文章编号:1674-2869(2020)04-0456-06

基于AOP的契约定义及其与JML契约的转换

张 进,何成万*,石 尤
武汉工程大学计算机科学与工程学院,湖北 武汉 430205

摘 要:为了解决在使用基于Java建模语言(JML)契约的过程中存在维护困难、契约不能重用等问题,提出了一种JML契约与基于面向方面编程(AOP)的契约转换方法。首先给出了基于AOP的契约定义方法,并在分析JML和AOP语言的语法规则基础上,提出了AOP契约和JML契约之间的转换策略,为进一步实现基于AOP的契约到JML契约的自动转换奠定了基础。通过JML契约到AOP契约的转换,可以方便地实现契约的维护,而通过AOP契约到JML契约的转换,可以实现契约的重用以及自动检查。

关键词:重用;面向方面编程;JML契约;契约转换
中图分类号:TP311.5 **文献标识码**:A **DOI**:10.19843/j.cnki.CN42-1779/TQ.201912025

AOP Contracts Definition and Its Conversion to JML Contracts

ZHANG Jin, *HE Chengwan*, *SHI You*
School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan 430205, China

Abstract: To solve the maintenance and reusability problems of Java Modeling Language (JML) contracts, we proposed a conversion method between JML contracts and aspect-oriented programming (AOP) contracts. First, the method of specific definition for AOP-based contracts was presented, and then a conversion strategy between AOP contracts and JML contracts was proposed by analyzing the syntax rules of both languages of JML and AOP, which lays a foundation for further improving the automatic conversion from AOP-based contracts to JML contracts. By the conversion from JML contracts to AOP contracts, the maintenance of contracts can be easily achieved, and the reusability and automatic checking of contracts can be implemented through the conversion from AOP contracts to JML contracts.

Keywords: reusability; aspect-oriented programming; JML contract; contract conversion

契约式设计(design by contract,DbC),又叫合同设计方法,可以用于确保程序的准确性和提高程序的可靠性,它是基于Hoare的公理化证明方法而开发的设计思想[1]。基于合同的开发主要是通过在程序模块中添加前提条件、后置条件和不变量来为模块设计契约[2]。前提条件描述一个模块对调用该模块人员提出的要求,调用此模块之前必须满足该模块的前提条件。后置条件描述的是调用者在调用某模块之后应该满足的条件。不变量是对于整个模块而言,不变量必须在模块执行前后被满足。在面向对象的设计中使用合同编程的思想,形成了各种支持契约式的语言,如Eiffel、Larch、Java建模语言(java modeling language,JML)、还有Microsoft.Net平台上的SPEC#等。其中,埃菲尔语言是最著名的规范语言[3]。

面向方面编程(aspect-oriented programming,AOP)通过封装横切关注点的思想为解决合同编程中所出现的代码交织等问题提供了很好的解决

收稿日期:2019-12-31
作者简介:张 进,硕士研究生。E-mail:1130027927@qq.com
***通讯作者**:何成万,博士,教授。E-mail:hechengwan@hotmail.com
引文格式:张进,何成万,石尤. 基于AOP的契约定义及其与JML契约的转换[J]. 武汉工程大学学报,2020,42(4):456-461.

方案,并且,AOP的语言结构与契约之间也有着自然的对应关系[4],当不变式以及前置条件和后置条件均看成横切关注点,AOP就可以很容易地完成它们与其他功能模块之间的相互分离,使得开发人员可以在独立的编织器模块中实现和编写这些模块的功能,并以较为灵活的方式调用它们[5]。AOP的这些特性正好可以弥补合同编程的缺陷,所以,在面向对象设计中使用AOP技术来支持合同设计是一个增强软件系统可靠性的新方法[6-7]。

