文章编号:1000-5862(2020)06-0633-06

新型 Action 泛型机制在 PAR 平台中的实现和应用研究

汪 雄[1,2],薛锦云[2*],刘震伟[1,2]
(1. 江西师范大学计算机信息工程学院,江西 南昌 330022;2. 江西师范大学国家网络化支撑软件国际科技合作基地,江西 南昌 330022)

摘要: 在软件开发过程中,开发语言中的泛型机制能够显著提高软件开发的效率和软件可靠性,但现有高级语言中一般只有类型和子程序(如过程、函数和方法等)作参数,限制了泛型机制的应用. 该文对新型泛型机制进行深入探索,着重研究以 action 作为参数的泛型机制. 在 Apla→Java 程序自动生成系统中实现了将 Web 服务作为 Apla 语言中的 action 参数的泛型机制,完善了 Apla 语言中的泛型安全的约束机制,成功地将新型泛型机制应用于 PAR 平台中.

关键词: Apla 语言; 泛型; PAR 方法; PAR 平台
中图分类号: TP 311 **文献标志码:** A **DOI:** 10.16357/j.cnki.issn1000-5862.2020.06.15

0 引言

随着经济的快速发展,人们对计算机软件的需求也愈来愈多样化. 如何提高软件开发的效率,成为计算机学界内的研究热点. 而泛型程序设计和服务组合是解决这一问题的有效途径[1].

有较多支持泛型程序设计的高级程序设计语言[2-6],如 ML[7]、Java[8]、Eiffel[9]、Haskell[10] 和 Ada 等. 但这些语言的泛型机制大部分只支持简单的类型作为参数,只有 Ada 语言支持子程序作为参数[11]. 目前,高级程序设计语言的泛型机制较为简单,难以满足泛型程序设计纷繁复杂的需求. 针对这一现实问题,本文首先对笔者于 2015 年在国际形式化研讨会上提出的新型泛型机制进行了更深入的研究,并全面地在 Apla 语言中实现了笔者提出的新型泛型机制,完善 Apla 语言中的泛型安全约束机制;添加泛型约束变量,解决了原来 Apla 语言中泛型约束不明确的问题[12-13].

由于单个 Web 服务的功能有限,只有将服务进行组合,实现高效的复合服务,才能满足现在用户纷繁复杂的需求[14-15]. 为此,本文设计并实现了将 Web 服务构件作为 action 参数的泛型机制.

1 新型泛型机制

2015 年在法国巴黎召开的国际形式化研讨会中笔者提出:现有泛型机制难以满足现在软件产业多样化的需求,在高级程序设计语言中泛型机制,不应只有以类型作为参数,更应该对泛型机制进行扩展,将 action(包括函数、过程、服务和子系统等)和 ADT(抽象数据类型)作为泛型参数.

1.1 action 作为泛型参数

目前,在所有的高级程序设计语言中,只有 Ada 语言支持类似 action 作为泛型参数的泛型机制,Ada 语言支持将子程序作为泛型参数,但其功能有限. 在笔者提出的新型泛型机制中,action 不仅可以是函数或过程,还可以是 Web 服务、子系统等. 本文的主要工作,就是将 action 作为泛型参数在 Apla 语言中完整实现,完善 Apla 语言的泛型机制.

在算法与软件开发的过程中,经常会遇到需要将一个操作或动作当作参数的情况. 比如实现一个求数组最长降序段的算法后,再实现一个求数组最

收稿日期:2018-05-05
基金项目:国家自然科学基金重大国际合作项目(61020106009),国家自然科学基金面上项目(61272075,61472167),国家自然科学基金地区科学基金(61462041),江西省自然科学基金(20171BAB202008)和江西省教育厅科学技术研究(160329)资助项目.
通信作者:薛锦云(1947-),男,江苏海门人,教授,博士生导师,主要从事软件形式化和自动化的研究. E-mail: jinyun@vip.sina.com

