一种基于活动图的并发软件测试线索生成方法

曾 一 张利武 张元平 袁 纲 李 强
(重庆大学计算机学院 重庆 400044)

摘 要 分析并发软件的控制原理,提出了三个并发软件的基本测试策略。这三个策略给出了并发软件测试的基本原则:同步测试、关键覆盖和进程覆盖;接着提出了一种使用 UML 活动图对软件中并发控制过程建立模型图的方法;随后分析了并发软件测试难点即进程组合爆炸问题,给出了一种基于模型图生成测试线索集的方法,证明了按照生成的测试线索集测试并发软件既能解决进程组合爆炸问题,又能满足三个测试策略。通过对比得出本测试方法在易用性、适用性和稳定性等方面要优于基于状态图的测试方法;最后通过一个实例表明了应用本方法的可行性和有效性。

关键词 测试策略,活动图,组合爆炸,测试线索

Activity Diagram-based Method to Generate Test Sequence of Concurrent Software

ZENG Yi ZHANG Li-Wu ZHANG Yuan-Ping YUAN Gang LI Qiang
(College of Computer Science, Chongqing University, Chongqing 400044)

Abstract Three basic concurrent software test strategies are proposed after the principle of concurrent control is analyzed. These three strategies give out the basic principles of concurrent software test: synchronization cover, key method cover and threads cover etc. Then, a method based on activity diagram is developed to model the concurrent flow of the software. Following that, the difficulty of test concurrent software-thread combination explosion is analyzed, and amethod to generate test sequence is proposed. To test concurrent software with the test sequence generating by the method is proved to solve the difficulty of thread combination explosion and fulfill the test strategies given before, according to the analysis later. And a contrast shows that this method is better than the method based on state diagram when the stability, adaptability and convenience are concerned. At the end, an example is given to show the feasibility and the validity of applying the method.

Keywords Test strategy, Activity diagram, Combination explosion, Test sequence

1 序言

并发软件可以提高系统资源的利用率,提高软件的工作效率。但是,并发软件在同样的输入条件下多次执行,可以得到不同的但却都是正确的结果,即它的执行路径具有不确定性。这种不确定性使得对并发程序的测试比对传统串行程序的测试更加困难。目前,国内对面向对象软件的并发测试主要集中在性能测试,主要思想是采用某种方式模拟多个客户并发调用被测软件。这种方法类似于传统软件的随机测试方法,不能获得满意的测试效果。国外 Kim Soon-Kyeong 等人在文[1]里提出了基于 UML 状态图的 java 并发软件测试方法,其主要思想是对并发软件在执行过程中的等待进程个数进行状态建模,并在此基础上提出了遍历 3 种状态(等待线程数量为 1、2 和 3 的状态)的一种测试方法。该方法有 3 个方面的不足:(1)在该方法中提出了一种特殊的状态图建模方法,增加了在实际使用过程中的难度;(2)该方法利用了 java 中的 Object 对象和 Thread 对象的特性,因此具有一定的局限性;(3)该方法是对类的静态建模方法,在测试的时候只要求出现 2 个以上的进程对象同时等待,没有考虑进程对象的选取,故在实际的测试过程中只能测试有限次数的随机进程对象组合,其测试效果不稳定。

本文主要给出了一种基于并发控制流图生成并发软件测试线索集的方法。按照本方法生成的测试线索来测试并发软件可以解决组合爆炸问题并且满足本文提出的三个测试策略。并且和 Kim Soon-Kyeong 等人提出的基于状态图的测试方法相比,本文方法有更好的易用性、适应性和稳定性。最后,一个数据采集软件实例给出了应用本测试算法的例子,验证了本方法的可行性和有效性。

2 并发软件相关理论

并发软件和传统软件最大区别是并发性,但是并发软件的错误并不是全部由软件的并发性引起的,本文主要考虑测试并发性引起的软件错误。

设 A 和 B 代表软件中的两个并发模块。

若 A 和 B 之间无任何资源或数据依赖关系,那么无论 A 和 B 执行的先后顺序怎样,都不会影响软件的结果,称这样的并发软件是等价可串行化的并发软件。这类软件里存在的问题不是由软件的并发性引起,本文不予考虑。

