**并发变异体的生成报告**

# 1所有变异算子介绍

## 1.1 并发变异算子介绍

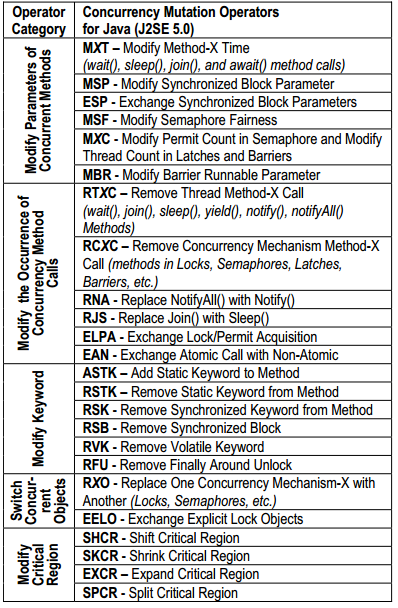


图1.1 Java并发程序的变异算子

对待测的优先级队列算法进行变异的依据是Bradbury在[1]中提出的24种并发变异算子，如图1.1所示。可以分为5类：改变并发方法的参数、改变并发方法的调用、改变关键字、转换并发对象以及改变临界区。

待测的5个程序涉及到的并发机制有三种：一，synchronized关键字修饰方法；二，原子类型；三，重入锁机制。因此根据每一个变异算子的应用情况，以上24种变异算子可以作用到待测程序的有8种。

接下来的文章中在第2章介绍运用到待测程序(SUT)的变异算子，第三章介绍每一SUT的变异情况。

## 1.2 传统变异算子介绍

表1.1 传统变异算子介绍

|  |  |  |
| --- | --- | --- |
| Category | Operator | Description |
| Arithmetic | AORB  AORU  AORS | Arithmetic Operator Replacement (binary)  Arithmetic Operator Replacement (unary)  Arithmetic Operator Replacement (short-cut) |
| AOIU  AOIS | Arithmetic Operator Insertion (unary)  Arithmetic Operator Insertion (short-cut) |
| AODU  AODS | Arithmetic Operator Deletion (unary)  Arithmetic Operator Deletion (short-cut) |
| Relational | ROR | Relational Operator Replacement |
| Conditional | COR  COI  COD | Conditional Operator Replacement  Conditional Operator Insertion  Conditional Operator Deletion |
| Shift | SOR | Shift Operator Replacement |
| Logical | LOR | Logical Operator Replacement |
| LOI | Logical Operator Insertion |
| LOD | Logical Operator Deletion |
| Assignment | ASRS | Assignment Operator Replacement (short-cut) |
| Deletion | SDL | Statement Deletion |
| VDL | Variable Deletion |
| CDL | Constant Deletion |
| ODL | Operator Deletion |

对待测程序进行变异的依据是Ma在[3]中涉及到的16种变异算子，如表1.1所示。所有的变异算子可以分为7类：算术算子、关系算子、条件算子、位算子、逻辑算子、赋值算子以及删除算子。

# 2运用到SUT的变异算子介绍

## 2.1 并发变异算子的应用情况

通过分析待测程序并结合变异算子的作用情况，可以将表2.1中的8个变异体运用到5个待测程序中。每一个算子的详细运用信息在Bradbury的技术报告[2]中有详细的论述。

表2.1 并发变异算子的应用情况

|  |  |
| --- | --- |
| 变异算子编号 | 变异算子说明 |
| RCXC | 移除并发机制方法的调用 |
| SAN | 将原子调用转化为非原子调用 |
| ASTK | 在非静态synchronized方法的前面加上static |
| RSK | 移除synchronized关键字 |
| RFU | 移除包含unlock语句的finally关键字 |
| RXO | 用另一个锁对象代替目前的锁对象 |
| EELO | 交换锁对象 |
| ELPA | 改变lock调用的方法 |