长升序的算法,若能将降序和升序这 2 个操作以参数形式传入,则能大大降低编码长度,提高编程效率. 使用 Apla 语言表示如下:

```
function sort(someaction operation);
var A: array [0..n-1, integer];
    i, z, b, c: integer;
begin
    foreach(i: 0≤i<n: read(A[i]));
    i:=1; z:=0; b:=0; c:=1;
    do i≠n→
        if A[i-1]?A[i]→c:=c+1;
        ▯→c:=1;
        fi;
        z:=max(z, c); i:=i+1;
    od;
end;
someaction sort={≤, ≥, =}.
```

上述 Apla 程序,定义了一个泛型方法,其泛型参数为升序、降序和等于 3 个 action,根据实例化传入的 action 参数可以实现求最长升序、降序和最长平台数 3 个算法. action 的取值可以是预先定义的操作、自定义的函数、过程或服务,并使用 someaction 对操作的取值进行限定.

1.2 ADT 作为泛型参数

ADT(抽象数据类型)由数据和操作组成,当实现上述求数组最长降序算法时,数据类型和 action 可能会发生变化. 用 Apla 语言表示如下:

```
function sort(someadt adt);
var A: array [0..n-1, adt];
    z, b, c: integer;
    i: adt;
begin
    foreach(i: 0≤i<n: read(A[i]));
    i:=1; z:=0; b:=0; c:=1;
    do i≠n→
        if A[i-1]?A[i]→c:=c+1;
        ▯→c:=1;
        fi;
        z:=max(z, c);
        i:=i+1;
    od;
someadt sort={(integer, >), (real, =) (integer, <)}.
```

上述 Apla 程序定义了一个泛型方法,其泛型参数为 3 种类型的 ADT,根据实例化是传入的 ADT 参数可以实现求整数类型数组的最长升序、整数类型数组最长降序和实型数组最长平台数 3 个算法.

2 Apla 语言及其程序自动生成系统

2.1 抽象程序建模语言 Apla

Apla[16] 语言是 PAR 方法[17] 中定义的一种抽象程序设计语言,是 PAR 方法[18] 和 PAR 平台最重要的一个组成模块. 在设计 Apla 语言时,始终围绕着功能和数据的抽象理念,通过抽象,使得 Apla 语言易理解、易使用且易于验证. Apla 语言既是在 PAR 平台中 Radl→Apla 程序转换器的目标语言,同时也可以作为 Apla 到 Ada、C++、C#和 Java 等可执行程序转换器的源语言[16-21]. 基于关系代数的数据库处理机制、多媒体数据库处理机制、异常处理机制和本地事务处理机制已经成功融入 Apla 语言中,使 Apla 语言的实用性更强,更适合于企业进行软件开发[22-23].

2.2 Apla→Java 程序自动生成系统

由于使用 Apla 语言编写的程序不能直接运行,需要通过 PAR 平台中的 Apla→C++、Java 和 C#等程序自动转换系统,将 Apla 程序转换成对应语言的可执行程序. 同时,基于形式化和自动化的 PAR 方法可以对 Apla 程序进行验证,验证后的 Apla 程序通过自动转换系统进行转换,能够保证生成的目标语言程序的正确性.

Apla→Java 程序自动生成系统,共有 2 个重要组成部分,转换器和自定义 Java 可重用构件库. 其总体结构如图 1 所示. 转换系统由词法、语法分析以及程序转换和异常处理 4 个模块组成,并依赖由 Table 支撑库和 Strulib 部件库组成的自定义 Java 可重用构件库运行.

