第34卷 第10期 计 算 机 工 程 2008年5月
Vol.34 No.10 Computer Engineering May 2008
·软件技术与数据库· 文章编号：1000—3428(2008)10—0092—03 文献标识码：A 中图分类号：TP311.11

基于.NET平台的面向方面编程语言Eos

葛君伟，张 鹏，方义秋
(重庆邮电大学中韩合作GIS研究所，重庆 400065)

摘 要：目前面向方面编程(AOP)的主流工具是基于Java平台的AspectJ。该文论述了基于.NET平台的新AOP技术——Eos语言，该语言扩展了C#语言，引入一套完整的AOP语法，并提供Eos专用编译器，使其性能高于其他AOP实现技术。支持多语言的.NET平台的应用使跨语言应用AOP成为可能。

关键词：面向方面编程；C#语言；方面；.NET平台

.NET-based Aspect Oriented Programming Language——Eos

GE Jun-wei, ZHANG Peng, FANG Yi-qiu
(Sino-Korea Chongqing GIS Research Center, Chongqing University of Posts and Telecommunications, Chongqing 400065)

【Abstract】 At present the main technology of Aspect Oriented Programming(AOP) is AspectJ based on Java. This paper discusses a new AOP technology called Eos language, which is based on .NET. Eos extends the C# language, introduces an intact syntax of AOP, and supplies a special compiler; all of these improve the performance of Eos compared with the other AOP tools. The multi-language characteristic of .NET makes cross language use of AOP possible.

【Key words】 Aspect Oriented Programming(AOP); C# language; aspect; .NET platform

1 概述

面向方面编程(Aspect-Oriented Programming, AOP)提供了横切关注点的分离、捕获和实现的语言机制，较好解决了OOP和过程化编程方法不能解决的问题，如软件开发中的日志、安全和权限等横切关注问题。

Eos项目于2004年9月开始启动，2006年6月推出了最新版本0.3.4。Eos是一种基于.NET平台、用于C#语言的AOP工具。从语法定义角度上看，Eos在实现方式上近似基于Java平台的AspectJ。它引入了诸如join points, pointcuts, advice, introductions和aspects等关键词，并提供了专用的Eos编译器。Eos与AspectJ最大的不同是它支持方面实例化和实例级通知，即在Eos中可以用关键词new来实例化一个方面，同时这个实例可以有选择地通知对象[1]。

2 3种基于C#的AOP技术对比

目前AOP的主流实现方式可分为2种：(1)扩展或修改现有面向对象语言，直接在语言级实现AOP，比如Eos和ApsectJ；(2)在不扩展现有面向对象语言的基础上，利用代理技术或外部配置文件(如XML文件)来协助实现AOP，比如AspectC#和AOP.NET。在方面编织时，AspectC#用到了外部XML配置描述文件；AOP.NET用到了代理技术。

由于实现方式的不同，Eos, AspectC#和AOP.NET在方面定义、动态横切、静态横切支持、织入方式等方面也有不同，Eos和AspectC#既支持动态横切又支持静态横切，而AOP.NET仅支持动态横切。

2.1 Eos语言

由于Eos扩展了C#语言并把aspect也定义为引用类型，因此aspect可以像类一样包含数据成员(constants and fields)、函数成员(methods, properties, events, indexers, operators and constructors)和横切类型的成员(pointcuts, advices and introductions)[1]。在Eos中，aspect支持继承，aspects可以从抽象aspects或classes派生，但是一个class不能从一个aspect派生。aspect分为3种类型：Type-level, Instance-level以及Type-level和Instance-level的复合类型。为了区别这3种类型，Eos引入了方面修饰符intancelevel，对instance-level方面进行声明。Eos的动态横切与AspectJ相似，需要用切点和连接点来实现，且可以利用操作符和逻辑运算符(&&, ||, !)来组合多个切点。但在Eos中通配符需用any来代替*，以避免和C#中的指针*冲突。静态横切须用引用(introductions)来实现。在用Eos编织器进行编织时，输入的是Eos源代码，输出的是代码文件对象模型(Code Document Object Model, CodeDOM)。

