /* [out] */ POBJECT * ppObject) = 0;
  virtual CARAPI_(ULONG) AddRef() = 0;
    //增加引用计数
```

维普资讯 http://www.cqvip.com


```
virtual CARAPI_(ULONG) Release () = 0;
  //减少引用计数
virtual CARAPI Aggregate(
  //动态聚合,一般用户不会直接调用该方法
    /* [in] */ AggregateType type,
    /* [in] */ POBJECT pObject) = 0;
};
```

以上每个方法的实现都在定义接口的类里,由框架自动生成。

2.1　动态聚合

动态聚合是通过 IObject 的 Aggregate 方法来完成的,因此构件编写者定义的每个构件对象都具有聚合其他 aspect 对象的能力。实现逻辑如图 2 所示。

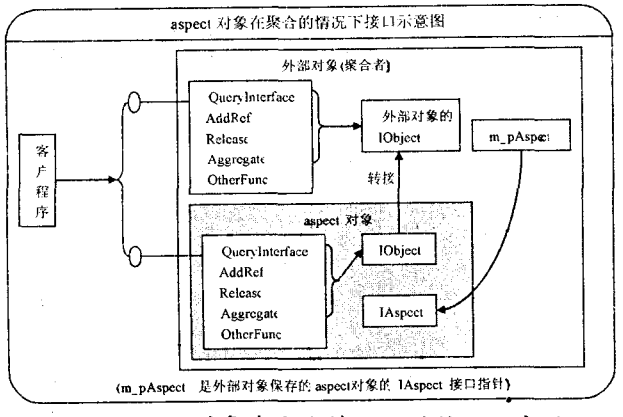

图 2　aspect 对象在聚合情况下的接口示意图

Aggregate 方法在聚合时是引发了一系列的调用来完成聚合过程的,下面是动态聚合的实现过程:

假设有一个构件对象 A(构件类为 CA)和一个 aspect 对象 B(构件类为 AB),并且在构件对象 A 里要聚合 aspect 对象 B,那么只要通过如下过程就可完成聚合任务(为了简便,将外部对象指针用 pOuter 表示,pAspect 表示 IAspect 指针)。

1)外部对象 A 调用:

```
pAspect->Aggregate(AggrType_Aggregate, pAggregator);
```

2)aspect 对象 B 及时反调外部对象的 Aggregate 方法:

```
pOuter->Aggregate(AggrType-AspectAttach, pAspect);
```

3)外部对象保存 aspect 对象指针;

4)aspect 对象保存外部对象指针并将所有的引用计数全部转嫁到外部对象,聚合过程完成。

可以看出,Aggregate 过程是在 CA 里调用了 CB 的 Aggregate 方法,其 AggregateType 为 AggrType_Aggregate,然后在内部对象 CB 里又及时反调了外部对象的 Aggregate 方法,这时的 AggregateType 为 AggrType_AspectAttach,这个过程可称之为 attach 过程,attach 过程保存了 aspect 对象的 IAspect 接口指针。

通过上面的聚合过程,外部对象就有了指向 aspect 对象的指针(m_pAspect),被聚合的 aspect 对象也有了指向外部对象的指针(m_pOuter)。

2.2　多面动态聚合

上面介绍的只是两个对象的聚合,外部对象只聚合了一个 aspect 对象。实际上,在面向方面编程时,往往需要一个对象聚合多个 aspect 对象,这就是多面聚合。在完成多面聚合时,实现上并没有多大的变化,就是创建多个 aspect 对象,多次调用 Aggregate 方法使一个对象聚合多个 aspect 对象。类似地,多面聚合的结果如图 3 所示。

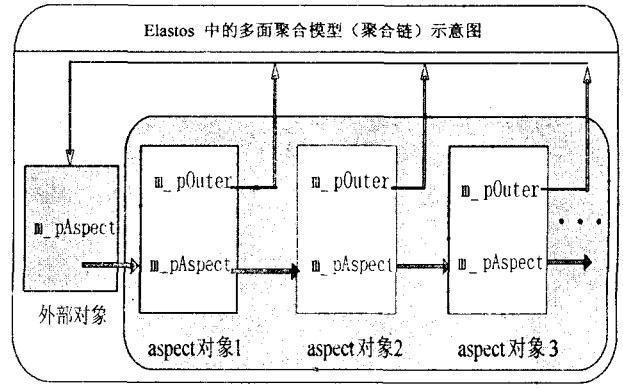

图 3　Elastos 中的多面聚合模型示意图

可以看出,构件对象中指向下层被聚合的 aspect 对象的指针(m_pAspect)构成了一个单向链表。而链表中的每一个 aspect 对象的 m_pOuter 指针都指向最外层的外部对象(聚合者)。通过这种方式,QI 调用可以在链表中从头向尾传递,而所有的 aspect 对象的引用计数都委托给了最外层的外部对象。这与前面所介绍的两个对象的聚合情况并没有多大区别。

当然,上图是一种比较理想的聚合情况,在某些情况下,下层的 aspect 对象的 m_pOuter 指针可能会指向中间的 aspect 对象,最坏的情况是每一个 aspect 对象的 m_pOuter 指针都指向它相邻的上层对象。虽然这样也能正确运行,不影响执行的结果,但运行效率却十分低下。

多面聚合在实现上与两个对象聚合的情形最大区别就是 attach 过程的实现。在某时刻,外部对象聚合一个 aspect 对象时,在 attach 的时候,外部对象首先检查自己是否已经聚合其它 aspect 对象,如果有(m_pAspect 不为空),就继续向 aspect 对象 1 attach,aspect 对象 1 又会检查自己的 m_pAspect,如果为空则保存将要被聚合的 aspect 对象指针,否则继续向聚合链里的下一个 aspect 对象 attach,直到聚合链的最后一个 aspect 对象。

2.3　动态拆卸聚合

CAR 构件库提供了 Unaggregate 函数来实现动态

维普资讯 http://www.cqvip.com

拆卸聚合：