若 A 和 B 之间存在共享资源依赖关系,那么由于并发性的存在,A 和 B 的执行顺序是不确定的,A 和 B 访问共享资

曾 一 硕士、教授,主要研究领域为软件工程,软件度量,面向对象技术及应用;张利武 硕士,主要研究领域为软件测试、面向对象技术、并发系统的设计和应用;张元平 硕士,主要研究领域为面向对象技术、XML 数据搜索和交换技术;袁 纲 硕士,主要研究领域为面向对象技术、面向服务的体系架构;李 强 硕士,主要研究领域为软件测试、系统集成研究和应用。

源的顺序也是不确定的，这样就有可能出现软件某次按照某个顺序执行后得到的结果是错误的。本文主要考虑寻找这类软件中的错误。

为了解决共享资源的冲突问题，现在比较普遍的做法是使用锁机制来控制对共享资源的访问，任何并发进程只有申请得到对应的锁之后才能访问共享资源，这样通过限制锁的发放可以限制进程对共享资源的并发访问，从而实现并发软件的正确性。

2.1 并发软件测试策略

为了并发软件测试能取得好的效果，针对并发控制的特点，提出如下三个并发软件的测试策略。

同步测试策略：一个并发软件的测试案例在执行测试过程中至少使得软件在某一时刻有 2 个或 2 个以上的并发模块在等待对共享资源的访问。这个策略偏向于测试并发模块处理同步的问题，故称为同步测试策略。

关键覆盖策略：一个并发软件的测试案例在执行测试过程中关键资源相关的每个操作至少会被一个进程对象访问。这个策略要求测试时要同时覆盖关键资源的每个操作，故称为关键覆盖策略。若每个操作刚好被一个进程对象访问，则称为恰好覆盖。

进程覆盖策略：一个并发软件的测试案例集合应该满足全部测试案例集执行完毕后软件中的每个并发进程至少出现过一次等待对共享资源的访问。这个策略主要是要求达到并发进程的覆盖，故称为进程覆盖策略。

上述三个策路覆盖了并发控制的关键点，若一个测试过程能同时满足上述三个测试策略，那么本文认为该测试过程对发现并发错误是比较有效的。

2.2 并发软件测试建模

为了测试并发软件，需要先利用某种方式描述被测软件。孙钟秀等人提出了一种进程代数的形式化描述方法[2]，它将并发进程描述为动作序列，其中动作又可对应到规约级事件，这种方法使用难度较大，对测试人员的要求较高。UML 的活动图一般用来对系统中的对象建模，易于描述系统的动态特征，具有描述并发控制流的能力，适合于描述动态的并发过程。为了利用活动图进行建模，首先做如下抽象：

1) 系统中具有并发执行能力的执行单位可抽象为一个进程对象，每个进程对象只访问一个关键操作；

2) 系统中每个关键资源抽象出一个与之对应的封装对象，系统中所有和关键资源相关的访问和操作都通过该对象提供；

3) 关键资源封装对象自己拥有管理对应关键资源对象锁的能力(一般由运行环境提供)。

可以看出，这 3 步抽象过程只是并发执行系统的面向对象视图，并不影响系统的真实执行过程。经过上述抽象，使用活动图建立的并发控制流图如图 1 所示。实际的系统中可能存在多个并发进程，这里限于篇幅只画出四个并发进程，其它进程对象的执行过程与此类似。

图 1 中每个进程对象的同步条(活动图中的合并条)表达了进程对象对关键对象的并发约束依赖关系。关键对象中的分支条表达了任意时刻至多只能有一个线程获得关键资源锁的约束关系，保证并发系统的排它性，而且任何进程对象执行等待锁操作后处于等待状态，该对象直到获得分配的资源锁才能继续执行，这样就保证了图中任意时刻至多只有单个线程能够达到访问关键资源的目的。该图准确描述了并发系统中并发控制的关键特性，为并发测试工作提供了基础。