## 2.2 传统变异算子的应用情况

表2.2 传统变异算子的应用情况

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operator | SimpleLinear | SimpleTree | SequentialHeap | FineGrainedHeap | SkipQueue |
| AORB |  |  |  |  |  |
| AORU |  |  |  |  |  |
| AORS |  |  |  |  |  |
| AOIU |  |  |  |  |  |
| AOIS |  |  |  |  |  |
| AODU |  |  |  |  |  |
| AODS |  |  |  |  |  |
| ROR |  |  |  |  |  |
| COR |  |  |  |  |  |
| COI |  |  |  |  |  |
| COD |  |  |  |  |  |
| SOR |  |  |  |  |  |
| LOR |  |  |  |  |  |
| LOI |  |  |  |  |  |
| LOD |  |  |  |  |  |
| ASRS |  |  |  |  |  |
| SDL |  |  |  |  |  |
| VDL |  |  |  |  |  |
| CDL |  |  |  |  |  |
| ODL |  |  |  |  |  |

v

表2.2中是传统变异算子在5个待测程序的应用情况：在5个待测程序中分别去除使程序不能正常执行的变异体之后剩下的变异体对应的变异算子。打“”则表示某一个待测程序去除不能执行的变异体之后剩下的变异体对应的变异算子。

表2.3 所有应用的变异算子分类

|  |  |  |
| --- | --- | --- |
| Category\_1 | Arithmetic | AORB、AORS、AOIU、AOIS |
| Category\_2 | Relational | ROR |
| Category\_3 | Conditional | COR、COI、COD |
| Category\_4 | Logical | LOI |
| Category\_5 | Deletion | SDL、VDL、CDL、ODL |
| Category\_6 | Modify method calls | RCXC、SAN、ELPA |
| Category\_7 | Modify keyword | ASTK、RSK、RFU、 |
| Category\_8 | Switch objects | RXO、EELO、ELPA |

## 2.3 所有应用的变异算子分类

## 将所有的待测程序中应用的变异算子可以分为8类，如表2.3。

# 3每一个待测并发程序的变异体

本文并发变异体的设计满足并发机制的定义使用覆盖准则。例如根据某一个或者某些变异算子变异每一个lock对象的创建以及使用。

## 3.1 SimpleLinear程序的变异体

## 3.1.1 并发变异体

该程序的并发机制体现在Bin类中表3.1，该类有3方法：put()、get()、isEmpty()。并且这三个方法均加上了“synchronized “关键词。其中isEmpty()方法判断当前的列表中是

表3.1 Bin类

|  |
| --- |
| 1. **public class** Bin<T> { |
| 1. List<T> **list**; |
| 1. **public** Bin() { **list** = **new** ArrayList<T>(); } |
| 1. **synchronized void** put(T item) {**list**.add(item);} |
| 1. **synchronized** T get() { |
| 1. **try** {**return list**.remove(0);} |
| 1. **catch** (IndexOutOfBoundsException e) {**return null**;}} |
| 1. **synchronized boolean** isEmpty() {**return list**.isEmpty();}} |

否存在元素，在我们的设计的测试用例中没有覆盖该方法因此，不考虑这个方法的变异。并且由于我们研究的重点在于get()方法，因此不考虑put()方法。根据该程序的并发机制可知可以用RSK以及ASTK变异算子进行变异。RSK变异算子的变异原理为：移除synchronized关键字；ASTK变异算子的变异原理为：在非静态synchronized方法的前面加上static。因此运用RSK变异算子可以得到1个变异体。然而在考虑ASTK变异算子时，由于要在get()方法上加上“static“关键字，则Bin类的成员变量list也要加上”static“关键字，否则系统报错。当成员变量前面加上static时，即便创建多个Bin对象也只有一个list不符合程序的逻辑：每一个Bin对象都有自己的list。

## 3.1.2 传统变异体

该程序的传统变异体是用Mujava工具自动生成的，去掉错误的变异体以及等价变异体还剩下11个变异体。变异体的具体情况见SimpleLinear传统变异体信息.xlsx文件。

