软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(11):2639−2651 [doi: 10.3724/SP.J.1001.2011.03892]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向方面的计算误差处理技术:实例研究与评估*

崔展齐[1,2], 王林章[1,2+], 刘慧根[3], 李宣东[1,2]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)

[2](南京大学 计算机科学与技术系,江苏 南京 210093)

[3](南京大学 天文学系,江苏 南京 210093)

Computational Error Handling as Aspects: A Case Study and Evaluation

CUI Zhan-Qi[1,2], WANG Lin-Zhang[1,2+], LIU Hui-Gen[3], LI Xuan-Dong[1,2]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

[3](Department of Astronomy, Nanjing University, Nanjing 210093, China)

+ Corresponding author: E-mail: lzwang@nju.edu.cn, http://cs.nju.edu.cn/lzwang/

Cui ZQ, Wang LZ, Liu HG, Li XD. Computational error handling as aspects: A case study and evaluation. *Journal of Software*, 2011,22(11):2639−2651. http://www.jos.org.cn/1000-9825/3892.htm

Abstract: Traditional approaches tangle the error handling concerns with the primary functional code, which inevitably increase the degree of coupling and decreases the understandability and the maintainability of programs. This paper presents an empirical study of a real-world Satellite Orbit Forecasting system by refactoring error handling policies as methods of classes and aspects, respectively. A set of experiments are constructed to evaluate the original version with the two refactored versions. Based on the results of experiments from the refactoring process, it can be concluded that the modularity and maintainability of the program have been improved without a noticeable compromise in performance by encapsulating computational error handling polices as aspects.

Key words: aspect-oriented programming; refactoring; computational error handling; scientific computing

摘 要: 传统的实现方法通常把误差处理策略与程序的基本功能交织到一起,这会增加程序的耦合度,使得程序难以理解与维护.针对这一问题,提出一种面向方面的解决方案,即将程序的各种误差处理策略封装为方面.为评估该方法的有效性,将一个真实的卫星轨道测算系统中的误差处理策略分别封装为类中的方法和方面,并设计了一系列实验来评估原程序及分别采用两种方式重构后的程序在关注点分离度、耦合度、程序规模及运行时间上的差异.结果实验结果和重构过程中获得的经验可以得出,采用面向方面技术将误差处理功能封装为方面来实现能够有效提高程序的模块化程度和可维护性,并且不会引起程序性能的显著下降.

关键词: 面向方面程序设计;重构;计算误差处理;科学计算

中图法分类号: TP311 **文献标识码**: A

科学计算与理论和实验一样,是当今科学研究的三大分支之一.当数学问题不能求出精确的解析解时,只能

* 基金项目: 国家自然科学基金(60721002, 90818022, 61170066); 国家重点基础研究发展计划(973)(2009CB320702)
收稿时间: 2009-12-08; 定稿时间: 2010-05-14

通过计算及数值分析的方法求解.数值分析关注于通过计算的方式来求解科学技术研究中所遇到的复杂数学问题.计算机在数值分析中发挥了巨大的作用,随着计算机计算能力的不断增强,更多更复杂的数值分析方法被广泛地应用于生物信息学、计算化学、药物筛选、新型材料设计及航空航天等领域.在众多的应用领域中,如何合理地处理计算所引入的误差,始终是一个无法回避的问题.因此,如何采用有效的方法来控制计算误差,成为应用领域专家和科学计算程序开发人员共同关注的重要问题.

由于计算误差直接影响到计算结果的精确性和可靠性,误差处理策略的设计和实现问题是影响科学计算程序、尤其是安全关键系统质量的核心因素之一.科学计算程序中的每一步计算操作都可能会引入计算误差.传统的实现方法采用在可能引入误差的语句前后插入误差处理代码片段的方式来处理误差,这会导致实现同一误差处理策略的程序片段重复出现在程序的多个位置,与实现基本功能的代码交织在一起.这种现象的出现将会引起代码纠缠(code tangling)问题[1],导致程序难以理解与维护.同时,误差处理策略通常需要不断调整和优化,以适应不断变化的计算精度需求和计算能力限制.这类频繁的需求变动导致误差处理代码被不断地修改,使代码纠缠问题变得更为严重,从而增大了程序出现潜在缺陷的概率.

横切关注点是一类散布于系统中多个位置、与系统的多个功能模块交织在一起的特殊系统需求.面向方面的程序设计方法(aspect-oriented programming,简称 AOP)[2]引入了方面(aspect)这种新的程序结构,将横切关注点封装为独立的方面单元,并支持描述方面与程序中的基本结构单元(如类)之间的集成关系,使程序开发人员可以更方便地处理横切关注点.AOP 中方面的引入为横切关注点提供了一个更高效的解决方案,使程序结构变得更为清楚和易于理解.

通常情况下,同一类误差的处理策略是相似的,并且会横切多个基本功能模块,是典型的横切关注点.为了提高科学计算程序的模块化程度和可维护性,本文提出使用面向方面程序设计方法将与计算误差处理相关的关注点封装为方面.本文使用 AspectJ 来重构了一个真实的卫星轨道测算系统(satellite orbit forecasting,简称 SOF)中的误差处理功能.我们首先将该程序中与误差处理相关的代码进行两种方式的重构:一是将散布在程序中的误差处理代码重构为类中的方法,二是在其基础上将误差处理方法重构为方面.然后,将重构后的程序与原程序进行一系列的比较实验,以评估将面向方面技术应用于计算误差处理领域的可行性和有效性.

本文的主要贡献在于:

- 提出将面向方面技术引入科学计算领域,以解决计算误差处理所导致的代码纠缠问题;
- 针对一个真实的卫星轨道测算系统完成了实例研究,设计了一系列比较实验来评估所提出的方法.

本文第 1 节介绍 AOP 和计算误差的背景知识.第 2 节介绍使用方面来封装误差处理功能的基本方式.第 3 节讨论如何将 SOF 系统中的误差处理策略重构为方面.第 4 节评估在科学计算程序中使用 AOP 的可行性和有效性.第 5 节介绍相关的工作.第 6 节总结本文的工作并进行展望.

1 背景介绍

1.1 面向方面程序设计及AspectJ

横切关注点这类特殊系统需求的设计和实现方法一直是软件的开发过程中所需要关注的关键问题之一.传统的方法,如面向对象程序设计方法(object-oriented programming,简称 OOP),关注于纵向地分解和封装关注点,这类方法无法将散布于系统多个基本功能模块中的横切关注点封装为单独的程序模块.1997 年,Kiczales 提出了 AOP 的概念[2].AOP 不是对 OOP 的取代,而是作为 OOP 的一个补充,AOP 以一种更加模块化的方式来处理横切关注点,使程序开发人员可以将横切关注点分解和封装为单独的方面来处理,从而提高系统的模块化程度.这一机制使横切关注点的实现更加模块化.提高系统的模块化程度,可以使程序结构与人的思维方式更加接近,使代码更易于理解和维护,从而提高系统适应不断变化的需求的能力.近年来,随着 AspectJ[3],AspectC[4]及 AspectC++[5]等面向方面程序设计语言的提出与不断成熟,AOP 也得到了学术界和工业界越来越多的关注.