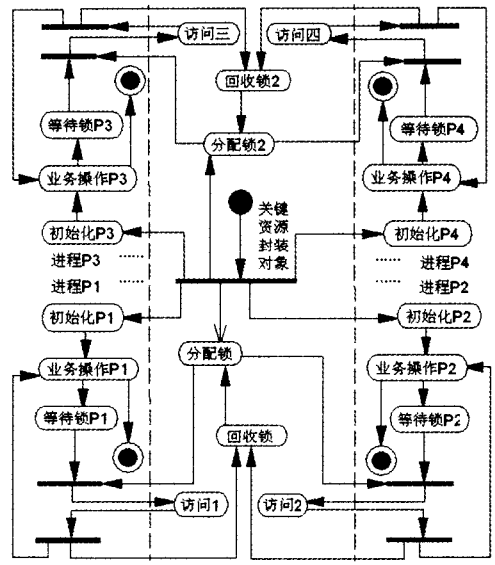

图 1 并发控制流图

3 进程组合爆炸问题

从建立的模型图可以清楚看到系统并发控制流程，若能够测试软件在任何流程组合情况下都是正确的，那么就可以确定软件中是不存在并发错误的。然而，这只是一种理想情况。不妨假设系统中关键对象有 n 个关键操作，分别记为 $OPR_1, OPR_2, OPR_3, \cdots, OPR_n$。系统运行时访问关键操作 OPR_k (k 为整数且 $0<k\leqslant n$) 的并发进程对象有 $m(k)$个，分别记为 $OBJ_k^1, OBJ_k^2, \cdots, OBJ_k^{m(k)}$，在 $m(1), m(2), m(3), \cdots, m(n)$ 中不妨设 $m(1)$最大。系统中进程对象总数 M 为

$$M=\sum_{k=1}^{n} m(k)$$

一般而言，并发系统中进程至少有 2 个，即 M 不小于 2。系统一次运行结束后，每个并发进程对象至少获得过锁一次，所以在整个运行过程中系统分配锁的次数 L 可以看作 M 的线性函数，不妨设为 $L = a \cdot M$(a 为大于 1 的常系数)，则自然状况下系统运行的全部进程组合数 C 为

$$C = M^L = M^{aM} \geqslant 2^M \quad (M \geqslant 2) \qquad (1)$$

由此可以看出系统中进程数 M 线性增长时，系统中进程组合数 C 的增长速度甚至超过了指数级的增长速度。所以，在实际系统中当并发进程对象较少时，图中的流程组合数量也较少，测试时还可以覆盖图中的全部流程组合。但是，随着并发进程对象数量的线性增多，系统中进程组合的数量会出现爆炸性增长，在测试中全部覆盖这些组合将会非常困难，也没有现实意义。这类似于传统软件测试过程中的路径组合爆炸问题。

为了降低一般情况下测试的难度，同时获得较好的测试效果，需要从模型图中选择一些具有代表性的流程组合来测试软件，以发现软件的并发错误。每个流程组合称之为一个测试线索[3]，多个流程组合形成测试线索集。

4 并发软件测试线索生成算法

4.1 测试线索表示

为了便于表示测试线索，引入如下几个符号；

“()”表示整体关系；

“,”表示先后顺序；

“||”表示并发关系。

设 a1,a2,a3 为活动图上的活动节点，则(a1||a2),a3 表示 a1 和 a2 是并行完成的，a1 和 a2 一起作为整体要先于 a3 完成，更多的例子请参考第 5 章的线索例子。

4.2 测试线索生成算法

本文介绍的测试线索生成方法引用了一种基于活动图的功能测试线索生成方法[4]，其主要思想是在活动图的顶点标上对应的方法和参数，然后深度遍历活动图，获得多条带参数的遍历路径，按照约束将其中一些路径合并，最后结合时序图将这些方法和参数替换成底层调用序列，形成测试线索。

