文章编号:1000-5862(2016)01-0077-06

一种抽象泛型机制的新型 Java 实现

田 方,石海鹤*,左正康,王昌晶,薛锦云
(江西师范大学江西省高性能计算重点实验室,江西 南昌 330022)

摘要:泛型程序设计可大幅度提高程序的可重用性、可靠性和开发效率,高抽象的泛型机制则有助于降低泛型编程的复杂度. 该文介绍了一种抽象泛型机制的核心思想和编程技术,分析了新型 Java 中与之对应的实现机理,并归纳出相应的转换规则,且在既有平台中实现了抽象泛型程序到新型 Java 泛型程序的自动转换和运行,最后通过一个 Kleen 算法实例,展示了抽象泛型机制的实际运用和相应 Java 程序的自动生成. 抽象泛型机制的新型 Java 实现,在降低可重用算法构件的设计复杂度、提高算法构件可靠性的同时,也为泛型构件的设计和描述提供了行之有效的新途径.

关键词:抽象泛型机制;Java 语言;可重用性;算法构件
中图分类号:TP 311 **文献标志码**:A **DOI**:10.16357/j.cnki.issn1000-5862.2016.01.14

0 引言

软件的开发效率和安全性一直是现代软件工程中重点关注的焦点,提高软件的可重用性为解决这些问题提供了有效的方法. Doug Mcllroy 于 1986 年提出了可重用软件部件的概念[1],使得软件的生产效率得到较大的提高. 早期在 Ada 语言中,用户首先可以自行定义类型或者操作作为参数,在执行程序时,对这些参数赋值,利用这种方法可以设计出更具有通用性的构件,使构件的可重用性大大提高.

由于传统程序开发的可重用构件的抽象程度不高,反而使得问题变得更加复杂;而泛型程序的高抽象性和高可重用性特点为开发重用构件提供了有效的手段[2]. 当前许多高级程序设计语言中含有反映泛型程序设计思想的语言机制,如 C++中的模板、ML 中的多态函数、Java 中的类型参数化等. 在抽象程序设计语言 Apla[3] 中,也定义了大量的机制来支持泛型程序设计,使 Apla 语言更加简便、高效,描述能力更强. 泛型程序体现了方法学中的一种共性和特性的设计思想,其功能是在不影响运行结果的情况下使得算法结构等共性特点从具体的应用中分离出来,从而增加了程序的可读性和可重用性[4]. 随着软件的需求不断提高,泛型程序得到了广泛的应用,也使得泛型机制得到不断优化和扩展,为解决更多复杂问题提供了有效的解决方法.

高抽象的泛型机制使得算法设计思想成为关注焦点,而不用过多考虑繁杂语言细节,有助于简化泛型程序的设计和描述,提高泛型构件的可重用性,其可验证性也保证了泛型构件的可靠性. 如何在实现抽象泛型机制的同时,保持上述这些优势,是一个具有实际意义的研究课题.

本文通过分析 Apla 语言中抽象泛型机制的特征,研究了 Java 中的相关机制,总结和归纳出支持 Apla 抽象泛型机制的新型 Java 实现方法,并进一步在既有平台中予以实现.

1 泛型机制

1.1 Apla 中的抽象泛型机制

Apla(abstract programming language)语言是形式化方法 PAR[6-9] 中定义的一种抽象程序设计语言,充分体现了功能抽象、数据抽象的程序设计思想,具有简单实用、便于程序开发等特点,易于阅读理解和验证,且易于转换成 C++、Java 等可执行语言程序.

泛型程序设计是一个参数化的程序设计,其中参数是指数据、数据类型、操作、构件、服务、子系统

收稿日期:2015-09-17
基金项目:国家自然科学基金(61272075,61363012,61363013,61462039)和江西省自然科学基金(20142BAB217026)资助项目.
通信作者:石海鹤(1979-),女,江西乐平人,副教授,博士,主要从事可信软件和基于组件的软件工程研究.

