程序设计 软件容错 软件可靠性

计算机科学1996 Vol. 23№. 1

66-68

使用防卫式程序设计实现软件容错*)

万剑怡 薛锦云

(江西师范大学计算机系 南昌 330027)

TP311.11

摘 要 Defensive programming is a method to realize software fault tolerance that needn't to use the N-version programming and recovery block technique. In this paper, we will analyze the characteristics, the functions and the applied ranges of defensive programming. We will also study its basic idea, explain its usage by examples, and compare it with traditional fault tolerance techniques.

关键词 Software fault tolerance, Defensive programming, Exception handling.

软件避错是提高软件可靠性的主要方法之一，它包含程序检验、测试、正确性证明等技术。然而，随着软件规模越来越大，结构越来越复杂，以及软件避错技术自身存在的复杂性和局限性，软件避错已远远不能保证软件的可靠性；另一方面，随着计算机的应用进入一些高可靠性要求的领域，对软件可靠性的要求也越来越高，因而软件容错成为提高软件可靠性的重要手段。软件容错的基本思想是从硬件容错中引伸而来，利用软件设计的冗余和多样化来达到掩蔽错误的影响，提高软件可靠性的目的。软件容错的基本技术有多版本程序设计技术和恢复块技术，它们都是基于设计冗余的思想，这给程序设计者和处理机都增加了许多工作，而且它们的结构本身又带来了一些问题和困难，如多版本程序设计结构中的相关性错误问题、恢复块结构中接受测试的设计等。Ian Sommervice 在文[1]中介绍了一种新的实现软件容错的方法——防卫式程序设计(Defensive Programming, DP)。本文研究了 DP 的基本思想，并通过实例说明使用DP 实现软件容错的方法，分析了它的特点、功能和适用范围，并和传统的容错技术作了比较。

一、防卫式程序设计的方法

1. DP 基本思想

防卫式程序设计是一种不采用任何传统容错技术就能实现软件容错的方法。对于程序中存在的错误和不一致性，防卫式程序设计的基本思想是通过在程序中包含错误检查代码和错误恢复代码，使得一旦错误发生，程序能够撤消错误的状态，恢复到一个已知的正确状态中去。

2. DP 的容错措施

作为一种软件容错的方法，和其它方法一样，在防卫式程序设计的容错措施中，也应包含错误检测、破坏估计和错误恢复几个方面。

(1)错误检测——插入断言法 在程序中插入状态断言，所谓状态断言就是包含状态变量的逻辑谓词，这些断言可插入到一些重要的赋值语句之前，使得那些可能导致错误的赋值在变量状态发生变化之前被检测出来。如：实型变量 Grade 表示一个学生的成绩，C 为一个实型整型变量，那么，赋值语句 Grade:=C 之前可插入断言{C>=0.0 and C<=100.0}，以保证对 Grade 赋值的正确性。

插入断言法尤其适用于抽象数据类型，因为在抽象数据类型中，断言检查代码只需定义一次，就可对该类型变量的操作进行检查。如：

```
Package Posive_even is
  type numb is limited private;
  procedure assign(A:in out numb;B:natural;
      state_error:in out boolean);
  function eval(A:numb)return natural;
  function eq(A,B:numb)return boolean;
Private
  type numb is new natural;
end Positive_even;
```

以上是一个正偶数抽象数据类型的说明，assign 是一个对该类型变量进行赋值的过程，在该过程中，可以采用插入断言{B mod 2=0}来保证对变量 A 的赋值为正偶数，该过程如下所示：

```
Procedure assign(A:in out numb;B:natural;
```

*)国家863发展计划，国家自然科学基金和国家军用共性软件发展计划资助项目，万剑怡 硕士研究生，研究方向：软件重用技术与容错计算，薛锦云 教授，研究领域：程序设计方法学，软件工程与容错计算。

```
    state_error;in out boolean)is
Begin
  if B mod 2≠0 then state_error:=true;
  else state_error:=false;A:=numb(B);
  end if;
End assign;
```

但在程序中普遍采用状态断言检查将占用大量时空，折中的办法是在重要操作处进行检查，发现非法状态，再进行破坏估计和错误恢复。

(2)破坏估计——破坏指示器　插入断言法是试图在变量状态改变之前对可能引起错误的操作予以避免，而破坏估计的任务是在变量可能已经遭到破坏的情况下，判断破坏是否已发生，以及状态空间的哪些部分受到了错误的影响。在抽象数据类型中，DP 的方法是设置一个破坏指示器，通过询问破坏指示器来估计破坏发生的情况。

