Java 并发系统的 ACP 模型

刘 园 徐宝文
(东南大学计算机科学与工程系 南京210096)

ACP Model of Java Concurrency

LIU Yuan XU Bao-Wen
(Department of Computer Science and Engineering, Southeast University, Nanjing 210096)

Abstract Feasibility and efficiency of analyzing concurrent programs mostly rely on the programs' representations. This paper proposes a model of Java concurrent system by using ACP, so that Java concurrent system can be transformed to process algebra expressions which facilitate model checking or some further analysis.

Keywords ACP, Java concurrent system, Thread, Monitor

1 引言

Java 作为一种面向分布式计算环境的语言，提供了完全意义上的多线程支持，能有效利用资源，提高系统效率，但是多线程并发也带来了很多严重问题，如死锁。为了详尽分析 Java 并发系统，首先必须对并发系统进行建模。

目前用于描述并发系统的方法主要有基于 Petri 网的方法[1~3]和基于流程图的方法[4]。这些方法各有长短，通常具有针对性，但没有一种方法能很好地支持所有的分析。本文也提出一种描述方法，采用进程代数理论对 Java 并发系统进行建模。由于进程代数[5,6]具有很强的描述通信自动机系统的能力，能很清晰地描述有限状态，方便地处理过程合成及隐藏内部细节，能有效地解决状态爆炸问题，故采用该体系非常适于分析并发系统。本文以三种最著名的进程代数理论中的一种作为描述工具，即 ACP（Algebra of Communication Processes）。本文首先简述了 Java 的并发机制；随后采用 ACP 对 Java 并发系统进行了建模，使得 Java 并发系统可转换为进程代数表达式，以便于进一步分析 Java 并发系统；最后在总结中提出基于进程代数形式化系统的工作展望。

2 Java 的并发机制

Java 提供了互斥锁的管程机制来保护临界区。因此 Java 的并发机制是一种管程机制。在 Java 中，所有的 Java 对象都有一个与之相连的隐含管程，Java 为每一个拥有 synchronized 方法的对象实例提供了一个唯一的管程。当调用 synchronized 方法时，线程进入相对应的管程，此时系统给临界区加锁。在同一时刻，只允许一个线程进入管程。当管程中已有一个线程在运行时，其它希望进入管程的线程必须等待，这种等待是由管程自动管理的。当线程退出管程后，即临界区开锁后，其它希望进入管程的线程才能再次竞争以获得管程控制权。

当一个线程进入管程后，可以通过 wait，notify 和 notifyall 这三个方法进行同步通信。wait 方法可以使进入管程的线程退出管程，在管程外休眠等待，以便其它线程进入管程，直至有其它线程进入同一管程并调用 notify 或 notifyall 方法将其唤醒。notify 方法用于唤醒等待当前锁的某一线程，而 notifyall 方法则唤醒等待当前锁的所有线程。

Java 规定这三种方法只能在同步方法中使用，如果一个在管程外部执行的线程使用了 wait，notify 或 notifyall 方法，则会产生错误。

3 Java 并发系统的 ACP 模型

进程代数的思想可追溯到经典自动机理论[5]。所不同的是，经典自动机理论只能描述单个自动机执行的迹，不具有描述通信自动机系统的能力。而进程代数是一种用代数结构描述和分析并发系统行为的强有力的形式化方法[6]。其中最有名的三种理论是 CCS、CSP 及 ACP。本文以 ACP 作为建模工具。

针对 Java 的管程机制，本文提出一种简单直观的管程模型，如图1所示。其中 wait，notify(notifyall)和 synchronized 方法分别和 monitor 模块交互通信。本文只考虑这些方法，不考虑 sleep，suspend 和 destroy 等等其它方法对线程的影响。

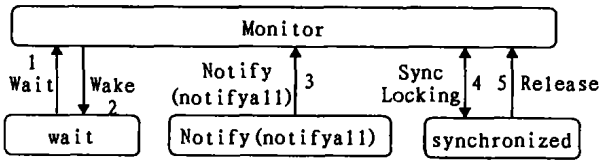

图1 Java 管程模型

(1)wait 模块从通道1向 monitor 发出 wait 消息，表明调用了 wait 方法，退出管程，等待来自通道2的 wake 消息。

(2)notify(notifyall)模块从通道3向 monitor 发出 notify(notifyall)消息，表明可以唤醒一个因调用了 wait 方法而休眠的线程。

(3)synchronized 模块首先从通道4向 monitor 发出了请求允许进入同步方法的消息，monitor 收到该消息后，经判断及调度，从通道4向 synchronized 模块返回一个 locking 消息，若管程中无活动线程，则 locking 消息将给临界区加锁，否则

刘 园 博士研究生，从事形式化工程和模型检查等方向的研究。徐宝文 教授，博士生导师，主要从事程序设计语言、软件工程、并行与网络软件、知识与信息获取技术等方面的教学与科研工作。

locking 消息拒绝同步方法进入管程的请求，并要求其等待。当一个同步方法从管程中退出时，synchronized 模块将从通道5向 monitor 发送一个 release 消息，表明该同步方法已释放管程的控制权。

(4)monitor 模块是总控模块，分别从通道1，2，3，4和5接收和发送相应的消息，进行最重要的调度工作，具体功能同 Java 管程的调度。

根据上述管程模型，下面给出 Java 并发系统的 ACP 模型的定义：

定义1 Java 并发系统的 ACP 模型可用一个6元组 $\Sigma=(M,T,W,N,S,\pi)$。

其中：M--Java 并发系统的所有管程集合。

T--对应于某一个管程 m 的所有线程集合为 T_m，$m\in M$，$T=\underset{m\in M}{Y} T_m$。

W--对应于某一管程 m 的某一线程中的 wait 方法为 $W_{m,t}$，$W=\underset{m\in M}{Y}\underset{t\in T_m}{Y} W_{m,t}$。