但是,作为一种形式化的方法,基于AOP的契约很难执行。此外,若用手动方法将基于AOP的契约转换为可执行程序会导致性能下降和难以维护等问题。

JML作为Java的注释语言同样允许规范,而且相关工具也很成熟。因此,本文分别讨论了两种语言,提出了一种基于AOP的契约定义的方法,并且给出了实例,本文还提出了一种将高抽象层次的基于AOP的契约规范翻译为低抽象层次的JML规范的策略。目标是使用现有的基于JML的工具来验证基于AOP的契约代码。

1　国内外相关工作的比较分析

对于面向方面编程和JML契约有很多研究者进行了研究,Rebêlo[8]曾在一篇论文中提过一种使用AspectJ来实现JML的方法,他提到使用AspectJ来实现一个新的JML编译器,它生成一个符合检测的字节码,同时使用Java SE和Java ME应用程序。并进行了比较研究来证明该编译器生成的最终代码的质量,将代码的大小与现有JML编译器生成的代码进行比较。此外,还评估了在Java应用程序中实现JML断言所需的额外代码量。结果表明,该编译器生成的代码大小的开销非常小,这对Java ME应用程序至关重要。而本文的工作是用AOP来定义契约合同,然后根据定义的基于AOP的契约与JML契约之间的映射关系来介绍一种进行基于AOP的契约与JML契约相互转换的模型策略。

Smith[9]介绍了一种将横切特征表示为逻辑不变量,然后生成由手动编写的方面得到的代码的方法,并举了各种例子说明了不变方法。该方法的关键思想是利用不变量捕获方面的意图。方面编织是通过在基本代码的适当位置生成和插入代码片段来保持不变的过程。在这种方法中,切入点规范是从不变量派生出来的,它描述了可能破坏不变量的一组代码点。对于代码中的每一个这样的中断点,维护代码的规范是从不变式中派生出来的。与本文研究不同的是本文直接用AOP语言来描述契约合同,而不是进行方面编织。

Carr[10]提出了一种通过自动代码将静态分析结果应用于现有代码的新工具(code contracts bot,CCBot),CCBot使用方法前置条件,后置条件和对象不变量来检测代码,这些不变量在运行时或静态使用静态合同检查器检测故障。程序员需要执行的唯一配置是为CCBot提供她想要检测的代码的文件路径。这使程序员可以轻松地采用基于契约的静态分析。如果原始代码确实如此,CCBot的代码检测版本可以保证编译。此保证意味着程序员可以立即部署或测试已检测的代码,无需额外的手动操作。插入的契约可以检测常见错误,例如空指针解引用和超出范围的数组访问。本文研究的是将基于AOP契约转换为JML契约,然后通过jmldoc来生成带有JML注释的文档,从而实现将JML注释自动插入到Java代码中。

徐倩颖[11]提出了面向方面编程领域的一种新型设计模式:方面桥模式,该模式解决了构件与行为模式间的耦合问题,体现了“高层分离,低层耦合”的原则。在该模式中,行为模式是通过抽象化方面来表现的,主要是对接口进行模块化。而本文的研究是对契约进行模块化,利用AOP语言来定义基于AOP的契约,从而实现契约的重用。

2　基于AOP的契约定义

2.1　AOP-DbC的对应关系

契约的基本内容包括:

前置谓词:描述一个模块对调用该模块的人员提出的要求,调用此模块之前必须满足该模块的前提条件。

合法的输入:输入的格式和取值是合法的。

期望的输出:返回给用户的结果。

后置谓词:不管某类操作的执行过程如何,在其终结之后必须满足后置条件

不变式:在执行某个操作之后变量保持不变。

图1所展示的是契约式开发的一般逻辑。

DbC与AOP机制提供的语言结构,诸如连接点、切入点以及通知等有着直接的对应关系,使用before通知检查前置条件,必须在执行某个切入点确定的方法之前被检查,而使用after通知检查后置条件,必须在方法执行之后进行检查。同时使用before和after通知或者使用around通知检查不变式,必须在方法执行前后进行检查。当前的研