```
EZAPI Unaggregate(
  /* [in] */ PObject pAggregator,
  /* [in] */ RCLASSID rAspectClsid );
```

pAggregator 为外部对象，rAspectClsid 为 aspect 对象构件类的 CLASSID 标识，这个函数可以使外部对象动态地拆卸某个已被聚合了的 aspect 对象，其大致的实现过程如下：

1)外部对象通过 aspect 对象标识 QI 出相应的 aspect 对象 IAspect 接口指针：

```
pAggregator->QueryInterface(
  (REFIID)rAspectClsid,
  &pAspect );
```

由于 IAspect 接口没有 IID 与之对应，IAspect 接口对上层也是透明的，所以通过 aspect 对象的构件类来标识 QI。虽然 CLSID 只是用于标识构件类，一般情况下不能标识对象，但对被聚合的 aspect 对象却可以，因为聚合同一个 Aspect 构件类的多个对象实例是没有意义的。如果当前对象找不到与之匹配的类标识，那么会传替给聚合链里的下一个对象，直到匹配为止，并返回对应的 IAspect 指针。

2)与 rAspectClsid 匹配 aspect 对象断开自己并维持剩余的聚合链表。

3)还原引用计数。由于在聚合时 aspect 对象将引用计数完全转嫁到了外部对象，所以在拆卸的时候必须还原，还原的过程就是聚合时转嫁的相反过程。

3 结束语

介绍了如何将 Elastos 的 CAR 构件技术和 AOP 结合起来，以及 Elastos 上 AOP 的编程模式和实现方法。利用 AOP 技术分离关注点，不仅方便了开发人员，更极大地提高了系统的灵活性和重用性。可以看到，基于 CAR 构件的 AOP 编程模型简单而灵活，可随时聚合，随时卸载，是真正的面向方面的聚合模型。本研究成果已经在 TD-SCDMA 手机中应用。

参考文献：

[1] 科泰世纪. Elastos 资料大全[EB/OL]. 2003-09. http://www.elastos.com.cn.

[2] Koretide. CAR's Manual[M/CD]. 2005-06. http://www.koretide.com.cn. 2004/2005.6.

[3] Chen Rong. The Application of Middleware Technology in Embedded OS[C]//Workshop on Embedded System, In Conjunction with the ICYCS(6th). Hangzhou:[s.n.],2001.

[4] 潘爱民. COM 原理与应用[M]. 北京：清华大学出版社，1999:175-192.

[5] Grundy J. Aspect-Oriented Requirements Engineering for Component-based Software Systems[C]//Proceedings of RE'99. Limerick, Ireland:[s.n.],1999.

(上接第 8 页)

的性能，将这些串行程序并行化，并进行性能优化成为一种有价值的应用研究。文中以手动并行化串行光线跟踪程序 PBRT 为例，介绍了在串行程序并行化中的一些关键问题，如并行模型的设计与实现，并行程序正确性验证，以及并行后程序的性能优化等问题。同时，还给出了这些问题相应的解决方案，为将其他串行程序并行化，并优化其性能提供了一些有益的经验。

参考文献：

[1] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach[M]. Third Edition. Beijing: China Machine Press, 2002.

[2] Kalla R, Sinharoy B, Tendler J. IBM POWER5 Chip: a Dual-Core Multithreaded Processor[J]. IEEE Micro, 2004, 24(2):40-47.

[3] Borkar S, Dubey P, Kahn K, et al. Platform 2015: Intel processor and platform evolution for the next decade[EB/OL]. 2005. http://www.intel.com/technology/magazine/computing/Platform-2015-0305.pdf.

[4] Yang C, Chen Y, Fu X, et al. A Comparison of Parallelization and Performance Optimizations for Two Ray-Tracing Applications[C]//In Proceedings of HPC&S'06. Bonn, Germany: IEEE CS Press, 2006:321-330.

[5] Jiang H, Lai C, Chen W, et al. Parallelization of Module Network Structure Learning and Performance Tuning on SMP[C]//In the 7th Workshop on Parallel and Distributed Scientific and Engineering(PDSEC'06). Rhodes Island, Greece: IEEE CS Press, 2006:25-29.

[6] Liu L, Hu W, Lai C, et al. Parallel module network learning on distributed memory multiprocessors[C]//In 2005 International Conference on Parallel Processing Workshops (ICPPW'05). Oslo, Norway: IEEE CS Press, 2005:129-134.

[7] Pharr M, Humphreys G. Physically Based Rendering: From Theory to Implementation[M]. San Francisco, CA, USA: Morgan Kaufman, 2004.

[8] Wald I, Benthin C, Dietrich A, et al. Interactive Distributed Ray Tracing on Commodity PC Clusters - State of the Art and Practical Applications[C]//In Proceedings of Euro-Par 2003, Lecture Notes on Computer Science 2790. Klagenfurt, Austria: Springer-Verlag, 2003:499-508.