将活动图中活动节点和分支合并条看作顶点，那么并发模型图就可以看作一个有向图。为便于讨论，将其定义为：

$G=(V, E, v_s, v_e)$，

其中

$V=\{v_i \mid v_i$ 表示图上的顶点，i 为自然数$\}$，v_s 表示开始顶点，v_e 表示结束顶点，v_s 和 v_e 都属于 V；

$E=\{e_j \mid e_j$ 表示图上连接顶点的有向边，j 为自然数$\}$。

设有向边 e_j 的起点为 v_m，终点为 v_n，e_j 表示为有序对 (v_m,v_n)，则

$Begin(e_j)=v_m$，表示边 e_j 开始于顶点 v_m；

$End(e_j)=v_n$，表示边结束于顶点 v_n。

设 o_k 表示图上的第 k 个对象($k \geqslant 1$)，关键对象用 o_1 表示，则

$Owner(v_i)=o_k$，表示活动 v_i 的属主对象为 o_k，活动 v_i 在活动图上处于对象 o_k 的泳道内；

$Act(o_k)=\{v_p \mid Owner(v_p)=o_k, p$ 为自然数$\}$，表示对象 o_k 的所有活动集合。由于锁的分配、回收、申请和释放操作一般由底层环境提供，而测试的时候主要考虑业务操作的正确性，所以在 $Act(o_k)$ 中将不包括这些操作；

$VR(v_n)=\{o_k \mid$ 存在 e_j 和 v_m，满足 $Begin(e_j)=v_m$，$End(e_j)=v_n$，$Owner(v_m)\neq o_k$ 且 $Owner(v_m)=o_k\}$，表示顶点 v_i 的直接引用对象集。$|VR(v_n)|$表示 $VR(v_n)$中包含的对象元素数量。

测试线索生成算法描述如下：

1) 遍历顶点的集合 V，对 v_i 属于 V，若 $Owner(v_i)=o_k$ 则将 v_i 加入 $Act(o_k)$中，这一步结束后可以获得每个对象的顶点集合 $Act(o_k)$；

2) 遍历边的集合 E，对边 e_j 属于 E，若 $End(e_j)=v_i$，且 $Owner(Begin(e_j))$不等于 $Owner(v_i)$则将 $Owner(Begin(e_j))$加入 $VR(v_i)$中，这一步结束后可以得到每个顶点的引用对象集合；

3) 按顺序从关键对象每个顶点的引用对象集中选择一个进程对象(在选取的时候优先选取未在其它测试线索中出现的进程)组成一组对象集(若对象集中对象的数量少于 2 个，则再任意选择一个进程对象加入集中)，以这组对象集和关键对象相关的顶点作为顶点集构造原并发模型图的一个顶点子图，重复这一过程直到所有对象至少出现过一次，设 $Act(o_1)=\{v_1,v_2,v_3,\cdots,v_n\}$，$m=Max(|VR(v_p)|)$ $(0<p<=n)$，则获得 G 的 m 个子图的算法，记为子图算法，描述如下：

```
For ( i = 1; i ≤ m ; i++ ) {
    Oi = o1 ;
    Vi = Act(o1);
    k = 0 ;
    For ( j = 1 ; j <= n ; j++ ) {
        v = vj (vj 为 Act(o1)中的第 j 个顶点元素) ;
        k = i mod |VR(v)| ;
        oij = ok (ok 为 VR(v)中的第 k 个对象元素) ;
        Oi = Oi ∪ oij ;
        Vi = Vi ∪ Act(oij) ;
    }
    if(n = = 1 ){
        k = k + 1;
        oij = ok (ok 为 VR(v)中的第 k 个对象元素) ;
        Oi = Oi ∪ oij ;
        Vi = Vi ∪ Act(oij) ;
    }
    Gi = 以 Vi 为顶点集的 G 的顶点子图;
}
```