AspectJ 是目前使用得最为广泛的 AOP 语言,AJDT(AspectJ development tools: http://www.eclipse.org/ajdt/)是 Eclipse 平台下 AspectJ 的开发和编译环境.AspectJ 是对 Java 语言面向方面的扩展,AspectJ 程序经编译后生

成的类文件可以直接在 Java 虚拟机上执行.AspectJ 的扩展主要有:连接点(join point)、切入点(pointcut)、通知(advice)、类型间声明(introduction)和方面.连接点指程序执行过程中的事件,如方法的调用和执行、属性的读取和赋值、异常的执行等.切入点相当于过滤器,用于选择程序控制流中满足其描述的连接点,并屏蔽其他连接点.通知则是定义在其对应的切入点所需要执行的动作,通知可以在连接点之前、之后或周围执行.类型间声明用来为其他类定义属性或方法,也能用于声明类与其他类之间的继承关系.而方面则是一种类似于类的模块化单元,用于封装相关联的切入点、通知及类型间声明.

1.2 计算误差

使用科学计算的方法来处理数学问题会不可避免地引入误差.科学计算中的误差可以分为模型误差、观察误差、截断误差及舍入误差[6].首先,使用科学计算方法解决现实问题需要建立数学模型.它是通过对实际问题进行抽象和简化而得到的,因而所求出的解是近似的.抽象的数学模型与实际问题之间的这种差别称为模型误差.只有抽象、简化出的数学模型合理,才能使所求出的解具有较高的精度.其次,在数学模型中往往会有一些需要通过观测来获取的物理量,如温度、速度、压力等.通过观测设备测量这些物理量也会引入误差,这种误差称为观测误差.此外,当数学模型不能得到精确解时,通常要使用数值方法来求出近似解.近似解与精确解之间的误差称为方法误差或截断误差.最后,在使用计算机求解所建立的数学模型时,由于计算机用于存储数据的字长有限,原始数据在计算机上进行表示时会产生误差,计算机存储计算过程中产生中间结果时也可能由于近似舍入而产生误差,这种误差称为舍入误差.

在数值分析中,除了需要研究如何为现实问题建立合适的数学模型以外,还需要考虑如何使计算结果的误差满足计算精度的要求,这就是误差处理问题.数值分析方法通常只关注求解数学模型过程中所产生的截断误差,这也是本文所讨论的误差.

2 误差处理与方面

由于计算误差是使用数值分析方法求解数学问题的过程中不可避免的副产物,恰当地控制误差才能让计算结果在满足精度需求的同时不至于过多地影响性能.尽管科学计算程序通常都需要进行误差处理,但在不同的环境下对不同的目标所要求的计算精度是各不相同的.为了满足特定的计算精度需求,计算的参数甚至算法都需要做出相应的优化和调整.科学计算程序中误差处理问题的难点在于,误差处理本身是一个系统级的关注点,通常会分散在多个系统模块中.这导致与误差处理相关的代码散布于整个系统中,与实现基本功能的代码交织在一起.随之而来的问题是,当需要调整误差处理策略时,非常难以定位和修改,往往需要走查整个系统的代码来定位与该误差处理策略相关的代码,并分别对程序中多个位置的代码进行修改.因此,误差处理策略的传统实现方法不仅可能会引入潜在的缺陷,而且会为验证程序的正确性和将来的维护任务带来困难.常见的误差处理策略包括算法优化、误差估计及误差控制.

2.1 算法优化

实践中,科学计算问题通常是非线性的大规模复杂问题,而且往往对解空间有严格的限制.单个简单的算法常常会造成局部最小值或者解不收敛等问题,不能取得理想的结果.将简单的算法与其他优化手段相结合是一种降低误差的有效方法.有很多优化算法是解决一类计算问题的通用算法,可以将其封装为黑盒的方式,如常用的防止局部最小解的混沌算法和模拟退火算法等.这类优化算法与实际问题关系并不紧密,不应与实际问题交织在一起进行实现,应该将这类优化算法封装为方面来实现.图 1 的第 1 行~第 11 行描述了用迭代法解形式为 $x=\varphi(x)$的方程的算法,第 12 行~第 32 行是将该算法进行面向方面重构后的结果,第 12 行~第 16 行计算指定 x 时 $\varphi(x)$的值,第 17 行~第 32 行则是将原程序中误差处理相关的部分重构而成的方面.在该算法中,第 5 行、第 6 行实现的是使用 Aitken 法对迭代的过程进行加速,是对迭代法解方程的优化.在第 19 行~第 22 行中将其封装在 ErrorProcess 方面的 Aitken 通知中,该通知将在 SolveEquation 方法中的 14 行运行结束后自动调用.

```
1   double x0, x1, x2, e;
2   double SolveEquation(double x0){
3     //Input initial value x0, return solution x1
4     x1=φ(x0);
5     x2=φ(x1);
6     x1=x2−(x2−x1)^2/(x2−2x1+x0);
7     e=|x1−x0|;
8     if (e>ε)
9       SolveEquation(x1);
10    else return x1;
11  }
```

```
12  double x0, x1, x2, e;
13  double SolveEquation(double x0){
14    x1=φ(x0);
15    return x1;
16  }
17  public aspect ErrorProcess{   //Error handling policy aspect
18    pointcut Aitken execution SolveEquation(…) && set x1
19    after: Aitken(){   //Advice
20    x2=φ(x1);
21    x1=x2−(x2−x1)^2/(x2−2x1+x0);
22    }
23    pointcut Estimate execution Aitken(…)
24  after : Estimate(){   //Advice
25      e=|x1−x0|;
26    }
27    pointcut Control execution Estimate(…)
28  after: Control(){   //Advice
29      if (e>ε)
30        SolveEquation(x1);
31    }
32  }
```

Fig.1 Solve linear equation with iteration method

图 1 使用迭代法解线性方程

2.2 误差估计

数值分析中除了研究数学问题的算法之外,还要研究计算结果的误差是否满足精度需求,这就是误差估计问题.与数值分析方法求出的解是精确解的一个近似值相似,误差估计同样是计算精确误差的一个近似值.进行误差估计最重要的原因是需要对数值分析所求出的近似解的置信区间进行估计.尽管误差估计不可避免地会引入额外的计算开销,但在绝大多数科学计算程序中都需要进行误差估计,这是因为没有置信区间的解是没有意义的.此外,误差估计的结果还可以用于误差控制.误差估计一般都是在一种算法执行结束后、误差控制执行之前进行的,将为误差控制提供指导.误差估计的方法具有相似性,应该将误差估计的方法封装为方面来实现.在图 1 中,第 7 行为误差估计,即将本次迭代结果与上次迭代结果的差的绝对值作为迭代法的估计误差.在第 24 行~第 26 行中将其封装在 ErrorProcess 方面的 Estimate 通知中,将在 Aitken 通知执行结束后自动调用.