等,并以此为基础,编制出具有通用性的程序. Apla 泛型机制主要分为以下 2 种类型:

(i) 类型参数化. 在 Apla 语言中,引入关键字 sometype 来定义程序类型参数、类型变量、过程函数参数的类型,也可以直接在类型声明中以参数的形式来说明组合数据类型的基本类型,使得程序实现泛型化. 例如,定义泛型程序的语法结构为:

program 〈程序名〉(sometype 〈类型参数〉);

对于泛型程序,实例化时需使用具体类型来作为类型实参. Apla 中实例化的语法结构为:

program 〈新程序名〉: new 〈程序名〉(具体类型);

(ii) 子程序参数化. Apla 中提供了 func 和 proc 来声明子程序的函数参数和过程参数. 泛型子程序可以带有普通值参、类型参数或者子程序参数. 在 Apla 中定义泛型子程序的语法结构为:

procedure 〈过程名〉|function 〈函数名〉(形参表);

形参表: = [普通值参] | [sometype 〈类型参数表〉] | [proc 〈过程形参表〉] | [func 〈函数形参表〉]

泛型子程序使用之前需进行实例化,才能使用该子程序. 实例化时,实参需和形参在参数个数、参数类型上完全匹配. 实例化泛型子程序的语法结构为:

procedure [function] 〈新过程名或新函数名〉: new 〈泛型过程名〉(〈类型实参表〉|〈子程序实参表〉);

使用子程序参数进行泛型程序设计,可编写出清晰、易理解、可重用的程序,反映出程序中算法的实质,是编写抽象程序的有力工具.

Apla 语言的特点之一是能较好地描述和开发复杂算法程序,使用 Apla 设计的泛型算法结构具有较高的可重用性[10].

1.2 Java 泛型机制

泛型是 Sun 公司发布的 JDK5.0 中的一个重要特性,主用基于类型参数化来进行泛型 Java 程序设计[11-12]. 泛型的引入简化了 Java 程序设计,使得一些要在运行时才可能暴露出来甚至极难发现的错误在编译时就可以发现,进一步加强了代码的类型安全检查,增强了 Java 程序的健壮性和可读性[13-14].

1.2.1 Java 泛型机制 Java 语言中支持泛型程序设计的机制 (i) Java 泛型类. 通过类型参数来抽象数据类型,而不是将变量的类型都定义成 Object,从而使泛型的类型安全检查在编译阶段,所有的类型转换均是自动和隐式,保证了类型的安全性. 泛型类声明的参数类型 T 可用来定义类中的成员变量、方法参数及方法返回值类型. 泛型类的格式为:

[访问修饰符] class 类名〈类型参数〉{类体…}. 其中类型参数个数可以同时定义多个.

(ii) Java 泛型方法. 指含有形式类型参数的方法,是否拥有泛型方法与其所在的类是否泛型无关,只需将参数类型置于方法名或参数之前. 泛型方法的使用使得程序变得高效和简洁. Java 泛型方法的声明如下:

[访问权限修饰符] [static] [final] 〈类型参数列表〉返回值类型方法名([形式参数列表])

其中 [ ] 表示为可选的,类型参数列表用 T 来表示未知类型,使用时可用 extends 和 super 关键字分别来指定 T 类型的上限和下限范围.

(iii) Java 泛型接口. 泛型接口与泛型类的定义相似,尖括号中表示类型参数,同样接口中也可以定义一个或一个以上个数的类型参数,泛型接口与一般接口一样不能直接用来创建对象,但可以用来声明一个对象变量,声明方式如下:

[访问权限修饰符 public] interface 接口名〈类型参数列表〉

泛型接口的使用同接口一样,需要通过类来实现接口. 若需要在实现泛型接口时,保留接口中的泛型,定义类时就必须要保留泛型接口中的泛型声明. 实现泛型接口的类必定为泛型类,且其类型参数的个数必须大于或等于泛型接口中类型参数的个数.

(iv) 通配符. Java 中使用通配符(?)来解决使用固定泛型类型时的不便. 通配符的作用是表示一组类型的集合,可以匹配特定范围内的类型. 在使用泛型类的时候,既可以指定一个具体的类型,如 List〈String〉,也可以用通配符"?"来表示未知类型,如 List〈?〉. 通配符所代表的其实是一组类型,但具体的类型是未知的. 通配符的使用形式有 3 种:无界通配符:?,上限通配符:? extends T,以及下限通配符:? super T.