本步骤结束后还可以得到每个子图 G_i 相关的进程对象集 O_i。

4) 对子图 $G_1,G_2,G_3,\cdots,G_m$ 利用基于活动图的功能测试线索生成方法[4]分别生成测试线索 $s_1,s_2,s_3,\cdots,s_m$，这些测试线索构成系统的测试线索集 S。

4.3 算法证明

从步骤 3 的子图算法中很容易看出整个测试集中包含的线索数为 m(外部 for 循环)，线索 s_i 包含了 n 个进程对象(内部 for 循环)，由前面 C 的推导过程可知，按线索 s_i 执行软件的过程中分配锁的次数 L_i 为

$L_i=b_i \cdot n$，(b_i 为常系数)

进程组合数为 C_i 为

$C_i=n^{L_i}=n^{b_i \cdot n}=n^{b_i} \cdot n^n$

所以整个测试集中包含的进程组合数 C' 为

$$C'=\sum_{i=1}^{m} C_i = n^n \cdot \sum_{i=1}^{m} n^{b_i}$$

一般情况下，关键对象包含的操作数不会太大，即 n 一般较小，可以合理地认为 $n<=3$，则此时的进程组合总数

$C' = l \cdot m$ (l 为常系数，m 为顶点的最大进程引用数)

又因为 $M=\sum_{p=1}^{n} |VR(v_p)|$

而 $m=Max(|VR(v_q)|)$ $(0<q\leqslant n)$

故 $m \leqslant M$，设 $m = d \cdot M$(d 为常系数，$0 < d \leqslant 1$)，则

$C'=d \cdot l \cdot M$

即进程组合数 C′是随着进程数 M 线性增加的，解决了前述的组合爆炸问题。

从算法内部 for 循环可知，任意测试线索 s_i 中至少包含 2 个进程对象。那么当系统未分配锁给任何一个进程，按照 s_i 测试软件时，处于等待锁状态的进程数也至少为 2，满足了同步策略的要求；从算法内部 for 循环还可知关键对象的活动在线索 s_i 里至少出现一次，从模型图中又可以看出每个进程至少要访问一次关键对象操作才有可能进入结束状态，所以按照 s_i 测试结束后关键对象的相关操作都至少执行了 1 次，满足了关键覆盖策略的要求；从算法外部 for 循环和 m 的定义可知每个进程对象至少在某个线索里出现 1 次，所以按照整个测试集完成测试后，每个进程对象至少执行过一次，也就是至少等待过一次对资源的访问，即说明测试过程满足了进程覆盖策略的要求。

综上证明了按照本算法生成的测试线索集来测试并发软件可以解决进程组合爆炸问题，而且还满足了并发软件测试的三个测试策略。

4.4 方法对比

由于在并发软件中任一时刻，锁最多只能分配给一个进程，而进程只有获得锁之后才能脱离等待锁的状态，所以整个测试过程里，等待进程的数量只能以单个数量变化，而且任意线索 s_i 里至少含有两个进程对象，即同时等待的进程数量至

少为2，而测试结束后等待的进程数量为0，则在算法的执行过程中必然存在某个时刻等待锁的进程数量为1。即按照本算法生成的测试线索测试过程中等待锁的进程数量经历过0,1,2这3种状态变化，覆盖了Kim S-K等人提出的利用状态图建模的测试要求[1]，即本算法至少可以达到与其相同的测试效果。

另外，由于Kim S-K等人提出的方法依赖于Java，对每个进程对象都要单独进行状态图建模，过程较为繁琐，且模型建立后不易看出系统的并发控制流，不容易获得测试线索。该方法使用随机进程对象组合测试，不利于获得稳定的测试效果。本文提出的方法使用活动图一次建模，建模后系统的并发控制流清晰明了，给出了测试线索的生成方法，并且在测试过程中进程对象有序组合，有效地避免了组合爆炸问题，有利于获得稳定的测试效果。由此看出本文方法在适应性、易用性和稳定性方面优于Kim S-K等人提出的方法。