2.3 误差控制

在可接受的计算开销下,合适的参数和算法是使数值分析方法所求出的解满足所需计算精度的两个最重要的因素.由于计算目标等初始条件不同,使用固定的参数和算法来求解实际的数值分析问题是不合理的.比如,在一个积分程序中,待积分曲线的斜率在完成积分前是未知的,而斜率的不同又决定了应采用的积分算法和积分步长.在这种情况下,参数和算法的动态调整就显得尤为重要.误差控制一般都是在误差估计之后、基于误差估计的结果进行的.误差控制的方法具有相似性,应该将误差控制的方法封装为方面来实现.图 1 中,第 8 行、第 9 行为误差控制,即当估计误差大于精度要求ε时继续迭代,小于精度要求ε时结束迭代.在第 28 行~第 31 行中将其封装为在 ErrorProcess 方面的 Control 通知中,在 Estimate 通知执行结束后自动调用.

3 实例研究

轨道测算系统(SOF)是一个地面观测站使用的地球卫星轨道测算程序,用以预测卫星轨道,从而确定合适的卫星观测位置.通过输入卫星的初始坐标和相关的参数,SOF 将计算出该卫星后续一段时间内的瞬时轨道根数和平均轨道根数[7].原 SOF 系统采用 Java 程序设计语言开发.该程序共包含不含注释及空行的代码 2 006 行,其中 288 行代码与误差处理相关,这些代码分散在程序中的 15 处出现.该系统的核心模块是一组积分器,分别实现了 RKF4(5),RKF5(6),RK6(7)及 RKF7(8)[8]方法.这组积分器的功能是求解卫星轨道的微分方程,其形式为 $y'(t)=f(t,y(t))$,其中,y 是卫星轨道的坐标向量,t 代表时间参数.该方程的解为卫星在特定时间的坐标向量.

给定一段时间$[a,b]$和一个初始值 $y(a)=y_a$,求解此微分方程的方法是,将$[a,b]$分为 m 段($a=t_0<t_1<t_2<...<t_n<…<$

$t_m=b$),然后再逐步根据 y_{n-1} 迭代求出 $y(t_n)$.为求解 y_n,数值积分器取步长 h_{n-1},则 $t_n=t_{n-1}+h_{n-1}$,所求出的 $y(t_n)$即为 y_n 的近似解.计算不同的卫星轨道或计算结果的用途不同时,应采用的积分公式和步长是不一样的,误差的处理策略也需要随之灵活改变,即参数和积分器根据计算精度和性能需求的不同而做出相应的调整.

3.1 轨道测算系统中的误差处理

为了提高 SOF 系统的模块化程度和可维护性,我们从中抽取出 4 个与误差处理策略相关的关注点,分别是下山法、全局误差估计、步长调整和积分器调整的算法.为了研究使用方面来封装误差处理策略的有效性,我们将 4 个误差处理关注点先后进行了两种方式的重构:一是将散布的误差处理相关代码封装为类中的方法,二是在前一种方式的基础上将误差处理方法及其调用封装为方面.图 2 描述了两种重构方式.重构为类中的方法是将散布在系统中重复的代码封装为单独的方法体,再在原来散布代码出现的地方增加方法调用.重构为方面则是在前一种重构方式的基础上,进一步将方法体封装为通知,并根据方法调用的连接点编写切入点,从而将横切关注点封装为单独的方面模块.

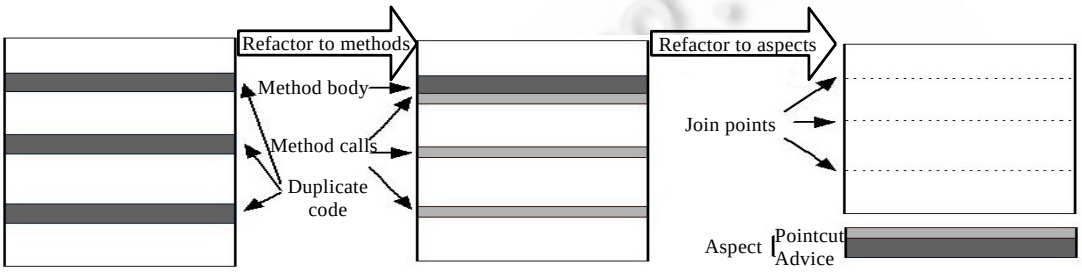


Fig.2 Comparison of two different kinds of refactoring

图 2 两种重构方式的比较

图 3 为 SOF 系统原 Java 程序和采用两种方式进行重构后的程序中,与 4 个误差处理策略相关的代码在程序中的分布情况.在将误差处理关注点重构为方法的程序中,程序的规模有所下降,但与主程序的耦合和代码纠缠问题并没有得到彻底解决.在将误差处理关注点重构为方面的程序中,上述误差处理策略被分别封装成了独立的方面,方面在程序运行阶段会自动地执行以对误差进行相应的优化、估计和控制操作.这一重构方式解决了代码纠缠问题.在 AspectJ 的支持下,维护阶段如需对误差控制策略进行调整,则只需对独立的方面进行相应修改即可,而不影响程序的基本功能模块.

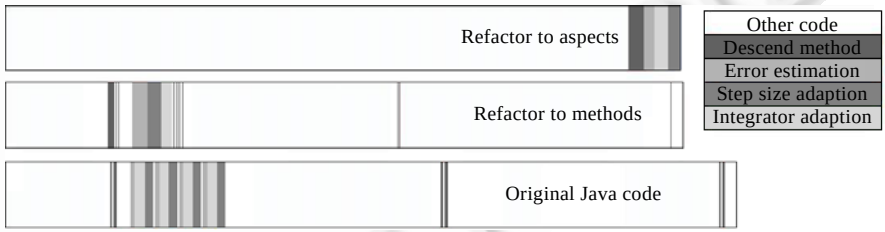


Fig.3 Scattering of error handling code (scaling by lines of code)

图 3 误差处理代码分布(宽度按代码行数所占比例绘制)

3.1.1 下山法

在本实例研究中,用来求解卫星、月球及太阳星历[7]的公式是开普勒方程,其形式为 $E-M-e\times\sin E=0$.在已知 e 和 M 的前提下,牛顿法被用于求解方程中的未知变量 E.牛顿法解方程的主要过程可以表示为:迭代计算 $x_{n+1}=x_n-f(x_n)/f'(x_n)$,直到 $f(x_n)$小于指定的阈值 ε.在该公式中,$f(x_n)$表示方程左部在 x_n 的值,$f'(x_n)$表示方程左部的导数在 x_n 的值.