1.2.2 "擦除"原理和泛型使用限制 泛型的信息只在编译阶段处理,然后擦除,而不会在运行阶段出现,即在编译阶段,生成的 Java 字节代码中不包含泛型中的类型信息,类型参数会被编译器在编译的时候去掉,该过程称为类型擦除. 由于泛型擦除的隐蔽性,泛型的使用存在以下约束限制[15]:

(i) 不能用泛型类型参数创建实例对象,如 T object = new T(). 由于类型擦除原理,编译时所有泛型类型都会变为相应的原始类型,这时泛型 T 是没有类型的,是无效的.

(ii) 不能使用泛型类创建数组,如 T []

elements = new T [capacity]. 如果先创建一个类型数组而后转型为 Object,会导致数组创建失败和类型擦除问题.

(iii) 不能将泛型用在静态的上下文中. 泛型类的所有实例均有相同的运行时类,这个泛型类的静态变量和静态方法被它的所有实例所共享,因此在泛型类中定义的静态变量和静态的泛型方法是非法的,会出现程序安全隐患.

(iv) 不能用基本类型替换类型参数,如 List〈int〉,其中 int 为基本类型,必须使用其包装类型 Integer.

(v) 异常类不能使用泛型,否则将泛型用在异常捕获中将会出现变异错误.

1.2.3　泛型机制的作用　在 Java 语言中该机制的作用有: (i) 泛化程序. 可以用 T 代表任意类型. Java 中,不仅语言、类型系统和编译器能支持泛型,而且类库中许多重要类也均泛型化. (ii) 消除类型的强制转换,提高了类型的安全性. 泛型可以消除源代码中的许多强制类型转换,可以使代码更加可读,并在编译阶段暴露类型错误. (iii) 向后兼容性. 支持泛型的 Java 编译器可以用来编译经过泛型扩充的 Java 程序,同时也可编译非泛型扩充的 Java 程序. (iv) 增加程序的可重用性. 程序可以创建以类型安全方式使用各种类型数据的类、方法、接口,类型转换均自动和隐式进行,提升了代码的可重用能力.

2　抽象泛型机制的新型 Java 实现

2.1　抽象泛型机制的 Java 实现

2.1.1　类型参数化泛型机制的实现　为便于阐述 Apla 泛型程序编程技术以及与之对应新型 Java 的实现机理,下面给出一个包含多种泛型结构的 Apla 泛型程序示例:

```
program BubbleSortProg( sometypebasetype) ;
constn = 9;
var a: array [0. . 9,basetype]; x,i: integer;
procedure BubbleSort( var a: array [0. . 9,base-
type]; n: integer) ;
…
begin
program int_BubbleSortProgram: new BubbleSort-
Prog ( integer) ;
end.
```

根据 Apla 抽象泛型机制和归纳的统一设计思想,得到转换规则如下:

(i) 整个 Apla 程序可转换为一个公共的 Java 泛型类,其中程序名及其类型参数对应转换为带泛型的类名,即为 BubbleSortProg〈T〉类形式,根据类型参数的个数可得到相应 T 类型的个数.

(ii) 常量定义 const 可直接转换为 Java 类中的全局定义形式,转换的对应语句为 public static [基本类型] .

(iii) 变量定义部分 var 定义的变量,转换后可作为公共类中成员变量. 当变量定义中涉及到类型参数时,则转换为相应的 T 类型变量.

(iv) 泛型程序中的普通过程 procedure 和函数 function,则转换为公共泛型类中的 public 方法,若该方法有返回值类型或者方法参数的类型,则也将其定义成泛型格式.

(v) 程序体转换为公共方法 maincall( ) ,并在此方法中一对一转换操作语句.

(vi) 最后为泛型程序的实例化部分,对应为 Java 类中的 main( ) 方法里语句. 在 main( ) 方法中首先实例化创建该泛型类,并带有相应的类型参数,然后使用此实例化变量调用 maincall( ) 方法,实现泛型类中类型参数的调用.