5 实例应用

本文方法已经在一个Java开发的数据采集软件得到实际应用，取得了较好的测试效果。

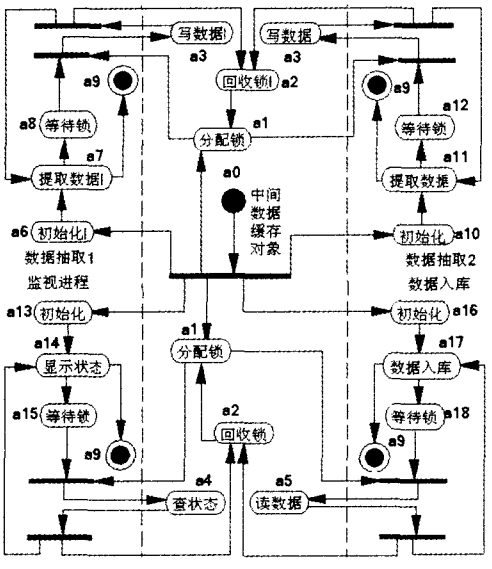

图2 数据采集软件并发控制流图

该数据采集软件主要功能是实现从几个不同的历史数据库里按照某种规则采集数据到目标数据库里。系统要求随时查看数据的读入写出情况。该系统中历史数据库分布的物理跨度较大，处于不同的网段，访问速度较慢，而目标数据库在本地，访问速度较快。为了提高数据采集的效率，系统采用并发机制来完成工作。每个历史数据库由一个独立的进程对象来完成数据采集，数据入库也由一个独立的进程对象完成，另外还有一个独立的进程对象负责监视数据读入情况。各个进程对象之间共享一个数据缓冲库，数据缓冲库缓冲数据，并记录数据的读入写出信息。这里关键对象为数据缓冲库，其余对象为并发进程对象，实际建立的模型如图2。

5.1 生成测试线索

根据3.2中介绍的算法生成本模型图的两个子图*G1*和*G2*分别如图3和图4。

再利用文[3]中的方法生成对应的测试线索分别为(线索均采用第4章介绍的方法表示)：

$s1 = a0, (a6, a7, a8 \parallel a1), a3, (a7, a9 \parallel a2), (a13, a14, a15 \parallel a1), a4, (a14, a9 \parallel a2), (a16, a17, a18 \parallel a1), a5, (a17, a9 \parallel a2)$。

$s2 = a0, (a10, a11, a12 \parallel a1), a3, (a11, a9 \parallel a2), (a13, a14, a15 \parallel a1), a4, (a14, a9 \parallel a2), (a16, a17, a18 \parallel a1), a5, (a17, a9 \parallel a2)$。