牛顿法由于具有收敛快的特点,是常用的求解微分方程的方法.但牛顿法也有局部收敛性的缺陷.也就是说,只有当输入的初始值与精确解比较接近时才能保证收敛.作为一个强化手段,下山法用来保证目标函数每次迭代求出的值能够稳定地逼近精确解.当 $f(x_{n+1})>f(x_n)$时,下山法将会对 x_{n+1} 进行调整:$x_{n+1}=\lambda\times x_{n+1}+(1-\lambda)\times x_n$.结合牛顿法和下山法能保证算法能够在一个更大的初始输入域内收敛,其中,下山法起到了优化牛顿法求解微分方程的作用.将下山法与求解微分方程的基本算法分离,能让程序更易于理解,并使得封装后的下山法能够作为一个通用的优化算法来复用.图 4 描述了将下山法先封装为方法,再封装为方面的过程.

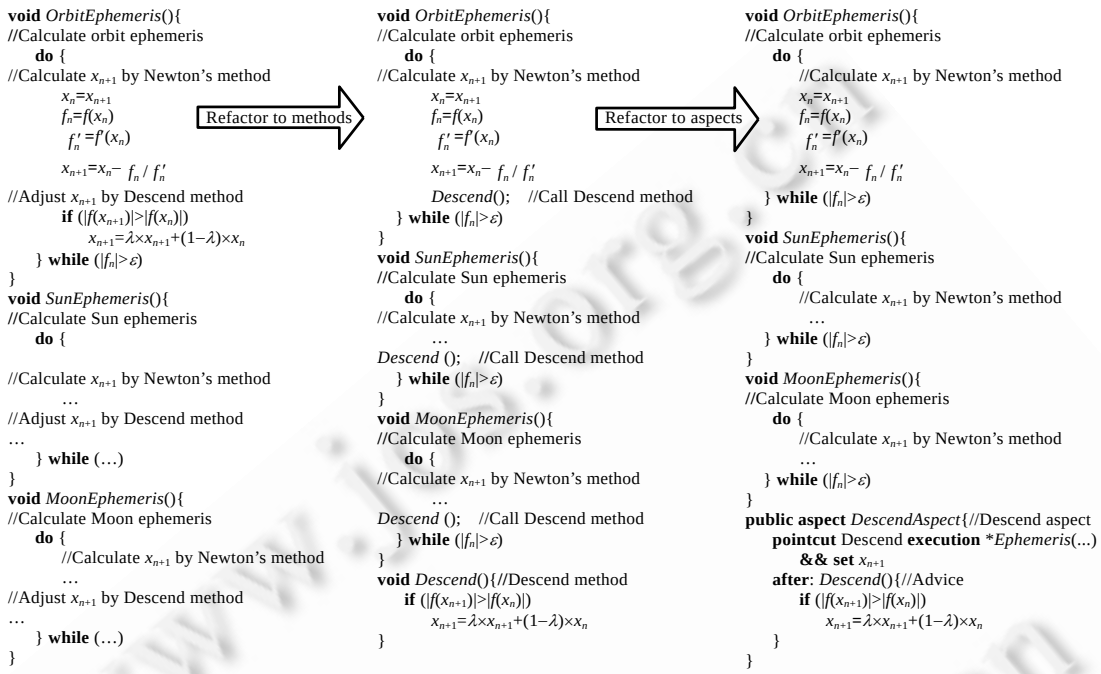


Fig.4 Encapsulating descend method as an aspect

图 4 将下山法封装为方面

3.1.2 全局误差估计

对积分器误差的估计包括全局误差估计和局部误差估计.局部误差是指单步积分所产生的误差,全局误差是指整个积分过程的累计误差.在本实例研究中使用嵌套的 Runge-Kutta 公式[8]来求解卫星轨道的偏微分方程.其基本思想是,构造成对的自包含 Runge-Kutta 公式,在使用 p 阶公式计算 $f(x_n)$时,同时求出 p+1 阶公式的解 $f^*(x_n)$,则$|f^*(x_n)-f(x_n)|$即为对 $f(x_n)$的局部误差估计.通过嵌套的 Runge-Kutta 公式组,在计算出 p 阶公式的解后,只需要少量的额外计算开销即可求出 p+1 阶公式的解.

一种常用的全局误差估计方法是使用两组不同阶的积分公式对同一积分区间进行积分.使用 p 阶公式计算出近似解 $f(b)$,同时使用 p^*阶公式计算出另一个近似解 $f^*(b)$,其中,$p^*>p$,则$|f^*(b)-f(b)|$即为对 $f(b)$的全局误差估计.另一种估计全局误差的常用方法是,使用两个不同的步长对同一积分区间使用同一组积分公式进行积分.使用步长 h 计算出近似解 $f(b)$,同时使用另一步长 h^*计算出另一个估计值 $f^*(b)$,其中,$h^*<h$,则$|f^*(b)-f(b)|$即为 $f(b)$的全局误差估计.图 5 描述了将估计全局误差的算法先封装为方法,再封装为方面的过程.