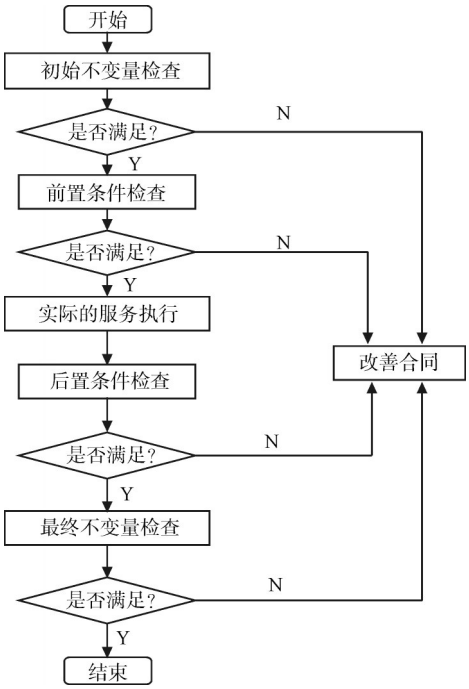

图1 契约式开发的流程图

Fig. 1 Flowchart of contract development

究领域不是偏重于下层的编码细节,就是偏重于并不是为AOP专门设计的对象约束语言(object constraint language,OCL)等契约语言,所以,目前并没有给出一个绝对的AOP-DbC的对应关系。本文图2给出了AOP和DbC之间的一个对应关系。

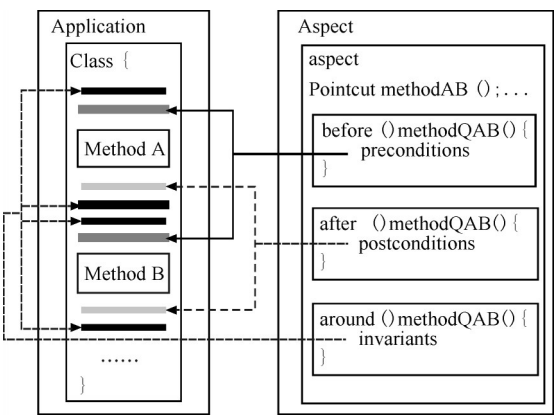

图2 AOP与DbC的对应关系

Fig. 2 Corresponding relationship between AOP and DbC

将非功能代码(即非断言所需的代码)与业务逻辑代码两者混合起来,这种使用断言机制支持契约的一种方法,会造成代码纠缠。如果目的是为了分离关注点,从而达到灵活而透明的特点,那么面向方面编程则是最佳选择[12]。前置谓词、后置谓词和不变式常出现于应用程序的各个模块中,在某种程度上与应用程序代码相混合,可以看作是横切关注点的表现。AOP的目标就是让开发人员可以在独立的模块中单独编写这些功能,然后自然地应用它们。

AOP可以将横切关注点模块化[13],而且DbC的实现也是横切的。JML无法编写单个前置和后置条件,并将其应用于特定类型的多种方法,只能在每个方法中单独添加,而这种前置或后置条件通常必须在几种方法中重复和分散。

另一方面,目前存在的用于静态分析的工具通常都需要很多程序员工作[14]。在大型代码库中,静态分析工具会产生数千个警告,期望用户查看如此庞大的列表并手动对每个警告进行更改是不现实的[15]。减少警告数量的一种方法是添加注释,通常,注释可能会禁用某些消息,指定程序的预期行为,或启用/禁用某些代码的分析。这些手动注释在小例子中是可管理的,但是从完全未注释的代码库开始时则不适用于大型代码库。自动插入合同而不是显示警告列表,有两个可用性优势,可以减少程序员所需的工作量。1)程序员可以使用熟悉的工具查看上下文的合同。2)接受或测试其代码的修改版本,无需任何手动步骤。

2.2 前置条件的AOP定义

用before通知与方法的前置谓词相对应,图3是前置谓词的定义。

```
public aspect Precondition {
  pointcut m():call(void c.m())&&args();
  before():m(){
  if(!α){
    throw new InternalPreconditionError();
    }
  }
}
```