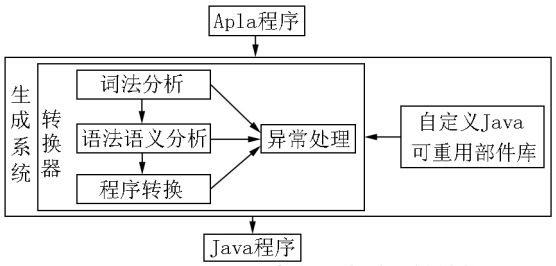

图 1 Apla→Java 程序自动生成系统结构图

3 Web 服务作为 action 参数

3.1 Web 服务作为 action 参数的重要性

随着经济的快速发展,人们对计算机软件的需求也变得纷繁复杂,对软件系统的更新迭代提出了

更高的要求. 传统的软件重用理论,对这种持续增加和改变的需求难以给出有效的解决方案. 近年来,人们开始研究在分布式异构的环境中用 Web 服务作为基本模块来满足用户需求,这种架构被称为面向服务的计算,这是一种新的计算规范,将服务作为构件,能够支持分布式应用,成本低且能敏捷开发. 通过将服务进行组合,满足用户的需求,大大提升软件的重用性[24]. Web 服务是一种平台独立、耦合度低、自包含的 Web 应用程序. 通过使用 XML 来描述和发布这些 Web 应用程序,Web 服务能够在一些普通的计算中提供一些服务,然后完成一个具体的任务,解决一个具体的问题.

因此,在 Apla 语言的泛型机制中,实现 Web 服务作为 action 参数,对提升 Apla 语言的重用性起着举足轻重的作用.

3.2　添加 Webservice 构件

为了能够实现在 Apla 泛型程序中 Web 服务作为 action 参数的泛型机制,将 Webservice 作为关键字添加到了 Apla 语言中,并在构件库中的 Strulib 库添加了 Webservice 构件. Webservice 构件一共有 2 个重要的属性: 一个是 Web 服务的 URL,另一个就是 Web 服务提供的方法. 因为在 Apla 语言中新增了 Webservice 关键字,相应地将转换系统的词法、语法分析以及程序转换 3 个模块关于新增的关键字进行了修改. 在词法分析模块中,先将 ID 值 610 赋给 Webservice,再对语法分析模块进行了修改,添加了关于 Webservice 的语法检查. 在 Apla 语言中定义一个 Webservice 的 action 的示例如下:

```
Webservice name ={ "URL", "METHOD"};
```

其中 URL 为 Web 服务的 Wsdl 地址,METHOD 为所要调用的 Web 服务的方法名.

3.3　Web 服务作为 action 参数

当在 Java 语言中调用 Web 服务时,首先需解析所调用的 Web 服务的 Wsdl(Web 服务描述语言) ,从而获取并引用所调用的 Web 服务的 Jar 包. 为能在 Apla→Java 程序自动生成系统中,实现 Web 服务作为参数这一泛型机制,在 Strulib 构建库中,对传入的服务进行解析,将调用的 Web 服务的 Jar 包动态加载到 Java 虚拟机中[25]. 并利用反射机制获得服务的代理类及代理类对象[26],最终通过使用代理类的实例化对象调用该服务. 在 Apla 语言中,Web 服务作为 action 参数的示例如下:

```
program web_example_one;
procedure gen_exam( someaction ff( x: integer; y: integer) : integer; ) ;
var
x: integer;
begin
X: =ff( 1,1) +ff( 3,3) ;
writeln( "X =", X) ;
end;
webservice web_value1 ={ "http: //192. 168. 1. 24: 8080/ComputeService/ComputeServicePort? wsdl", "addition"};
webservice web_value2 ={ "http: //192. 168. 1. 24: 8080/ComputeService/ComputeServicePort? wsdl", "multi"};
webservice web_value3 ={ "http: //192. 168. 1. 24: 8080/ComputeService/ComputeServicePort? wsdl", "subtract"};
someaction gen_exam ={ web_value1, web_value2, web_value3};
procedure gen_fun_1: new gen_exam ( web_value1) ;
procedure gen_fun _2: new gen_exam ( web_value2) ;
procedure gen_fun _3: new gen_exam ( web_value3) ;
begin
writeln( "web 服务作参数") ;
gen_fun _1 ( ) ;
gen_fun _2 ( ) ;
gen_fun _3 ( ) ;
end.
```