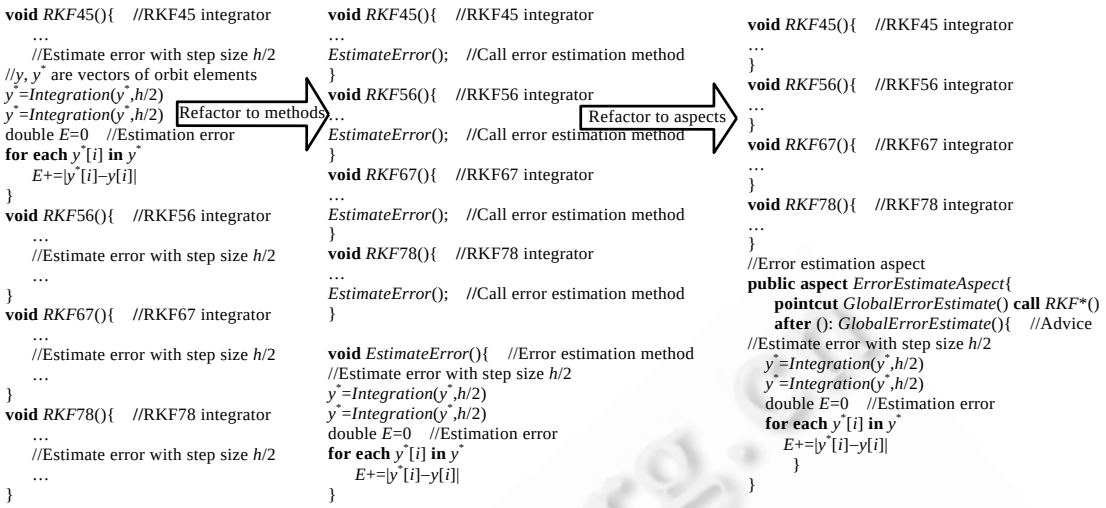


Fig.5 Encapsulating global error estimation as an aspect

图 5 全局误差估计封装为方面

3.1.3 误差控制

在给定的积分区间内,待积分曲线的曲率可能会发生改变,采用固定的积分器和固定的步长是不合适的.一方面,如果为满足积分曲线斜率变化最快部分的要求而采用小步长或高阶公式积分,将必然会引入过大的计算开销;另一方面,如果考虑到性能方面的需求而增大积分步长或者采用低阶积分公式,则可能会导致计算结果达不到所要求的精度.平衡计算精度和计算开销这对矛盾的常用方法有两种:一种是通过调整积分步长来控制误差,另一种是通过调整积分器来控制误差.这两种方法都可以在平衡精度和性能的同时使积分过程保持稳定.

(1) 步长调整

影响步长选择的因素主要有两个:一是计算精度,二是累积误差.如果积分步长过大,则积分过程可能会不稳定;而积分步长过小,则完成积分区间所需要迭代的积分次数将会增多,这会导致更大的累积误差和更多的计算开销.因此,在求解微分方程时调整步长是一种有效地控制误差的手段.特别是在积分曲线的斜率变化较大的情况下,步长调整可以使积分过程更加稳定并将误差控制在可接受的范围之内.尽管误差的估计和步长调整会引入额外的计算开销,但是合适的积分步长能够让求解微分方程的过程更加高效.步长调整是一种独立的误差控制策略,不应该与程序的基本功能代码交织在一起.

(2) 积分器调整

使用低阶的积分器积分会加快计算速度但会降低计算精度;反之,使用高阶的积分器能够提高计算精度但会引入更大的计算开销.选择合适阶的积分器需要在性能和精度之间进行平衡.事实上,当积分曲线在积分区间内曲率变化较大时,如果采用高阶公式积分,则会导致过多的计算开销;如果采用低阶公式积分,计算的精度则可能达不到要求.根据积分曲线的曲率动态调整积分过程中所采用的积分器,能够在保持积分过程稳定的同时将误差控制在可接受的范围内.积分器调整是另一种独立的误差控制策略,不应该与程序的基本功能代码交织在一起.

图 6 描述了将步长调整这一误差控制策略先封装为方法,再封装为方面的过程.

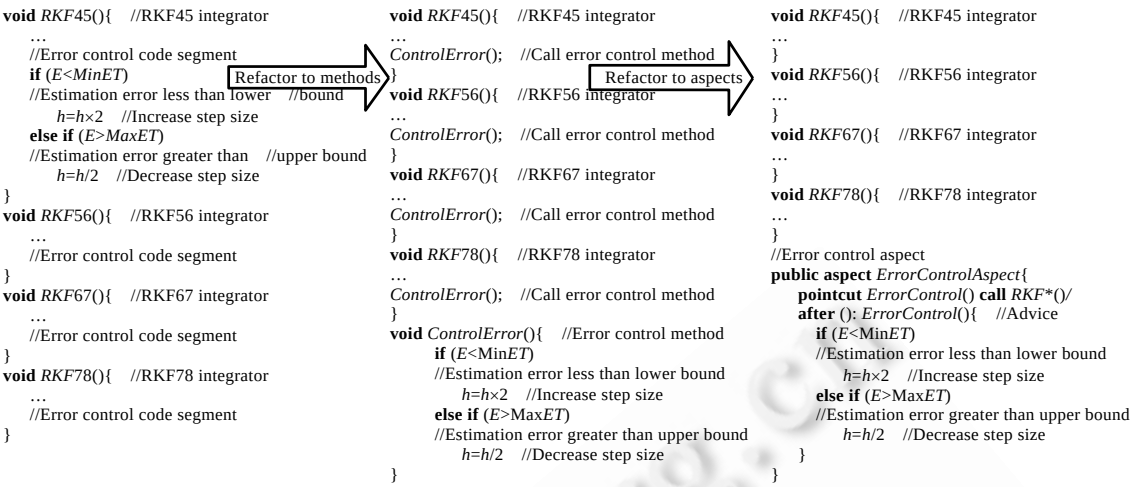


Fig.6 Encapsulating error control polices as an aspect

图 6 将误差控制策略封装为方面

4 实验设计及评估

本实例研究中所采用的软件环境是 Window Vista Business,JDK 1.6.0,AspectJ 1.5.2 及 Eclipse 3.3.2,硬件平台是 Q6600 2.40 GHz 处理器和 4GB 内存的计算机.

我们分别使用 3 种不同的方法来实现 4 个与误差处理相关的关注点:原程序采用标准的 Java 来实现,其误差处理相关的代码与程序的基本功能实现代码交织在一起;一个重构版本是将误差处理代码抽取并封装为类中的方法来实现;另一个重构版本是使用 AspectJ 将误差处理相关的代码封装为方面来实现.3 个版本的程序实现方法不同,但功能完全相同.

为了合理地评估将 AOP 用于实现误差处理策略的有效性,我们对 3 个版本的程序从关注点分离度、耦合度等方面来进行比较和评估.Walker 等人[9]提出采用 Debug 和修改程序所耗费时间的不同来评估使用 AOP 的有效性.但是这类度量维度较为主观,与维护人员在 AOP 和其他程序开发方法上所具备的经验有着密切的关系.为了更为客观地评估 AOP 系统,Garcia 等人[10]提出了一组易于量化的度量维度.基于 SOF 系统的特点,我们将文献[10]中的度量维度进行了调整和修改(见表 1),通过实现误差处理策略的操作数及变量数、误差处理相关操作体所调用的操作数及外部变量数、实现误差处理策略的代码行数等度量维度,分别比较不同版本程序间误差处理关注点的分离度、耦合度及程序规模的差异,从而评估程序的模块化程度.另外,我们通过比较不同版本程序完成同样的维护任务所需要修改的代码行数和操作数的差异来评估程序的可维护性.