图3 前置条件的定义

Fig. 3 Definition of preconditions

在图3中,pointcut表示方法的切入点,用来指定所选连接点中的关注,args表示方法调用的参数,如果不满足前置条件,程序抛出异常。

2.3 后置条件的AOP定义

用after通知与方法的后置谓词相对应,图4是后置谓词的定义。

```
public aspect Postcondition{
  pointcut m():call(void c.m()) && target()&& args();
  after():m(){
  if(!?){
  throw newInternalNormalPostconditionError();
    }
  }
}
```

图4 后置条件的定义

Fig. 4 Definition of postconditions

在图4中,pointcut表示方法的切入点,用来指定所选连接点中的关注,target表示目标对象,如果不满足后置条件,抛出异常。

2.4　不变量的AOP定义

采用around通知对应不变量,图5是不变量的定义。

在图5中,pointcut表示方法的切入点,用来指定所选连接点中的关注,target表示目标对象,args表示方法调用的参数,如果不满足不变量,程序便会抛出异常。

```
public aspect Invariant {
   pointcut (c): call (void c*(..)) && target ()
   around ():c.*(..){
   if (!ß) {
   throw new InvariantError ();
    }
   }
 }
```

图5　不变量的定义

Fig. 5　Definition of invariants

2.5　实　例

AOP有着将横切关注点模块化的特点,而契约的约束条件如前置条件、后置条件、不变量约束可以看作横切关注点,因此AOP可以将这些约束条件封装成方面,与业务逻辑代码分离。然后使用连接点、切入点来指定即将检查的契约代码的具体位置,并在相应的before或者after以及around通知中定义契约的内容。以边界判断为例,图6是用AOP的方式实现边界判断的契约检查。

从图6可以看出,在PointBoundsPreCondition方面中,声明了一个before通知,用于修改类Point的set*X*方法的行为。before通知可以应用于每个连接点,args关键字表示方法调用的参数。在PointBoundsPostCondition方面中,声明了一个normal after通知,该通知修改类Point的set*X*方法的行为。after通知可以应用于每个连接点,target和args关键字表示目标对象和方法调用的参数。在PointBoundsInvariant方面中,它声明了一个around通知来修改类Point的set*X*方法的行为。在每个连接点之前和之后检查不变量,target和args关键字表示目标对象和方法调用的参数。如果x大于MIN_*X*,小于MAX_*X*,则执行语句proceed(p, x);另一方面,在else子句中,通知直接调用set*X*方法。一般情况下,一旦契约不能被满足,程序就会抛出异常或直接退出程序。

```
 aspect PointBoundsPreCondition {
     before ( int x ): call ( void Point . set X ( int )) && arg ( x ) {
       if ( x < MIN  _ X || x > MAX  _ X ){
         throw new RuntimeException () ;
     }
   }
}
 aspect PointBoundsPostCondition {
   after  ( Point p  , int x ): call  ( void Point . setX ( int )) &&  target  ( p )&&  args ( x ){
     if ( p . getX  ()!=  x )
       throw new RuntimeException() ;
     }
   }
}
   aspect PointBoundsInvariant  {
    around  ( Point p  , int x ): call  ( void Point . setX  ( int )) &&  target  ( p )&&  args  ( x ){
     if  ( x > = MIN  _ X &&  x <= MAX _ X ){
      proceed  ( p , x );
    }  else if  ( x < MIN  _ x ) {
        p . setX  ( MIN  _ X );
    }  else {
        p. setX ( MAX  _ X );
    }
 }
```

图6　边界判断的基于AOP的契约检查

Fig. 6　AOP-based contract checking for boundary determination

3　AOP契约到JML契约的转换方法

3.1　JML语法

JML是一种通用规范建模语言,是面向Java的行为接口规范语言(behavior interface specification language,BISL)。它主要用于描述基于Java的调用模型的使用行为和详细的模型设计。