根据转换规则,可将上述 Apla 程序转换,得到相应的 Java 程序:

```
import…;
public classBubbleSortProg〈T〉{
public static int n = 9;
int x,i;
T [] a = (T []) new Object [n + 1];
public T BubbleSort( T [] a,int n) {…}
public void maincall( ) {
…
this. BubbleSort( a,n) ;
…}
public static void main( String [] args) {
BubbleSortProg〈Integer〉aco_int = new Bubble-
SortProg〈Integer〉( ) ;
aco_int. maincall( ) ;
…}
}.
```

2.1.2　子程序参数化泛型机制的实现　通过深入分析 Apla 抽象泛型机制中子程序参数化的设计思想,研究了 Java 现有机制,得到两者间的生成规则如下: (i) 首先将整个程序转换为一个 Java 公共类,程序名作为类名. (ii) 变量定义部分直接转换为相应类型的成员变量. (iii) 程序中的普通过程或函数

(不含有过程或函数作参数的),依旧直接转换为公共类中的 public 方法. 当检测为子程序时,将该过程或函数名转换为公共类中的一个抽象类,子程序参数则转换为抽象类中的抽象方法,该过程或函数体则转换为一个普通方法体. (iv) 整个程序体的实现则放入 maincall 方法中. (v) 子程序实例化部分转换. 根据关键字 begin 后是否含有关键字 procedure 或 function,new 判断进入泛型子程序的实例化. 具体实现为: 将过程名或函数名作为公共类中内部类继承抽象类,并在子类中具体实现抽象类的抽象方法. (vi) 在 maincall 方法中实例化子类对象,在 main 方法中实例化公共类对象,完成该对象对 maincall 方法的调用,实现泛型子程序的设计.

2.2 实例设计

这里,以 Apla 语言描述的经典 Kleene 算法为例,来展示 Apla 抽象泛型机制到 Java 的实现. 该算法将顶点对最短路径、传递闭包和最大容量路 3 个算法程序抽象为一个算法程序. 本文以 Apla 语言泛型子程序方法来描述该程序.

```
program Kleen;
const maxnum = 99999; num = 3;
var i,j,k,s: integer;
c1,c3: array [1..num,array [1..num,integer]];
c2: array [1..num,array [1..num,boolean]];
procedure kleene( sometype elem; func ⊙,□( a,b: elem): elem; n: integer; c: array [1..num, array [1..num,elem]]); … ①
va ri,j,k: integer;
begin
k: =1;
do (k≤n) →
foreach( i,j: 1≤i,j≤n: c [i,j]: =c [i,j] ⊙ (c [i,k] □c [k,j]););
k: =k+1; od;
i: =1;
doi≤n→j: =1;
do (j≤n) →write( c [i,j]," ,"); j: =j+1; od;
i: =i+1;
od;
end;
procedurefloyd: new kleene( integer; min; +); …②
procedure close_set: new kleene (boolean; ∨; ∧); …③
procedure capcity: new kleene (integer; max; min); …④
begin …⑤
writeln( "所有顶点对的最短路径:"); foreach( i,j,k: 1≤i,j,k < num: c1 [i,j]: = maxnum; s: =0;); c1 [1,3]: =11; c1 [3,1]: =3; c1 [1,2]: =4;
c1 [2,1]: =6; c1 [2,3]: =2;
c1 [1,1]: =0; c1 [2,2]: =0; c1 [3,3]: =0;
floyd( num,c1);
writeln( "传递闭包"); c2 [1,1]: = false; c2 [1,2]: = true; c2 [1,3]: = false; c2 [2,1]: = true; c2 [2,2]: = false; c2 [2,3]: = true; c2 [3,1]: = false; c2 [3,2]: = false; c2 [3,3]: = false;
close_set( num,c2);
writeln( "最大容量路");
foreach( i,j: 1≤i,j≤num: c3 [i,j]: = max-num;); c3 [1,3]: =11; c3 [3,1]: =3; c3 [1,2]: =4; c3 [2,1]: =6;
c3 [2,3]: =2; c3 [1,1]: =0; c3 [2,2]: =0; c3 [3,3]: =0;
capcity( num,c3);
end.
```

上述程序的①处定义了一个泛型子程序,其中所带形参表为一个类型参数 elem,2 个函数参数名为⊙□和,并且⊙和□可分别表示为要进行抽象操作的特殊符号,一个普通整型参数以及一个 2 维类型数组,这些未知类型将会在程序实例化时得到具体的类型实参. 在该泛型子过程后定义了实例化中的各个子过程的公共操作方法,由具体实参实现不同的功能. 在②、③、④处为各个子过程的实例化. 在⑤begin 处为所有顶点对最短路径、传递闭包和最大容量路 3 个算法的主程序具体实现.