上述 Apla 程序,先定义了一个泛型过程 gen_exam,其泛型参数为 Web 服务; 然后定义了 3 个 Webservice 的 action,分别为 web_value1、web_value2 和 web_value3,3 个 Webservice 变量均调用了部署在 IP 地址为 192. 168. 1. 24 的服务器的 ComputeService 服务,其中 web_value1 调用该 Web 服务的 addition(加法) 方法,web_value2 调用该 Web 服务的 multi(乘法) 方法,web_value3 调用该 Web 服务的 subtract(减法) 方法; 最后分别使用 web_value1、web_value2和 web_value3 对泛型过程 generic 进行了实例化. 经过 Apla→Java 程序自动生成系统转换成Java代码.

3.4　泛型安全机制

泛型约束是可信软件的一种重要技术,主要用于保证泛型程序的安全性[27]. 为了能够充分地保证 Apla 泛型程序的安全性,在 action 参数化与泛型 ADT 的 Apla 程序中. 定义了 someaction 和 sometype 这 2 种数据类型来存储泛型程序的安全值,在实例

化泛型过程和函数时,程序转换系统会对传入的实例化参数对照安全值进行核验. 若传入的参数在安全值的范围内,则为安全有效的实例化参数,反之则是不安全无效的实例化参数.

对于 Apla 泛型程序中 action 参数化的泛型安全约束机制,一共定义了 2 种. 一种是在针对泛型过程和函数,在实例化之前定义一个与之同名的 someaction 变量,并将合法的 action 参数赋值给该变量. 当实例化时,传入的 action 值将和 someaction 变量中存储的 action 值进行核验,若值在 someaction 中则为合法,反之则为非法. 另一种是在泛型过程和函数初始定义的同时,也定义 action 参数的形参类型、个数和返回值的类型,在泛型过程和函数的实例化时将对传入的 action 与在泛型函数和过程定义时给定的 action 参数的形参类型、个数和返回值类型进行比对核验,符合则为合法参数,反之则为非法参数. 其流程如图 2 所示.