JML规范是基于埃菲尔的DbC的思想。调用方有义务确保方法的先决条件是正确的,如果先决条件为false,则调用方与其被调用方之间的契

约将不再有效。JML遵循BISL的形式化语义，继承DbC的可执行能力。

除了DbC中定义的前置条件和后置条件以及不变量外，JML还有许多其他类型的约束，这些约束增强了JML的验证和调试机制。图7显示了含有JML注释的具体Java代码，JML注释通过/*@和@*/来表示。

```
/* @ public normal _behavior
   @ assignable frame -condition (s);
   @ requires pre -condition (s);
   @ ensure
   @ post -condition (s);
   @ also
   @ public exceptional _behavior
   @ requires exceptional condition ;
   @ signals  (Exception e ) R;
   @*/
public  /*@ pure @*/ Object throws Exception ;
```

图7 JML注释的模板

Fig. 7 Template for JML annotations

目前，许多JML工具已经成熟，如运行时断言、jmlUnit和JMLAutoTest。使用这些工具，JML规范可以转换为可执行代码，然后利用运行时断言进行检查。除此之外，不仅能利用JML工具自动生成测试框架，还可以自动生成测试用例。

3.2 转换策略

基于AOP的契约定义将前置条件、后置条件以及不变量封装，有助于重用，但也有一定的缺陷性。而将基于AOP的契约定义转换为JML契约，自动生成注释加入到每个方法之内，可以减少程序员的工作量，本文提出了一种两者之间转换的模型策略，如图8所示。

从图8中可以看出，本文是将基于AOP的契约与JML契约的转换实现为一个插件工具（虚线框部分），该插件利用Java的反射机制来调用翻译器进行两种契约之间的转换，而插件的用户界面是用Window Builder和Java Swing实现的。

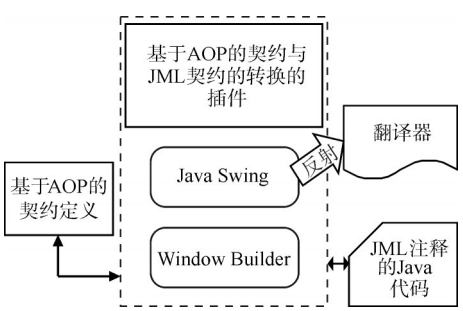

图8 基于AOP的契约和JML契约转换的模型

Fig. 8 Model of conversion between AOP-based contract and JML contract

3.3 从基于AOP的契约类型到JML类型的映射

3.3.1 基本数据类型 基于AOP的契约与JML契约之间最基本的映射是数据类型之间的映射。由于JML是针对Java定制的，而AOP与Java的实现相同。因此，在基本数据类型上，AOP与JML有着直接的对应关系，两种语言都主要包括int（整型）、double（双精度浮点型）、boolean（布尔型）、char（字符串型）等基本数据类型。

3.3.2 运算符 对于比较运算符，AOP和JML同样对应的都是Java类型，主要包括：==、！=、<、>、<=、>=等。但是二者在逻辑运算符上稍有不同，AOP和JML两种语言在逻辑运算符上的映射关系如表1所示。

表1 逻辑运算符

Tab. 1 Logical operators

AOP Type	JML Type
$!a$	$!a$
$a\&\&b$	$a\&\&b$
$a\|\|b$	$a\|\|b$
$(!a)\|\|b$	$a==>b$
$a==b$	$a<==>b$

3.3.3 集合类型 Java最核心的3种集合类型分别是set（集）、list（列表）、map（映射）。JML定义了对应的模型种类来表示各种不同的Java类的集合，分为对象集合和值集合，都实现了JMLCollection接口。对象集合中元素是对象的引用，并不关心对象的值。对象集合分为JMLObjectSet，JMLObjectBag，JMLObjectSequence 3种。值集合分为JMLValueSet，JMLValueBag，JMLValueSeque-nce 3种，值集合中的元素存储的是对象的值。集合类型的对应关系如表2所示。