2.2 AOP.NET

由于AOP.NET没有扩展C#语言，因此在AOP.NET中，用一个普通C#语言的类来表示aspect，而通知则用aspect类中的方法来表示。横切声明由传统的属性标志advice来实现，有效区分了advice和类中的方法，并向编织器提供了切点(pointcuts)信息，以保证相应advice能正确应用到pointcuts定义的joint points中。在AOP.NET中，输入编织器的是组件和aspect，编织器在不修改其IL(Intermediate Language)代码的情况下，将其编织成一个代理对象来表示aspect或原来的组件[2]。

2.3 AspectC#

在AspectC#中，aspect用一个C#源文件来实现。动态横切用一个XML文件来详细说明，这个XML文件称为Aspect配置描述符；静态横切使用属性[Introduction(<string>)]来表示。

基金项目：重庆市自然科学基金资助项目(2005BB2059)
作者简介：葛君伟(1961－)，男，教授、博士，主研方向：软件工程；张 鹏，硕士研究生；方义秋，副教授
收稿日期：2008-02-25 **E-mail：**xiaozhang206@yahoo.com.cn

本文中String表示将要被引入的新变量和方法的类。AspectC#编织器把基本类源代码、aspect源代码和aspect配置描述符作为输入，在不改变基本类代码和aspect代码的情况下，生成aspect配件作为输出，详细结构如图 1 所示[3]。

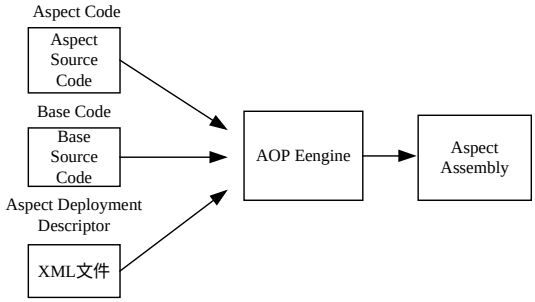

图 1　AspectC#编织结构

3　Eos 的编译结构

图 2 是 Eos 专用编译器的组成结构。在进行编译时，一个标记识别器(tokenizer)会先抽取 Eos 源代码中的标记符号；再把 Eos 源码交给剖析器(parser)进行分析处理，得到 C#基类源代码和 aspect 源代码，由各自的模块生成器输出 CodeDOM 形式的抽象语法树(Abstact Syntax Tree, AST)代码；然后编织器会把基类代码和 aspect 代码合并在一起输入代码生成器生成最终的源代码；最后由代码编译器将其编译成可在.NET 平台下直接执行的配件。

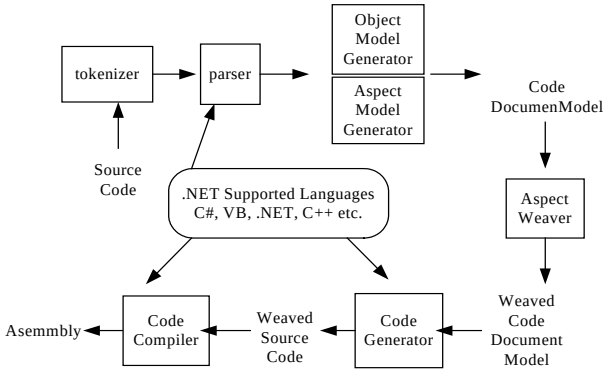

图 2　Eos 的架构和工作流程

3.1　标记识别器、剖析器和模型生成器

标记识别器主要用于识别并提取Eos源代码中的标记符号，以便于下一步剖析器的分析和处理。剖析器分析并处理Eos源代码的语句结构后，将C#的基类代码和aspect代码分离并交由模型生成器生成各自类型的AST。CodeDOM提供了一些接口、类和结构以对象图或树的形式来表示源代码文档，这些对象图和树是独立于语言的。不同语言编写的程序都可以建模成AST，而编织工作又可以在对象图层进行，因此，实现跨语言的编织，达到跨语言实现AOP应用的目标是可行的[4]。但由于CodeDOM目前不完全支持通用语言规范，因此要实现这个目标的还需较多工作。

3.2　方面编织器

方面编织器以 AST 为输入，在把方面织入 AST 后，输出 CodeDOM 形式的代码文件。在 Eos 中有 3 种类型的 aspect，各种 aspect 编织方式和编织后的编译流程是有区别的。

Type-level 类型的方面不能被实例化，且在方面中没有任何编织修饰符；Instance-level 类型的方面可以被实例化，以参数形式传递并获取返回值，并且有关键词 Intancelevel 标记；复合类型的方面中既包含 Type-level 型的 advice 也包含有 Instance-level 类型的 advice。