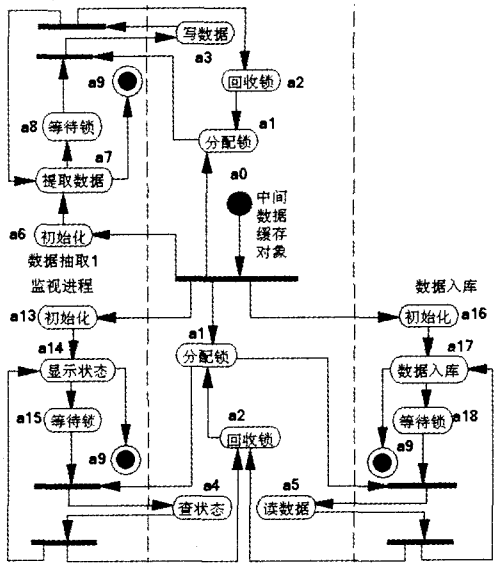

图3 子图G1

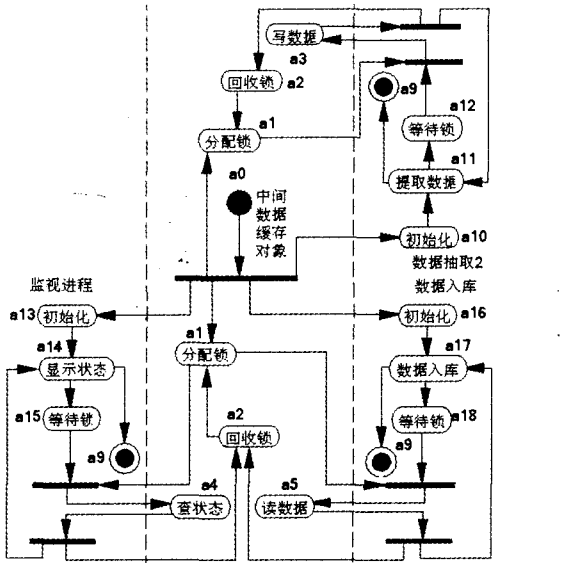

图4 子图G2

按分配锁的先后顺序不同每个子图有多个测试线索，这里只取一个线索作样例，其他的线索是类似的。图5给出了测试线索*s1*的示意图(图中粗线部分)，由图可见测试线索s1有效地减少了参与测试的进程数。该图中总进程数不多，但在实际系统中总进程数越多，图中减少的进程数也就越多，测试线索取得的效果也就越明显。

5.2 测试执行过程

由于本软件在java虚拟机上执行，数据缓存对象锁的分配和回收操作均由JVM自动实现，对象锁的分配顺序无法控制，为了实现测试线索对锁的控制要求，引入一个辅助对象控制锁的分配序号，对系统中每个实际进程引入一个包装对象进行包装，该包装对象在运行时首先向辅助对象申请序号，若申请得到的序号和其包装的进程在测试线索中应该获得锁的

序号相同，则启动它所包装的线程进行实际的执行，完成后将锁序号加1，这样系统中的线程就只能按照测试线索的顺序获得锁来执行直到结束。将该辅助对象和包装对象加入系统，此时系统的测试步骤如下：

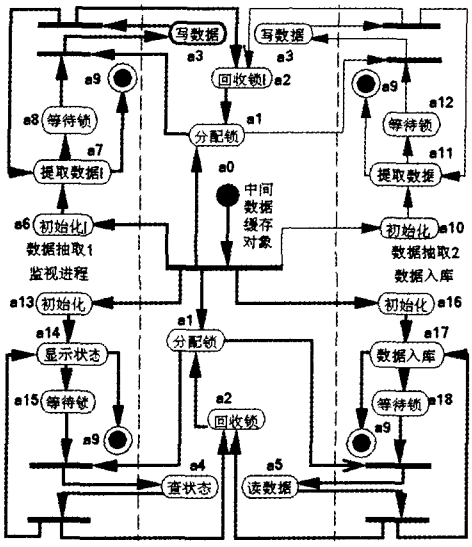

图5　线索 s1 示意图

1）启动系统的包装对象；

2）启动辅助线程对象，锁的初始分配序号为1；

3）系统开始正常执行，直到结束；

4）检查系统执行结果，给出测试结果。

5.3　测试结果

很容易验证测试线索 *s1* 和 *s2* 满足前面提出的三个测试策略。按照测试线索 *s1* 和 *s2* 执行测试，找出了原软件中的一个并发执行错误。检查发现数据读入对象读完数据但还未设置好状态时就释放了锁，这时数据监控对象读取了数据状态，导致数据监控对象显示错误的状态，即图5上的动作 *a3* 出错(粗线条表示)。在本实例中，按照测试线索 *s1* 或 *s2* 都能发现该并发错误，说明本文生成的测试线索具有较好的代表性。虽然该数据采集软件在实际运行过程中又发现了一些错误，但是经分析它们均不是由于并发而导致的错误。这个实例在一定的程度上表明本建模方法和测试算法对于查找软件中的并发错误有一定效果和帮助。