Java 语言中泛型机制具有泛化程序和增加程序可重用性的特点,更与 Java 语言的抽象类、继承、覆盖等特性结合,可有效设计出 Kleen 算法构件. 根据前述转换规则,得到新型 Java 实现的泛型 Kleen 算法如下:

```
public class Kleen {
abstract class kleene <T>{
public abstract T BigMulti( T a,T b);
public abstract T BigPlus( T a,T b);
public void genericrun( int n,T [][] c) {
for( k=1; k<=n; k++)
for( i=1; i<=n; i++)
for( j=1; j<=n; j++)
c [i] [j] = BigMulti (c [i] [j],( BigPlus
```

```
( c [i] [k],c [k] [j]) ) ) ;
    System. out. println( c [i] [j] + ",") ; }
                …
    class floyd 〈T〉 extend skleene 〈T〉{
    public T BigMulti( T a,T b) {
    return Math. min( a,b) ;  }
    public T BigPlus( T a,T b) {
    return a + b;  }  }
    class close_set 〈T〉 extend skleene 〈T〉{
    public T BigMulti( T a,T b) {
    return a || b;  }
    public T BigPlus( T a,T b) {
    return a && b;  }  }
    class capcity 〈T〉 extend skleene 〈T〉{
    public T BigMulti( T a,T b) {
    return Math. max( a,b) ;  }
    public T BigPlus( T a,T b) {
    return Math. min( a,b) ;  }  }
    public void maincall( ) {
    …
    floyd 〈Integer〉 floydproc = new floyd 〈Integer〉( ) ;
    floydproc. genericrun( num,c1) ;  …}
    public static void main( String [] args) {
    Kleenaco = newKleen ( ) ;
    aco. maincall( ) ;  }
    }.
```

在上述 Java 程序中,通过生成一个泛型抽象类来完成对 Kleen 算法共性功能的封装,按照算法构件的开发理念,首先在抽象类中定义 2 个抽象方法来描述构件能够提供的服务操作,使用普通方法 genericrun 来设计实现构件本身的功能,然后通过子类对抽象类的继承具体覆盖实现构件中提供的服务操作,最终通过泛型的实例化完成对抽象类中类型参数的传递,从而实现外界对构件接口的调用.

2.3　系统实现

Apla 到 Java 程序自动生成系统[10]是 Apla 程序到 Java 程序的桥梁,是支持 PAR 方法开发可靠 Java 程序的必要支撑工具. 它的目标是在自定义 Java 可重用构件库的支持下将 Apla 语言描述的抽象程序,按照一定的转换规则,自动地转换为与之相对应的 Java 程序,最后基于 JDK 环境和可重用构件库来编译运行 Java 程序.

扩充了 Java 构件库,并重构 Java 程序生成系统中泛型程序生成部分的代码,以支持抽象泛型机制的新型 Java 实现.

以前述 Kleen 算法的 Apla 程序作为扩充后 Java 程序生成系统的输入,可以自动生成对应的 Java 程序.

对生成的 Java 程序编译、运行,可得到程序的运行结果.

3　结束语

泛型程序设计作为面向对象编程技术的补充和扩展,受到了越来越多的关注,泛型程序的安全性和可重用性特点为软件开发提供了高可靠性和生产效率. 目前,国内外也将泛型程序设计广泛应用于实际软件开发中,主要使用其类型参数的特性来设计可重用构件.