Table 1 Evaluation metrics

表 1 度量维度

Attributes	Metrics	Definitions
Separation of concerns	Implementation operations	The number of operations (methods and advices) which are used to implement error handling polices.
	Implementation variables	The number of variables which are used to implement error handling polices.
Coupling	Dependent operations	The number of operations which are invoked in error handling operations.
	Dependent variables	The number of external variables which are used in error handling operations.
Size	Lines of code	The lines of code which is related to error handling polices.
Maintainability	Lines of modified code	The lines of code which is modified to complete a maintenance task.
	Modified operations	The number of operations which are modified to complete a maintenance task.

在图 7~图 9 中,我们对 3 个版本的程序在实现同样的误差处理策略时,程序的模块化程度进行了比较.其中,图 7 对关注点的分离度进行了比较,AspectJ 版本中实现误差处理相关策略所使用到的操作数和变量数都更少,

因此关注点分离度更高.从图 8 中可以看出,AspectJ 版本程序在实现下山法和误差估计时所依赖的操作数比原 Java 程序多,但不多于将下山法和误差估计封装为独立方法后的程序,而在实现步长调整和积分器调整时所依赖的操作数比另外两个版本的程序少.这是因为下山法和误差估计策略较为简单,在原 Java 程序中并没有调用其他方法,而步长调整和积分器调整较为复杂,需要调用多个方法.在误差处理策略较为复杂、相互调用的操作数较多时,AspectJ 版本的程序更能有效地降低耦合度.图 9 对 3 个版本程序的规模进行了比较,除下山法外,AspectJ 版本的程序实现误差处理策略所需的代码行数比其他两个版本的程序更少.上述实验结果表明,AOP 在实现误差处理相关问题时具备了更好的模块性.

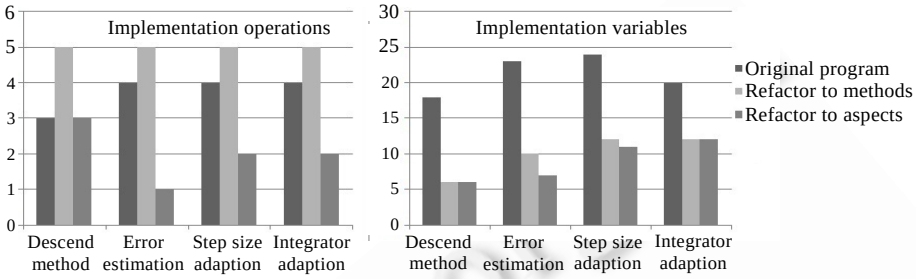


Fig.7 Comparison on separation of concern

图 7 关注点分离度比较

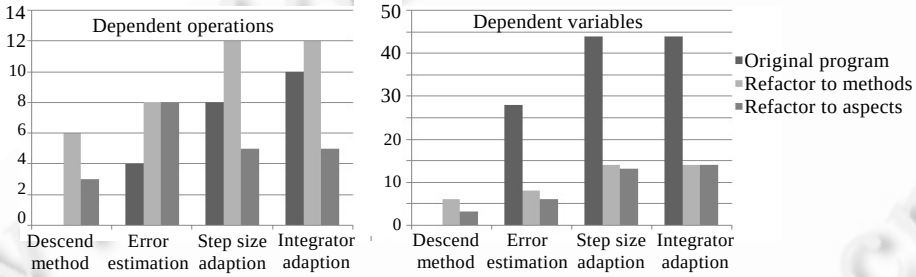


Fig.8 Comparison on coupling

图 8 耦合度比较

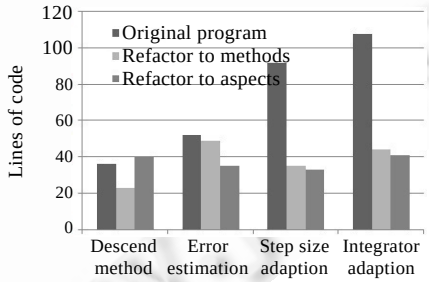


Fig.9 Comparison on program size

图 9 程序规模比较

另外,我们还比较了系统编码完成之后,分别在 3 个版本的 SOF 系统中修改误差处理策略所需要的维护开销.从图 10 中的结果可以看出,在执行同样的维护任务(修改误差处理策略)时,在 AspectJ 版本的程序中需要修改的代码行数和操作数都比原程序要少.虽然在部分情况下,AspectJ 版本与封装为方法的版本所需要修改的代

码行数和操作数相同,但 AspectJ 版本中所影响到的操作都是封装在独立的方面中的,更便于查找和定位.这一实验结果表明,将科学计算程序中的误差处理策略作为方面来实现,可以有效地提高系统的可维护性.

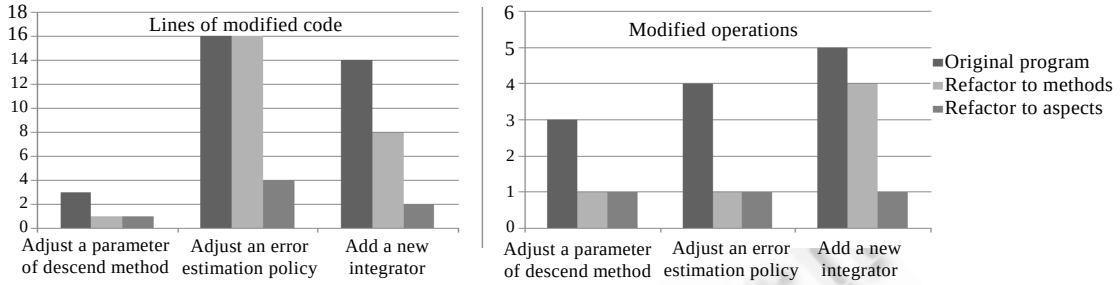


Fig.10 Comparison on maintenance cost

图 10 维护成本比较

最后,我们比较了 3 个版本的系统在分别引入 4 个与误差处理相关的关注点后,程序的运行时间(在相同输入条件下,执行相同功能 100 次的平均时间)上的差异.如图 11 所示,3 个版本的程序在性能上并没有显著的差异.由此可见,使用 AspectJ 来实现误差处理策略并没有引入过多的性能开销.