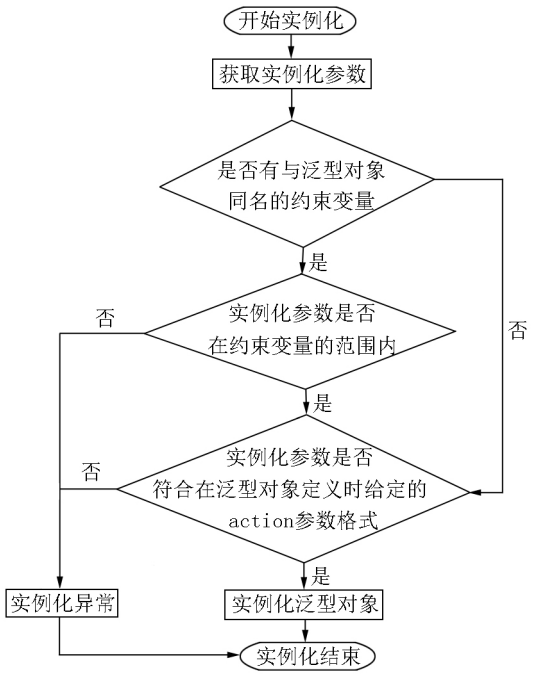

图 2 泛型安全检查流程图

在 Apla 程序中,对泛型安全机制进行检查的示例如下:

```
procedure generic_action_a( someaction region_a ( param_a: integer; param_b: integer) : integer) ; //定义一个泛型过程,给定 action 参数安全值,有 2 个 interger形参,返回值为 integer 的函数.
begin
result_a: = region_a( param_a, param_b) + region_a ( param_b, param_a) ;
writeln( "result_a = ", result_a) ;
end;
function generic_action_b ( someaction action_b ( real_a: real; real_2: real) : real; ) ; //定义一个泛型过程,给定 action 参数安全值,有 2 个 real 形参,返回值为 real 的函数.
begin
result_b: = action_b ( param_a, param_b) + action_b ( param_b, param_a) ;
writeln( "result_b = ", result_b) ;
end;
function fun_examp_a( x: integer; y: integer) : integer;
begin
fun_examp_a: = x + y;
end;
function fun_examp_b( x: integer; y: char) : integer;
begin
fun_examp_b = x;
end;
function fun_examp_c( x: integer; y: integer) : integer;
begin
fun_examp_c: = x + y;
end;
someaction generic_action_a = { fun_examp_a, fun_examp_b} ;
//该 someaction 约束的泛型对象为 generic_action_a.
procedure generic_a: new generic_action_a ( fun_examp_a) ;
procedure generic_b: new generic_action_a ( fun_examp_b) ;
procedure generic_c: new generic_action_a ( fun_examp_c) ;
procedure generic_d: new generic_action_b ( fun_examp_c) ;
```

当 generic_a 进行实例化时,传入的参数为 fun_examp_a,未超出泛型对象 generic_action_a 的安全值范围,generic_a 能成功实例化. 当 generic_b 进行实例化时,传入的参数为 fun_examp_b,也在泛型对象 generic_action_a 的安全值的范围内,但是函数 fun_examp_b 的形参类型,与在泛型对象 generic_action_a 定义时要求形参类型不同,故实例化失败. 在对 generic_c进行实例化时,传入的参数为 fun_examp_c,超出了泛型对象 generic_action_a 的范围,故实例化失败. 当 generic_d 进行实例化时,传入的 action 参数为 fun_examp_c,在程序中未定义变量名为 generic_action_b 的 someaction 变量,根据在泛型对象 generic_action_b 定义时给定的 action 的格式进行安全性检查,发现 fun_examp_c 形参和返回值均为 integer,与

在泛型对象 generic_action_b 定义时要求的形参和返回值格式不符,故实例化失败.

4　Web 服务作为 action 参数的应用

为能将 PAR 方法和 PAR 平台在世界范围内推广,将现有的很多科研成果部署在云端服务器上,方便广大用户使用. 在这个过程中,需要用到较多的 Web 服务. 通过 PAR 方法建模进行软件开发主要有以下几个重要步骤: (i) 使用基于结构化的自然语言[28-29] (Sruetured Natural Language,简称 SNL),描述问题的需求,建立需求模型; (ii) 在使用 SNL 需求模型转换服务,将需求模型转换为使用 Radl 语言建立的形式化规约模型; (iii) 使用 Apla 程序模型转换服务,将形式化规约模型转换为 Apla 程序模型; (iv) 再使用可执行程序转换服务,将 Apla 程序模型转换为对应的可执行程序. 其开发流程如图 3 所示.