结论和展望　本文在分析并发控制原理的基础上，给出了并发软件的三个基本测试策略，这三个策略覆盖了进程同步、多进程并发和关键操作等并发控制的关键点，对测试并发软件具有较好的指导性。接着本文提出了利用活动图来对系统的并发控制流建模的方法，在分析并发测试的难点之后提出了一种测试线索集的生成算法，并证明了该算法生成的测试线索集能有效地避免测试过程中进程对象的组合爆炸问题，提高了其实际应用价值。而且按照该测试线索集测试软件，可以满足本文所提出的三个并发软件测试策略，说明本算法产生的测试线索集对发现软件中的并发错误具有较好的指导意义。文中最后还通过对比得出本方法在适应性、易用性和稳定性方面优于 Kim 等人提出的基于状态图的测试方法，并结合实际软件给出了应用该方法的样例，验证了其可行性和有效性。

本文在介绍测试方法时主要考虑的是单个关键资源对象和进程对象只访问单个关键操作的情况。在实际的并发软件中，关键资源对象可能有多个，进程对象也可能要访问多个关键操作，在本文的基础上，进一步的工作将要考虑这种情况下测试线索的生成方法和测试方法。

参 考 文 献

1　Kim S-K, Wildman L, Duke R. A UML Approach to the Generation of Test Sequences for Java-based Concurrent Systems. In: Software Engineering Conference, 2005. Proceedings. 2005 Australian 29 March-1 April 2005. 100～109

2　孙钟秀，韩杰，谢立，等. 基于有限状态进程的事件约束定义[J]. 软件学报，2002，13(11)：2163～2166

3　曾一，赵炜，张利武. 基于 UML 活动图生成功能测试线索的方法[J]. 计算机工程与设计，2006(12)

4　赵炜. 基于 UML 提取系统测试线索方法的研究[C]：[硕士论文]. 重庆大学计算机学院，2006. 04

(上接第285页)

[Register]→[Register] checkForChange (cash, price) [Register]→[Front] returnCash(cash)

[Register] → [Register] displayPrompt ("Use CorrectChange")[Front]

场景 2：[Customer] Accept(cash, selection)[Front]→[Front]getCustomedinput(cash,selection)

[Register]→[Register] checkForChange (cash, price) [Register]→[Register] checkForChange (cash, price) [Dispenser]→[Register] returnCash (cash) [Register]→[Register]displayPrompt("Sold Out")[Front]

场景 3：[Customer] Accept(cash, selection)[Front]→[Front]getCustomedinput(cash,selection)

[Register]→[Register] checkForChange (cash, price) [Dispenser]→[Front]returnCash(cash)

[Register]→[Register] displayPrompt ("Sold Out") [Front]

场景 4：[Customer] Accept (cash, selection) [Front] [Front]getCustomedinput(cash,selection)

[Register]→[Register] checkForChange (cash, price) [Dispenser][Register]update (cash, selection)[Register]→[Register] receiveChange (cash,price)[Front]→[Register]releaseSoda(selection)[Dispenser]→[Dispenser] receiveSode(selection)[Front]

结束语　本文在分析多种测试模型的基础上，找到更为适合集成测试的测试模型并结合传统的白盒测试方法，总结出一种基于拆分、覆盖、重组的集成测试方法，在选择测试模型和改进方法方面较以前提高。

参 考 文 献

1　Binder R V. Testing Object-oriented System: Models, Patterns, and Tools. Addison-Wesley, 2000

2　UML Specification 2.0. http://www.uml.org, 2007

3　Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Addison—Wesley, 2001

4　袁洁松，王林章，李宣东，等. UMLTGF：一个基于灰盒方法从 UML 活动图生成测试用例的工具. 计算机研究与发展，2006，43(1)：46～53

5　Lund M S, StØlen K. Deriving tests from UML 2.0 sequence diagrams with neg and assert[c]. In: Proceedings of A the 2006 International Workshop on Automation of Software test, May 2006. 22～28