下面是一个设置了破坏指示器的抽象数据类型的说明：

```
Package Checked_array is
  type elem is range 1..64;
  type index is natural range 1..6;
  type T is private;
  procedure check(A:in out T);
  function is_damaged(A:T)return boolean;
  function eval(A:T;i:index)return elem;
  function assign(A:T;i:index;e:elem)return T;
Private
  type short_array is array(index)of elem;
  type T is record
    S:short_array;
    damaged:boolean;
  end record;
End Checked_array;
```

在类型 T 中含有一个分量 damaged，用以标识 T 中的数组 S 是否遭到了破坏。check 是一个对 A 进行检查并设置破坏指示器的过程，is_damaged 为对破坏指示器的查询过程。假定数组 S 中的元素要求为偶数，则 check 和 is_damaged 可作如下设计：

```
procedure check(A:in out T)is
  begin
    for i in 1..6 loop
    if A.S(i)mod 2≠0 then A.demaged:=true;
    end if;
   end loop;
  end check;
function is_damaged(A:T)return boolean is
  begin
    return A.damaged;
end is_damaged;
```

如果 S 中有元素被赋予了奇数值，则 check 可以检查出这种不符合要求的值的存在，并对破坏指示器加以设置，通过 is_damaged 对破坏指示器的询问即可了解破坏的情况。

(3)错误恢复　错误恢复可分为向前错误恢复和向后错误恢复两大类[3]，这是两类互补的恢复方法。向后恢复是把出错的系统进程卷回到一个已知的正确状态，向前恢复则根据故障特征，对错误状态进行相应处理，使系统进程正确运行下去。它们的一个不同在于向后恢复简单地把变量恢复到检查点的取值，而向前恢复将对一些变量的状态进行修改和处理，且这个恢复过程将由进程设计者设计[3]；另一个不同在于向前恢复适用于可预见的易定义的错误，而向后恢复可屏蔽不可预见的错误。在防卫式程序设计中，对于代码数据或链式结构的损坏，可用向前恢复，而向后恢复是一种更易于实现的技术，可以保持一个安全状态的细节，发生错误时再对该状态进行恢复。有两种处理办法：一种是数据状态的变换先不写入，等到处理全部完毕而没有错误出现时，才写入变换后的状态，这种处理多用于数据库；另一种在一定时间间隔的检查点制作安全状态的拷贝，发生错误时，恢复最近检查点的状态。

二、一个防卫式程序设计的完整实例

DP 的思想可用于任何一种程序设计语言，ADA 语言的数据抽象和异常处理机制是体现防卫式程序设计思想的有利工具，下面我们将介绍的是一个采用防卫式程序设计的方法开发的 ADA 程序实例。

1. 问题的描述

对任意类型的一组元素进行快速排序，设解决该问题的程序过程为 P，则有：

任意类型的一组元素 A(1..n) $\xrightarrow{\text{输入}}$ P $\xrightarrow{\text{输出}}$ 该组元素的一个有序排列 A'(1..n)

2. 解决问题的策略及程序结构

我们知道，快速排序的过程是这样的：

(1)在待排序的数组段中选取一个数组段；

(2)若该段只有两个元素，则直接进行比较；否则，以该段中的某一元素 X 的值为界对该数组段进行一次划分，划分得到三部分：[不大于 X 的元素]X[大于 X 的元素]；

(3)划分所得的每一段又为一待排序的数组段；

为实现这样一个过程，我们的策略是：

(1)采用一个集合，存放所有待排序数组段的左右端点(如初始时待排序的数组段为 A(1..n)，则集合的值为{(1,n)}，每次从集合中取出一个数组段进行处理，处理结束后将划分得到的数组段又存入集合中，直到集合为空，说明排序完成；

(2)在对一个数组段的处理过程中，使用两个子过程分别实现两个元素的排序和对数组的划分。

那么，在我们的程序结构中，包含如下程序单元：①程序包 SET_OP 是一个集合抽象数据类型的程序包，其中包含集合操作的实现；②子过程 SORT_TWO 实现两个元素的排序；③子过程 PAR-