下面是这 3 种aspect及其advices的示例代码[1]：

```
(1)public aspect A {/* Some Members */} ///Type-level aspect in Eos
(2)public instancelevel aspect B{ //Instance-level aspect in Eos
(3)pointcut callfoo():call(public void any.foo());  }
(4)public aspect C { // Aspect in Eos containing both type and instance –level advices
(5)pointcut fval(): fget(public int any.val);
(6)instancelevel after():execution(public void any.bar());
(7)after():fval(){// Do something}  }
```

在第(5)条代码中出现的 fget 相当于 AspectJ 中的关键词 get。为了避免与 C#中的关键词 get 和 set 冲突，Eos 中引入了关键词 fget 和 fset。

由于 aspect A 中没有任何编织修饰符，因此其中的 advice 直接在类级别静态织入。例如 aspect A 的 advice 可以织入一个类 P，进而影响类 P 的所有实例。关键词 Instancelevel 则把 Aspect B 的 advice 的编织推迟到了运行时，编译时编织器保存符合 pointcut 条件的 join points 的存根，以实现 Aspect B 动态织入。在 Eos 中允许方面中一部分通知静态织入而另一部分通知动态织入，例如在 aspect C 中，无编织修饰符的通知可以直接在类级别静态织入，而有编织修饰符的通知则可以在类型级别动态织入。这也是 Eos 相比于其他 AOP 工具的优点。

3.3　方面映射

Join points是AOP工具的核心，它定义了AOP工具控制横切的粒度。Eos在实例级构造Join points时静态决定其中哪些被通知，并利用关键词events来引入这些join points，而不是像AsepectJ那样通过关键词call来实现，在Eos中events支持动态的匹配和声明。下面的类Model的编织代码可以说明events的作用[1]：

```
(1)/*Generated by Eos version 0.1 * /
(2)public class Model {
(3)bool value ;
(4)public Model( ) { value = false ; }
(5)public void Set( ) {
(6)eos.Joinpoint thisJoinPoint =new
(7)eos.Joinpoint(null,this,null,newSystem.Object[ ]{ });
(8)try { value = true; } finally {
(9)if( ADP _ EOS _ After_ EXECUTION _ Set!=null)
(10)ADP _ EOS _After_ EXECUTION _ Set
      (thisJoinPoint) ;  }  }
(11)public bool Get( ) { return value ;}
(12)public void Clear( ) {
…     // Similar to Set
(19)}
(20)public event eos.ADP
ADP_EOS_After_EXECUTION_Set;
(21)public event eos.ADP
        ADP_EOS_After_EXECUTION_Clear;  }
```

在上面的代码中有 2 个join points被通知：after Set和after Clear。2 个events被引入并在Set和Clear函数体后声明：ADP_EOS_After_EXECUTION_Set 和 ADP_EOS_After_EXECUTION_Clear。而方面Consittency中的隐式函数addOject 和removeObject将利用这个events匹配advice以完成实例级的编织。方面Consistency的生成代码如下[1]：

```
(1)/* Generated by Eos version 0.1*/
(2)public class Consistency{
(3)Model m1, m2;
(4)bool busy;
(5)public Consistency(Model m1, Model m2){
(6)addObject(m1); addObject(m2);
(7)this.m1=m1; this.m2=m2;
(8)busy=false; }
(9)public void EOS_Advice_After0
(eos.Joinpoint thisJoinPoint){
(10)if(!busy){
(11)busy=true;
(12)Model m=(Model) thisJoinPoint.getTarget( );
(13)if(m==m1) m2.Set(); else m1.Set( );
(14)busy=false;  }  }
(15)public void EOS_Advice_After1
(eos.JoinPoint thisJoinPoint)
…// Similar to the first advice
(22)}
(23)public void addObject(object obj){
(24)if(obj==null) return;
(25)if(obj is Model) {
(26)Model cased_obj=((Model)(obj));
(27)casted_obj.ADPart_EOS_Afte_EXECUTION
(28)_Set+=new eos.ADP(EOS_Advice_After0);
(29)casted_obj.ADP_EOS_After_EXECUTION
(30)_Clear+=new eos.ADP(EOS_Advice_After1);  }  }
(31)public void removeObject(object obj){
(32)if(obj==null)return;
(33)if(obj is Model){
(34)Model casted_obj=((Model)(obj));
(35)casted_obj.ADP_EOS_After_EXECUTION
(36)_Set-=new eos.ADP(EOS_Advice_After0);
(37)casted_obj.ADP_EOS_After_EXECUTION
(38)_Clear-=new eos.ADP(EOS_Advice_After1);  }  }  }
```