表2 集合类型的映射

Tab. 2 Mapping of collection types

AOP Type	JML Type	
Set	JMLValueSet	JMLObjectSet
List	JMLValueSequence	JMLObjectSequence
Map	JMLEqualsToEqualsMap	JMLObjectToObjectMap

3.3.4 约束规范的映射 基于AOP的契约与JML契约在前置条件和后置条件以及不变量约束上均存在着一定的映射关系，使得2种契约之间的转换成为可能，2种契约的元素之间的对应关系如表3所示。

表3　约束规范的元素之间的映射

Tab. 3　Mapping between elements of constraint specification

AOP契约	JML契约
AOPClassName	JMLClassName
AOPMethod	JMLMethod
aspect	null
pointcut	null
before()	requires
after()	ensures
around()	invariants
AOPexpression	JMLexpression

4　结　论

基于AOP的契约可以将方法的前置条件、后置条件以及类不变式封装,实现契约的模块化处理,解决代码纠缠问题。它与JML相似,但是各有优缺点,JML具有更多的验证特性。

本文分析了两种不同抽象层次的规范语言:基于AOP的契约和JML契约。提出了一种基于AOP的契约与JML契约之间的转换策略,并给出了基于AOP的契约与JML契约转换之间的映射关系,使约束规范能够更具体地指导后续的软件开发。作为未来的工作,希望构建一个框架来支持基于AOP的契约到JML契约的自动化转换。

参考文献

[1] RUBIO-MEDRANO C E, AHN G J, SOHR K.Verifying access control properties with design bycontract: framework and lessons learned [C]// IEEE Computer Society. Kyoto:IEEE,2013:21-26.

[2] REBÊLO H, LEAVENS G T, BAGHERZADEM, et al. AspectJML: modular specification and runtime checking for crosscutting contracts [C]//International Conference on Modularity. New York: ACM, 2014: 157-168.

[3] REBÊLO H, LIMA R, LEAVENS G T, et al. Optimizing generated aspect-oriented assertion checking code for JML using program transformations: An empirical study [J]. Science of Computer Programming,2013,78(8):1137- 1156.

[4] 徐承志,张国玉.基于角色划分的AOP设计模式研究[J].自动化技术与应用,2019(8):59-64.

[5] 张秀峰.AOP技术及其在软件安全中的应用[D].北京:北京邮电大学,2008.

[6] 江华丽.基于AOP策略模式的实现机制[J].微型机与应用,2016,35(1):9-11.

[7] 张峻豪,陈媛,王俊杰,等.基于AOP的面向对象程序的单元测试的应用[J].电子技术与软件工程,2017(9):51-57.

[8] REBÊLO H, SOARES S, LIMA R. et al. Implementing Java modeling language contracts with AspectJ [C]// Symposium on Applied Computing. New York: ACM, 2008:228-233.

[9] SMITH D R. Aspects as invariants [M]. Berlin: Springer Netherlands, 2008:247-263.

[10] CARR S A, LOGOZZO F, PAYER M. Automatic Contract Insertion with CCBot[J]. IEEE Transactions on Software Engineering, 2017, 43(8):701-714.

[11] 徐倩颖,杨宗源.面向方面编程的一种新型设计模式[J].华东师范大学学报(自然科学版),2008(1):68-74.

[12] HANNOUSSE A. Dealing with crosscutting and dynamic features in component software using aspect-orientation: requirements and experiences [J]. IET Software,2019,13(5):434-446.

[13] 何成万,叶志鹏.基于AOP和动态污点分析的SQL注入行为检测方法[J].电子学报,2019,47(11):2413-2419.

[14] VASSALLO C, PANICHELLA S, PALOMBA F, et al. How developers engage with static analysis tools in different contexts[J]. Empirical Software Engineering, 2019,25(12):1419-1457.

[15] 崔少轩,喻垚慎.静态程序分析过程中形式化验证工具Frama-C的应用[J].计算技术与自动化,2019,38(1):114-117.

本文编辑:陈小平