TITION 实现对数组的一次划分；④主过程 ABSORT 通过对 SET_OP 中的过程进行调用实现集合元素(即待排序的数组段)的存取，通过调用 SORT_TWO 和 PARTITION 子过程实现对一个待排序数组段的处理。

需要说明的是，以上程序单元均为类属程序单元，排序通过 ADA 的类属机制实现，对于某具体类型，要先对 ABSORT 过程进行实例化，有关内容可参考文[6]。

3. DP 思想在程序实例中的应用

应用防卫式程序设计方法设计上述程序，就要求在每一程序单元的设计中考虑有关的错误检查和处理。例如，程序包 SET_OP 的说明如下：

```
generic
   type item is private;
package set_op is
   type set is private;
   (集合的操作说明);
   state_error:exception;
private
   ……
end set_op;
```

在对 SET 的操作中，所有对集合状态进行改变的操作过程中都需插入错误检查代码，如在 SET 中删除一个元素的操作过程 remove 的过程体为：

```
procedure remove(x:in item;s:in out set)is
  begin
     if not in(x,s)then
        error:=true;
        raise state_error;
     else
        (将 X 从集合 S 中删除)
     end if;
end remove;
```

如果所要删除的元素 X 不在集合 S 中，则置错误状态 ERROR 为真，并通过异常处理 STATE_ERROR 对错误进行相应处理。在抽象数据类型 SET 中，这样的错误检测与处理代码只需一次，对任意一个具体集合的操作过程中就会作出相应的检测和处理。这样，在 ABSORT 中，如果出现了对集合的错误操作，即将转入相应异常处理段予以处理。

为保证排序结果的正确性，在排序过程中也要插入状态检查和恢复的处理代码，如下所示：

```
with text_io;use text_io;
with set_op;
generic
……
procedure absort(b:in out vector;n:in out integer)
is
(类型说明、变量说明与过程说明)
sort_error:exception;
begin
  for i in 0..n-1 loop
  C(i):=b(i);
end loop;
(排序过程)
for i in 0..n-2 loop
  if b(i)>b(i+1) then
     raise sort_error;
  end if;
exception
  when sort_error=>
     for i in 0..n-1 loop
        b(i):=C(i);
     end loop;
     raise;
  when……
     (其它异常的处理程序)
end absort;
```

在排序之前，先使用一个数组 C 制作原始数组的备份，排序结束时，检查其变量状态是否满足排序算法的后置断言，不满足则说明排序过程中发生了错误，则转入 SORT_ERROR 的处理，恢复原始数组的状态。

4. 实例分析与评价

上述实例我们采用了 DP 的思想和技术以实现容错的功能。可以看出，使用 DP 的插入断言法和 ADA 中的异常处理机制可以处理可预见的错误，如集合操作中的一些可预见的出错情况的处理，在快速排序的主过程中，我们使用了 DP 后向恢复的制作拷贝方法，这种处理可以容忍一些不可预见的错误，如对偶然事件或硬软件的故障，引起的排序错误，程序可恢复到排序前的状态，但是，假如在快速排序的程序中存在逻辑错误，那么不管进行多少次后向恢复，都永远不可能得到正确的结果。

防卫式程序设计是一种较易实现的方法，比较灵活，尤其适用于小程序或局部程序的容错。尽管防卫式程序设计不能处理算法中存在的逻辑错误，但对程序运行过程中出现的硬件故障或偶然事件引起的错误能起到屏蔽的作用，并且由于其执行效率高、占用空间小、易于设计和实现，因此，是对软件容错技术的重要补充。我们赞成把 DP 方法与恢复块和多版本程序设计结合使用来提高软件的可靠性。

参考文献

[1]Ian Sommervice, Software Engineering, Addison-Wesley, 1992

[2]K. H. Kim, Software Fault Tolerance, Handbook of software engineering

[3]F. Cristain, Exception Handling and Software Fault Tolerance, IEEE Trans. on Computers. Vol. C-31. №. 6. 1982

[4]袁由光、陈以农，《容错与避错技术及其应用》，科学出版社，1992

[5]郦萌，《计算机软件的可靠性》，国防工业出版社

[6]薛锦云、吴云峰、万剑怡，循环控制抽象和迭代算子研究，中国计算机学会抗恶劣环境专委会软件学组成立及第一届学术讨论会论文，1994. 5