3.4 代码生成器和编译器

代码生成器的主要功能是把编织后的 CodeDOM 文件转化成编译器能够识别的源代码，但并不是所有 CodeDOM 文件都要转换成源代码。除非 CodeDOM 文件声明需要源代码转换，否则 CodeDOM 文件由 Eos 编译器直接编译成在.NET 平台下执行的配件。Eos 编译器既可以编译 CodeDOM 文件也可以编译代码生成器输出的源代码。

4 结束语

Eos 是一种新的 AOP 技术，具有很多优点，例如支持方面实例化和多种方式的织入机制、实现了与 C#语言的无缝对接等，极大推动了 AOP 技术的发展。

参考文献

[1] Rajan H, Sullivan K. Eos: Instance-level Aspects for Integrated System Design[C]//Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering Helsinki, Finland: [s. n.], 2003.

[2] Michael A. Blackstock M. Aspect Weaving with C# and .NET [EB/OL]. (2004-01-25). http//www.cs.ubc.ca/~michael/publications/AOPNET5.pdf.

[3] Kim H. AspectC#: An AOSD Impleme-ntation for C#[D]. Dublin, Ireland: Department of Computer Science, Trinity College, 2003, 52-57.

[4] 曾　路, 张立臣. AsoectC#——在 C#中应用 AOP[J]. 计算机应用研究, 2005, 22(5): 189-190.

(上接第 91 页)

endopen 将设备的 open 结果转为系统内核希望的方式返回给系统。这样屏蔽了系统信息对开发者的干扰，简化了开发工作。该方式实质上参照了设计模式中的代理模式。另外，框架为所有的操作提供了默认操作。即使开发者因为某种原因没有为设备定义某个操作，用户也不会因为执行了该操作导致内核访问出错。

4 框架下的设备驱动程序开发

有了框架作支撑，开发者不仅可以快速地开发 Linux 设备驱动程序，而且简化了对设备驱动程序的认识。

从开发者角度看，一个驱动程序由一个 KDriver 的子类和 KDevice 的子类构成。开发者为完成驱动程序所需的工作就是在 MyDevice 类中将继承自父类的虚函数重写，增加和设备操作实际相关的代码，如图 6 所示。

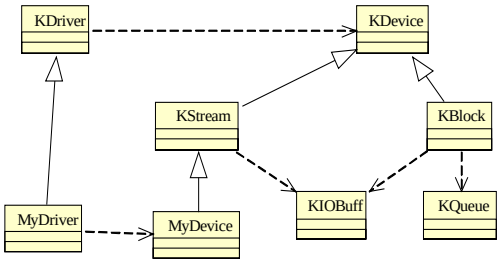

图 6　实际开发时驱动程序的构成

此时，开发者不再关心系统相关的细节，他们可以直入主题，考虑具体功能的实现。同时还可利用框架类库中提供的各种算法和容器进一步减轻开发工作量，但并不以程序质量下降为代价。

5 结束语

从程序开发的角度看，驱动程序和应用程序并没有本质的区别，但在工程化方面上却有较大差异，尤其是 Linux 系统。本文将 C++应用到 Linux 内核环境，并以此为基础设计了一个驱动程序开发框架，能帮助开发者规范 Linux 设备驱动的设计流程，降低设计难度。

参考文献

[1] Driverworks Help Documents[Z]. (2007-03-12). http://www.compuware.com/products/driverstudio/.

[2] Rubini A. Linux Device Drivers[M]. 2nd ed. [S. l.]: O'Reilly & Associates, 2001.

[3] Source File of Insmod[Z]. (2006-12-29). http:// www.koders.com.

[4] Bovet D P. Understanding the Linux Kernel[M]. 2nd ed. [S. l.]: O'Reilly & Associates, 2003.

[5] Lippman S B. Inside The C++ Object Model[M]. [S. l.]: Addison Wesley Longman, 1996.

[6] Ivar J, Grady B, James R. The Unified Software Development Process[M]. [S. l.]: Addison Wesley Longman, 1999.