本文介绍了一种抽象的 Apla 泛型机制以及 Java 新型泛型机制,分析了两者的核心思想和程序设计方法,前者抽象程度高,具有强大的复杂算法描述能力和构件设计能力,支持类型参数化和子程序参数化,后者则仅支持类型参数化,Apla 支持的子程序参数化在其中并无直接对应. 本文综合两者特点,设计了 Java 实现 Apla 泛型程序的相关规则,并对既有系统进行了扩充以支持从泛型 Apla 程序到 Java 程序的自动生成,使得在保持抽象泛型机制相关优势的同时,保证了程序设计的高效性和所得 Java 程序的可靠性. 通过一个精巧的 Kleen 泛型算法设计,展示了从抽象泛型算法到 Java 程序生成的过程.

可以预见,泛型程序因其可靠性和可重用性将会更加灵活的运用于软件工程特别是基于构件的软件开发中,本文的工作将为泛型构件的设计开发带来更多的便利.

4　参考文献

[1] McIlroy D. Mass produced software components [C]. Germisch: NATO Sci Committee,1969: 138-155.

[2] Li Yunlin, Novak G. Generation of geometric programs specified by diagrams [C]. New York: ACM Press,2011: 63-72.

[3] Xue Jinyun. PAR method and its supporting platform [R]. Macao: UNU-IIST,2006: 348.

[4] 徐文胜,薛锦云. 泛型编程扩展及其 Java 实现 [J]. 计算机工程与科学,2007,29(10): 89-93.

[5] Kleene S C. Representation of events in nerve nets and finite automata [C]. Princeton: Princeton University Press, 1956: 3-41.

[6] Xue Jinyun. Two new strategies for developing loop invariants and their applications [J]. Journal of Computer Science and Technology,1993,8(2):147-154.
[7] Xue Jinyun. A unified approach for developing efficient algorithm of programs [J]. Journal of Computer Science and Technology,1997,7(4):314-329.
[8] 石海鹤,石海鹏,薛锦云. 一种形式化开发非递归算法的方法 [J]. 计算机应用研究,2007,24(11):203-205
[9] 石海鹤,薛锦云. 基于 PAR 的算法形式化开发 [J]. 计算机学报,2009,32(5):982-991
[10] Wang Changjing,Xue Jinyun. Formal derivation of a high-trustworthy generic algorithmic program for solving a class of path problems [C]. Berlin,Heidelberg: Springer-Verlag,2009:27-39.
[11] 吴国凤,方钰. 基于 Java 语言中的泛型研究 [J]. 计算机技术与应用进展,2008:1043-1047
[12] Daniel Liang Y. Introduction to java programming [M]. 9th ed. Armstrong Atlantic State University,2012.
[13] Raul Marticorena,Carlos Lopez. Refactoring generics in Java: a case study on extract method [J]. IEEE,Computer Society,2010,38:1-10.
[14] Adam Kiezun,Frank Tip. Refactoring for parameterizing Java classes [C]. IEEE 29th International Conference on Software Engineering (ICSE07),2007:437-446.
[15] 孙斌. 面向对象、泛型程序设计与类型约束检查 [J]. 计算机学报,2004,27(11):1492-1504.
[16] 骆健. Apla-> Java 程序自动转换系统的研制 [D]. 南昌:江西师范大学,2002.

The Java-Based Novel Implementation for an Abstract Generic Mechanism Computer Engineering and Applications

TIAN Fang,SHI Haihe*,ZUO Zhengkang,WANG Changjing,XUE Jinyun
(Provincial Key Laborary High Performance Computing,Jiangxi Normal University,Nanchang Jiangxi 330022,China)

Abstract: Generic programming has emerged as a paradigm for the development of highly reusable and reliable software components,and highly abstract generic mechanism contributes to lowering the complexity of generic programming. Core ideas and programming techniques of an abstract generic mechanism are presented,corresponding Java mechanism is analyzed,and transformation rules between them are extracted,and further the automatic transformation from abstract generic program to Java program and Java program execution are implemented in our platform. Finally through a Kleen algorithm the real application of abstract generic mechanism and its Java program generation is illustrated. The novel implementation for this abstract generic mechanism based on Java reduces the design difficulty of algorithm component and improve its reliability,as well as gives an effective solution to the design and description of generic component.

Key words: abstract generic mechanism; Java language; reusability; algorithm component

(责任编辑:冉小晓)