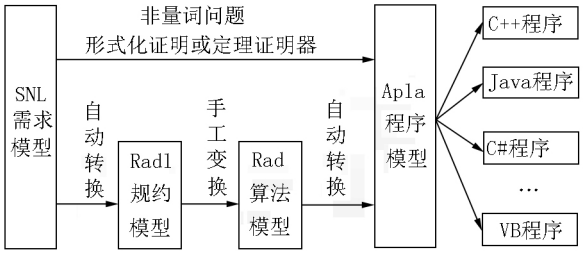

图 3　使用 PAR 方法开发流程

在整个过程中,需要用到 3 个 Web 服务,分别为 SNL 需求模型转 Radl 形式化规约模型服务、Radl 形式化模型转 Apla 程序模型服务和 Apla 程序模型转可执行程序服务. 为减少编写代码的工作量,提高程序的可重性,在开发过程中,将 Web 服务作为泛型参数,实现上述功能. 其 Apla 代码为:

```
program webservice;
procedure generic ( someaction ff ( x: string): string; string);
var
x: string;
begin
x: =ff( x);
end;
//SNL 需求模型转 Radl 规约模型
webservice web_SnlToRadl ={ "http: //192. 168. 1. 30: 8080/TranslateService/TranslateServicePort? wsdl", "SNLtoRadl"};
//Radl 规约模型转 Apla 程序模型
webservice web _ RadlToApla ={ "http: //192. 168. 1. 34: 8080/TranslateService/TranslateServicePort? wsdl", "RadltoApla"};
//Apla 程序模型转可执行程序
webservice web_AplaToExe ={ "http: //192. 168. 1. 29: 8080/TranslateService/TranslateServicePort? wsdl", "AplatoExe"};
someaction generic = { web_SnlToRadl,web_RadlToApla,web_AplaToExe };
procedure generic1: new generic( web_ SnlToRadl);
procedure generic2: new generic ( web _ RadlToApla);
procedure generic3: new generic( web_AplaToExe);
begin
var
Snl: string,Radl: string,Apla: string,Exe: string;
Radl = generic1( Snl);
Apla = generic2( Radl);
Exe = generic3( Apla).
```

通过 Apla→Java 程序自动生成系统,将上述 Apla 程序转换为 Java 程序添加到 Web 工程中,然后部署在云服务器上,实现了使用 PAR 方法进行在线软件开发的整个流程.

根据用户选择的转换类型,将对应的 Web 服务作为 action 参数传入,最终实现相应的模型转换功能.

5　结束语

泛型程序设计能够有效提高编程效率,提高程序的可重用性和可靠性. 将新型泛型机制应用在 PAR 平台中,使得 Apla 语言的泛型机制更加丰富和先进. 在 Apla 语言中设计并实现了 Web 服务构件,实现了对 Web 服务的支持,实现 Web 服务作为 action参数的泛型机制,有助于 Apla 语言在服务组合领域中广泛应用.

通过枚举和定义匹配相结合的方法进一步完善了 Apla 中的泛型安全机制,保障了 Apla 泛型程序的安全性.

最后,通过列举了一个将 Web 服务作为 action 参数的示例,对新增的泛型机制进行了测试,测试结果说明本文新增的 action 泛型机制较为成功.

6　参考文献

[1] Musser D R,Stepanov A A. Generic programming [M]. Berlin: Symbolic and Algebraic Computation,1988: 13-25.