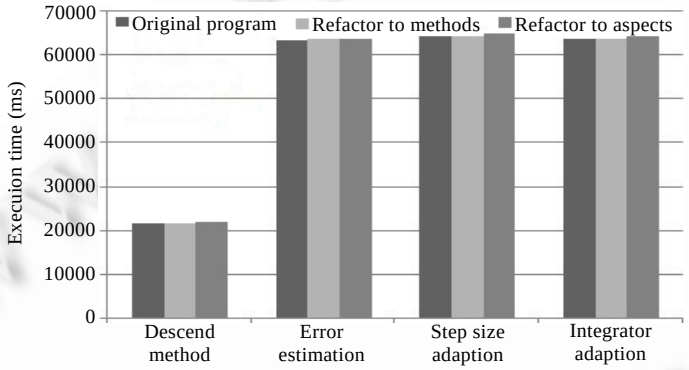


Fig.11 Comparison on execution time

图 11 运行时间比较

上述实验结果表明,采用 AspecJ 来实现 SOF 系统,可以在提高系统模块化程度的同时降低系统的维护成本.面向方面技术是一种适合用于实现误差处理策略的方法.将误差处理功能作为方面来实现具有如下的优势:首先,采用方面来封装误差处理策略能够使代码更简洁,条理更加清晰,更易于理解;其次,AspectJ 提供了分离横切关注点的机制,让程序员可以先将注意力集中在程序的主要功能上,在实现系统的基本功能后,再让误差分析专家将误差处理策略作为方面单独实现.这种实现方法不会影响到系统的基本功能模块,而且还能让误差控制策略更易于修改,也使系统的模块化程度和代码的可复用性都得到了较大幅度的提高.同时,实验结果还表明,AspectJ 版本的 SOF 系统在性能上并没有显著的下降.此外,AspectJ 还提供了 Exclude/Include 方面的机制,可以将实现误差优化、估计和控制的方面自由装配后再进行编译,从而为程序开发人员的 Debug 过程带来便利.

此外,在实验过程中,我们还发现了几处现有的 AspectJ 在开发科学计算程序时的不足:

- 为 set 类型切入点提供获取变量被赋值前值的支持.在 AspectJ 中,set 类切入点对应的 before 型通知和 after 型通知获取的都是变量被赋值后的值.为了获取变量在赋值操作前的值,我们只能通过显式地使用 Java 的反射机制来获取.
- 语句级别的切入点.由于现有的 AspectJ 缺乏对语句级切入点的支持,只能在拟添加通知的语句前后插

入方法体为空的 Hook 方法[11]调用,再通过捕获 Hook 方法调用来间接捕获目标语句.如果能提供语句级切入点的支持机制,就可以删除人为引入的 Hook 方法,使程序更加简洁和易于理解.

- 局部变量访问机制.一般而言,科学计算程序在很大程度上是数据密集型程序,监控其中一些变量的赋值和操作显得尤为重要.在当前版本的 AspectJ 中,方法内的局部变量在方面中是不可见的.若能提供局部变量的访问机制,则能为使用 AspectJ 开发科学计算程序带来更多的便利.

5 相关工作

重构是对程序内部结构的一种调整,其目的是在不改变程序外部行为的前提下简化程序结构,以提高其可理解性、降低其维护成本[12].面向方面的重构帮助重新组织与横切关注点相关的代码,从而进一步提高程序的模块化程度,并消除代码分散和代码纠缠现象.面向方面的重构为传统的重构技术提供了附加手段.面向方面的重构与传统重构技术的不同之处在于它引入了 AOP 新增的结构模块——方面,可以方便地将异常处理、并发控制等策略封装为单独的方面.面向方面的重构方式按其重构的关注对象和关注点的不同可以分为 3 类[13]:方面感知的面向对象重构(aspect-aware OO refactoring)、对面向方面程序结构块的重构(refactoring for AOP constructs)和对横切关注点的面向方面化重构(refactoring of crosscutting concerns).本文所采用的是对横切关注点进行面向方面化重构,即采用方面来代替原程序中非模块化的实现方式.

随着面向方面的软件开发(aspect-oriented software development,简称 AOSD)领域研究工作的逐渐深入,AOSD 的相关研究工作已经从学术理论研究开始向大型系统的应用研究发展.为评估 AOP 实现系统不同类型的横切关注点的有效性,研究人员在多个领域进行了大量的实例研究和分析,如安全[14,15]、数据持久化存储[16]、竞争检测[17]及多线程同步[18]等.国内研究人员也对将 AOP 应用于各种软件系统中的可行性开展了一系列的工作.陈向群等人从构件重构、系统演化与设计及系统安全、性能检测与容错这 3 个方面分析了在操作系统中应用 AOP 思想的可行性和存在的问题,并对面向方面操作系统研究的前景进行了展望,认为 AOSD 的研究有可能对未来操作系统的发展产生重大影响[19].梅宏等人在基于构件、面向体系结构的软件开发方法(architecture based component composition,简称 ABC)基础之上,提出了通过构件运行支撑平台的支持,在运行时刻动态地植入通知的方法——ABC-S^{2}C[20].该方法使用方面对横切关注点进行建模和封装,并将连接子结构化和实体化,通过连接子将各个方面和构件关联在一起,由连接子在运行时刻分析构件的服务请求,动态调用对应的通知.该方法为构件化软件提供了模块化处理横切关注点的系统支持技术及机制.

在大多数的实例研究中,AOP 均表现出了与传统软件开发方法相比的优势.但 AOP 并不适合于所有类型的程序,Kästner 等人为了提高嵌入式系统 Berkeley DB 的可配置性,将其中 38 个功能点重构为方面,实验数据却表明,AspectJ 并不适合用于重构该系统的功能点[11].这一实验结果说明,在应用 AOP 到特定类型的软件系统之前,进行充分的实例研究是很有必要的.

Filho 等人在文献[21]中使用 AOP 进行的错误处理(error handling)是指程序中的异常(exception),而本文所讨论的是计算误差处理(computational error handling)问题,即误差的优化、估计及控制问题.Harbulot 等人将 AOP 引入了科学计算领域,但该项研究关注于使用方面封装科学计算程序中的并发控制策略[22,23].而本文是采用 AOP 解决科学计算程序中的另一类横切关注点,即误差处理问题.使用数值分析方法解决数学问题将不可避免地引入误差,数值分析解要达到一定的置信度,合理地处理误差是不可或缺的手段.Shampine 等人在文献[24]中提出了求解带初值的常微分方程时的误差估计和控制算法.在其实现方法中,误差控制策略与求解常微分方程的算法交织在一起,这将给随后的修改和维护带来问题.

6 结 论

在科学计算程序设计实践中,传统的误差处理方法将引起代码纠缠现象,导致程序难于被理解和维护.针对这一问题,本文提出将程序中与误差处理相关的代码抽取出来并封装成方面.为评估该方法的有效性,我们在卫星轨道测算系统 SOF 上进行了实例研究,将该系统中与误差控制相关的分散代码封装为 4 个方面,并对程序的

关注点分离度、耦合度、程序规模及运行时间等方面进行了比较和评估.实验结果表明,采用 AspectJ 重构的程序在性能没有明显损失的情况下,有效地提高了系统的模块化程度和可维护性.