## 3.2 SimpleTree程序的变异体

## 3.2.1 并发变异体

该类的并发机制的实现是基于原子变量，因此可以用SAN变异算子进行变异。由于目前本文只考虑removeMin()方法的并发情况因此可以得到一个变异体：SAN\_1,源程序与并发变异体如表3.2。

表3.2 SimpleTree源程序与变异体对照

|  |  |
| --- | --- |
| 源程序： | SAN\_1: |
| public T removeMin() { | public T removeMin() { |
| TreeNode node = root; | TreeNode node = root; |
| while(!node.isLeaf()) { | while(!node.isLeaf()) { |
| if (node.counter.getAndDecrement() > 0 ) { | if (node.counter.get() > 0 ) { |
| node = node.left; | node.counter.set(node.counter.get() - 1); |
| } else {node = node.right;}} | node = node.left; |
| return node.bin.get();} | } else {node = node.right;}} |
|  | return node.bin.get();} |

## 3.2.2 传统变异体

传统变异体的情该程序的传统变异体是用Mujava工具自动生成的，去掉错误的变异体以及等价变异体还剩下29个变异体。变异体的具体情况见SimpleTree传统变异体信息.xlsx文件。

## 3.3 SequentialHeap程序的变异体

## 3.3.1 并发变异体

在对源程序进行实验的过程中发现，在一些情况下，源程序可能出错。因此我们在removeMin()方法以及swap()方法的加上了synchronized关键字。由此就可以用RSK以及ASTK变异算子进行变异。RSK变异算子的变异原理为：移除synchronized关键字；ASTK变异算子的变异原理为：在非静态synchronized方法的前面加上static。利用RSK变异算子针对removeMin()方法以及swap()方法可以产生2个变异体。利用ASTK变异算子及进行变异时，涉及到很多的成员变量要增加static关键字。这样做是不恰当的。因此该待测程序的并发变异体有2个。

## 3.3.2 传统变异体

传统变异体的情该程序的传统变异体是用Mujava工具自动生成的，去掉错误的变异体以及等价变异体还剩下244个变异体。变异体的具体情况见SequentialHeap传统变异体信息.xlsx文件。

## 3.4 PrioritySkiplist程序的变异体

## 3.4.1 并发变异体

该程序的并发机制是在原子变量的基础上实现的，因此可以用SAN变异算子，共得到1个变异体。该类涉及到的原子变量类型有两个分别是：AtomicBoolean。变异策略为：将该类变量调用的原子方法转化为非原子调用。例如：marked是AtomicBoolean类型的变量，可以将marked.compareAndSet(false,false)转化成两个非原子语句：

if (curr.marked.get() == false){ curr.marked.set(false);}

## 3.4.2 传统变异体

传统变异体的情该程序的传统变异体是用Mujava工具自动生成的，去掉错误的变异体以及等价变异体还剩下244个变异体。变异体的具体情况见SkipQueue传统变异体信息.xlsx文件。

## 3.5 FineGrainedHeap程序的变异体

## 3.5.1 并发变异体

该程序通过重入锁实现并发的机制。因此可以用RXO、EELO、ELPA算子对该程序进行变异。将以上三种变异算子运用到removeMin()方法上一共可以得到7个变异体。

## 3.5.2 传统变异体

传统变异体的情该程序的传统变异体是用Mujava工具自动生成的，去掉错误的变异体以及等价变异体还剩下244个变异体。变异体的具体情况见SkipQueue传统变异体信息.xlsx文件。

# 参考文献

[1] Bradbury, Jeremy S., J. R. Cordy, and J. Dingel. "Mutation Operators for Concurrent Java (J2SE 5.0)." *The Workshop on Mutation Analysis* IEEE Computer Society, 2006:11.

[2] Mutation Operators for Concurrent Java (J2SE 5.0) Technical Report 2006-520

[3] Description of muJava's Method-level Mutation Operators