[2] Garcia R,Jarvi J,Lumsdaine A,et al. A comparative study

of language support for generic programming [J]. OOPSLA,2003,36(3):115-134.
[3] Kermarrec Y,Pautet L,Tardieu S. GARLIC: generic Ada reusable library for interpartition communication [EB/OL]. [2019-02-13]. https://dl.acm.org/doi/10.1145/376503.376591.
[4] 张玉春,程春英,李海峰. 浅谈 C#泛型和 C++模板 [J]. 内蒙古民族大学学报,2008,14(2):51-52.
[5] Ghosh D. Generics in Java and C++: a comparative model [J]. ACM Siglan Notices,2004,39(5):40-47.
[6] 韩志强. 对. NET 平台中泛型技术的探究 [J]. 赤峰学院学报:自然科学版,2010,26(11):23-24.
[7] Milner R,Tofte M,Harper R,et al. The definition of Standard ML [M]. Massachusetts: MIT Press,1997.
[8] Stroustrup B. The C++ programming language [M]. New York: Pearson Education,2013.
[9] Gosling J,Joy B,Steele G,et al. The Java language specification [M]. New York: Pearson Education,2014.
[10] Meyer B. Eiffel: the language [M]. New York: Rentice Hall,1992.
[11] Hutton G. Programming in [M]. Haskellambridge: Cambridge University Press,2007.
[12] Xue Jinyun. Genericity in PAR Platform [EB/OL]. [2019-01-12]. https://link.springer.com/chapter/10.1007/978-3-319-31220-0_1.
[13] 徐华珍,薛锦云,朱小征. Apla→Java 程序生成系统中泛型机制实现方法研究 [J]. 江西师范大学学报:自然科学版,2017,41(1):52-55,92.
[14] 常亮,刘进,古天龙,等. 基于动态描述逻辑的语义 Web 服务组合 [J]. 计算机学报,2013,36(12):2468-2478.
[15] 刘阳. 云计算中服务组合与选择技术研究 [D]. 北京:北京邮电大学,2013.
[16] 游珍,薛锦云,应时. Apla 语言中并发分布式机制的研究 [J]. 计算机科学,2012,39(1):104-108.
[17] Xue Jinyun. PAR method and its supporting platform [C]// Proceeding of International Workshop on Formal Method for Devecoping Software. Macao: UNU-IIST,2006:348.
[18] 王昌晶,薛锦云. PAR 平台从规约出发的算法推导与自动生成 [J]. 计算机工程与应用,2007,43(2):41-42,59.
[19] 骆健. APLA-JAV. 程序自动转换系统的研制 [D]. 南昌:江西师范大学,2002.
[20] 赖勇. APLA 到 C++自动程序转换系统的研制 [D]. 南昌:江西师范大学,2002.
[21] 左正康. Apla→C#自动程序转换系统的设计与实现 [D]. 南昌:江西师范大学,2004.
[22] 朱小征,薛锦云,夏鲸,等. 在建模语言 Apla 中实现多媒体数据库应用的方法研究 [J]. 江西师范大学学报:自然科学版,2017,41(1):46-51.
[23] 朱小征. 若干软件新技术在 Apla→Java 程序自动生成系统的实现研究 [D]. 南昌:江西师范大学,2016.
[24] Blake M B. Decomposing composition: service-oriented software engineers [J]. Software IEEE,2007,24(6):68-77.
[25] 左天军,朱智林,韩俊刚,等. Java 动态类加载分析 [J]. 计算机科学,2005,32(4):194-196.
[26] 尹松强,傅鹂. Java 反射机制探究 [J]. 软件导刊,2008,7(11):85-87.
[29] 左正康,薛锦云. Apla 中泛型约束机制研究 [J]. 软件学报,2015,26(6):1340-1355.
[28] 张少平. 基于结构化自然语言的算法规约研究 [D]. 南昌:江西师范大学,2003.
[29] 王昌晶,薛锦云,左正康. SRLtoRadl 生成系统及其范畴论语义 [J]. 电子学报,2014,42(1):137-143.

The Implementation and Application of New Action Generic Mechanism in PAR Platform

WANG Xiong[1,2],XUE Jinyun[2*],LIU Zhenwei[1,2]
(1. College of Computer Information Engineering,Jiangxi Normal University,Nanchang Jiangxi 330022,China;
2. The State Base of Networked Supporting Software of International S/T Cooperation,
Jiangxi Normal University,Nanchang Jiangxi 330022,China)

Abstract: In the process of software development,the generics mechanism in the development language can significantly improve the efficiency and reliability of software development. However,the high-level languages generally only take types and subroutines(such as procedures,functions and methods) as parameters,which limits the application of the generic mechanism. The deep understanding of the new generic mechanism proposed by Professor Xue Jinyun is made,and the generalization mechanism of taking action as generic parameters and its realization method are focused on in this paper. The generic mechanism of taking Web service as action parameters is realized in the Apla→Java program automatic generation system,and the generic security constraint mechanism in the Apla language is improved,and the new generic mechanism is successfully applied to the PAR platform.

Key words: Apla; genericity; PAR method; PAR platform (责任编辑:冉小晓)