在这个实例研究的基础上,在接下来的工作中,我们将尝试使用面向方面技术来解决科学计算程序中更为底层的误差处理问题.另外,我们还计划开发自动识别和重构误差处理横切关注点的相关支撑工具.

References:

[1] Lopes CIV. D: A language framework for distributed programming [Ph.D. Thesis]. Boston: Northeastern University, 1997.

[2] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes CV, Loingtier JM, Irwin J. Aspect-Oriented programming. In: Aksit M, Matsuoka S, eds. Proc. of the 11th European Conf. on Object-Oriented Programming. Berlin: Springer-Verlag, 1997. 220−242. [doi: 10.1007/BFb0053381]

[3] Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold WG. An overview of AspectJ. In: Knudsen JL, ed. Proc. of the 15th European Conf. on Object-Oriented Programming. Berlin: Springer-Verlag, 2001. 327−353. [doi: 10.1007/3-540-45337-7_18]

[4] Coady Y, Kiczales G, Feeley M, Smolyn G. Using AspectC to improve the modularity of path-specific customization in operating system code. In: Proc. of the 8th European Software Engineering Conf. New York: ACM Press, 2001. 88−98. [doi: 10.1145/503209.503223]

[5] Spinczyk O, Lohmann D, Urban M. Advances in AOP with AspectC++. In: Fujita H, Mejri M, eds. Proc of the 4th Int'l Conf. on New Trends in Software Methodologies Tools and Techniques. Amsterdam: IOS Press, 2005. 33−53.

[6] Li QY, Wang NC, Yi DY. Numerical Analysis. 4th ed., Beijing: Tsinghua University Press, 2001 (in Chinese).

[7] Liu L. Orbit Dynamics for Artificial Earth's Satellites. Beijing: Higher Education Press, 1992 (in Chinese).

[8] Fehlberg E. Classical 5th-, 6th-, 7th-, and 8th-order runge-kutta formulas with stepsize control. Technical Report, NASA-TR-R-287, NASA, 1968.

[9] Walker RJ, Baniassad ELA, Murphy GC. An initial assessment of aspect-oriented programming. In: Proc. of the 21st Int'l Conf. on Software Engineering. New York: ACM Press, 1999. 120−130. [doi: 10.1145/302405.302458]

[10] Garcia A, Sant'Anna C, Figueiredo E, Kulesza U, Lucena C, Staa AV. Modularizing design patterns with aspects: A quantitative study. In: Proc. of the Trans. on Aspect-Oriented Software Development I. Berlin: Springer-Verlag, 2006. 36−74. [doi: 10.1007/11687061_2]

[11] Kästner C, Apel S, Batory D. A case study implementing features using AspectJ. In: Proc. of the 11th Software Product Line Conf. Los Alamitos: IEEE Computer Society Press, 2007. 223−232. [doi: 10.1109/SPLINE.2007.12]

[12] Fowler M, Brant J, Opdyke W, Roberts D. Refactoring: Improving the Design of Existing Code. Reading: Addison-Wesley Publishing Company, 1999.

[13] Hannemann J. Aspect-Oriented refactoring: Classification and challenges. In: Proc. of the AOSD Workshop on Linking Aspect Technology and Evolution. 2006.

[14] Viega J, Bloch JT, Chandra P. Applying aspect-oriented programming to security. Cutter IT Journal, 2001,14(2):31−39.

[15] Mourad A, Laverdière MA, Debbabi M. An aspect-oriented approach for the systematic security hardening of code. Computers & Security, 2008,27(3-4):101−114. [doi: 10.1016/j.cose.2008.04.003]

[16] Soares S, Laureano E, Borba P. Implementing distribution and persistence aspects with AspectJ. In: Proc. of the 17th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications. New York: ACM Press, 2002. 174−190. [doi: 10.1145/582419.582437]

[17] Bodden E, Havelund K. Aspect-Oriented race detection in Java. IEEE Trans. on Software Engineering, 2010,36(4):509−527. [doi: 10.1109/TSE.2010.25]

[18] Raje RR, Zhong M, Wang TY. Case study: A distributed concurrent system with AspectJ. ACM SIGAPP Applied Computing Review, 2001,9(2):17−23. [doi: 10.1145/512000.512004]

[19] Chen XQ, Yang FQ. Research on aspect oriented operating systems. Journal of Software, 2006,17(3):620−627 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/620.htm [doi: 10.1360/jos170620]

[20] Mei H, Cao DG. ABC-S^2C: Enabling separation of crosscutting concerns in component-based software development. Chinese Journal of Computers, 2005,28(12):2036−2044 (in Chinese with English abstract). [doi: CNKI:SUN:JSJX.0.2005-12-010]

[21] Filho FC, Garcia A, Rubira CMF. Error handling as an aspect. In: Proc. of the AOSD Workshop on Best Practices in Applying Aspect-Oriented Software Development. New York: ACM Press, 2007. [doi: 10.1145/1229485.1229486]

[22] Harbulot B, Gurd JR. Using AspectJ to separate concerns in parallel scientific Java code. In: Proc. of the 3rd Aspect-Oriented Software Development Conf. New York: ACM Press, 2004. 122−131. [doi: 10.1145/976270.976286]

[23] Xi C, Harbulot B, Gurd JR. Aspect-Oriented support for synchronization in parallel computing. In: Proc. of the AOSD Workshop on Linking Aspect Technology and Evolution. 2009. 1−5. [doi: 10.1145/1509847.1509848]

[24] Shampine LF. Error estimation and control for ODEs. Journal of Scientific Computing, 2005,25(1):3−16. [doi: 10.1007/BF02728979]

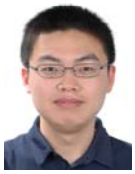
附中文参考文献:

[6] 李庆扬,王能超,易大义.数值分析.第 4 版,北京:清华大学出版社,2001.

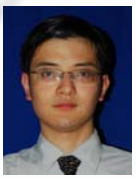
[7] 刘林.人造地球卫星轨道力学.北京:高等教育出版社,1992.

[19] 陈向群,杨芙清.面向 Aspect 的操作系统研究.软件学报,2006,17(3):620−627. http://www.jos.org.cn/1000-9825/17/620.htm [doi: 10.1360/jos170620]

[20] 梅宏,曹东刚.ABC-S^{2}C:一种面向贯穿特性的构件化软件关注点分离技术.计算机学报,2005,28(12):2036−2044.



崔展齐(1984－),男,贵州金沙人,博士,主要研究领域为软件工程,软件测试.



刘慧根(1984－),男,博士生,主要研究领域为太阳系动力学,地外行星形成.

王林章(1973－),男,博士,副教授,CCF 高级会员,主要研究领域为软件工程,软件测试.

李宣东(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,形式化方法,软件建模与分析,软件测试与验证.