N--对应于某一管程 m 的某一线程中的 notify 或 notifyall 方法为 $N_{m,t}$，$N=\underset{m\in M}{Y}\underset{t\in T_m}{Y} N_{m,t}$。

S--对应于某一管程 m 的某一线程中的 synchronized 方法为 $S_{m,t}$，$S=\underset{m\in M}{Y}\underset{t\in T_m}{Y} S_{m,t}$。

π--π(W,N,S)为并发系统中 W,N 与 S 的偏序集。

下面给出该 ACP 模型的语义，其中 s_i 和 r_i 称为通信原子操作。

$\forall\, m\in M, t\in T_m$：

$$W_{m,t}=s_1(m,t)\cdot r_2(m,t) \tag{1}$$

☆wait 模块从通道1向 monitor 发出 wait 消息。

$$N_{m,t}=s_3(m,t) \tag{2}$$

☆notify(notifyall)模块从通道3向 monitor 发出 notify(notifyall)消息。

$$S_{m,t}=s_4(m,t)\cdot r_4(m,t)\cdot S'\cdot s_5(m,t) \tag{3}$$

☆synchronized 模块首先从通道4向 monitor 发出了请求允许进入同步方法的消息，再从通道4接收来自 monitor 模块的 locking 消息。若该同步方法能进入管程，则当该同步方法从管程中退出时，synchronized 模块将从通道5向 monitor 发送一个 release 消息。其中 S′是同步方法中的代码(模块)，通过这种方法，可以轻松实现嵌套。

$$M=(\sum_{m\in M,t\in T_m} r_1(m,t)+\sum_{m\in M,t\in T_m} s_2(m,t)+\sum_{m\in M,t\in T_m} r_3(m,t)+\sum_{m\in M,t\in T_m} r_4(m,t)+\sum_{m\in M,t\in T_m} s_4(m,t)+\sum_{m\in M,t\in T_m} r_5(m,t))\cdot \mathrm{Schedule}\cdot M$$

且 $r_5(m,t)<r_3(m,t)<s_2(m,t)<r_1(m,t)<s_4(m,t)<r_4(m,t)$, (4)

其中“<”为优先级操作符[5]，左边优先级低于右边优先级。

☆monitor 模块分别从通道1，2，3，4和5接收和发送相应的消息，进行调度，其中 Schedule 表示调度，是一个内部状态，可以通过封装操作将其移去。该表达式是一个递归表达式，因为管程是服务性的守护进程。由于 Java 规定这三种方法只能在同步方法中使用，并且对于一个特定的管程及一个给定的线程，上述的通信原子操作是有顺序的，因而有上述的优先级顺序。

$\partial_H(M\|\pi(W,N,S))$，其中 $H=\{s_i,r_i\,|\,i=1,2,3,4,5\}$ (5)

☆$M\|\pi(W,N,S)$表示整个 Java 并发系统。由于 s_i，r_i 是通信原子操作，属于内部操作，因而需要进行封装操作。

$$\upsilon_{\alpha(M)}\circ\partial_H(M\|\pi(W,N,S))=\sum_{m\in M,t\in T_m} r_4(m,t)\cdot s_4(m,t)\cdot r_1(m,t)\cdot s_2(m,t)\cdot r_3(m,t)\cdot r_5(m,t)\cdot\upsilon_{\alpha(M)}\circ\partial_H(M\|\pi(W,N,S)) \tag{6}$$

其中 $H=\{s_i,r_i\,|\,i=1,2,3,4,5\}$，$\circ$ 是函数或操作符的合成，$\alpha(M)$是 M 的字母表。

☆在式(4)中，由于诸通信原子操作需满足一定的时序，因而可对式(5)使用局部化操作 $\upsilon_{\alpha(M)}$，从而式(4)可重写成式(6)。

根据上述的 ACP 表达式(1)，(2)，(3)，(5)和(6)，即可构成 Java 并发系统的 ACP 模型。通过对 Java 并发程序代码(或设计阶段的诸模块)的扫描，很容易就能得到6元组 $\Sigma=(M,T,W,N,S,\pi)$。该 ACP 模型语义清晰，简洁直观地描述了 Java 管程机制。

结束语 本文采用了 ACP 对 Java 并发系统进行了建模。由于 ACP 具有很强的处理过程合成及隐藏内部细节的能力，因此以进程代数形式化系统为基础，可以方便地进行死锁检测、可达性分析、通信关系分析、控制流分析及模型检查等等。对于上述某些方法，我们正在研究[7,8]，今后将着重于模型检查。

参 考 文 献

1 Peterson J L. Petri Net Theory and the Modeling of Systems. Prentice Hall, Englewood Cliffs, N J. , 1981

2 Dwyer M B, Nies K A, Clarke L A. Compact Petri Net Representation for Concurrent Programs. In: Proc. Intl. Conf. Software Engineering, 1994

3 Gedela R K, Shatz S M, Xu Haiping. Compositional Petri Net Models of Advanced Tasking in Ada-95. Computer Language, 1999, 25(2): 55～87

4 Long D L, Clarke L A. Task Interaction Graphs for Concurrency Analysis. ICSE'89, 1989. 44～52

5 Baeten J C M, et al. Process Algebra. Cambridge University Press, Cambridge, UK, 1990

6 Baeten J C M. Applications of Process Algebra. Cambridge University Press, Cambridge, UK, 1990

7 Liu Yuan, Xu Baowen, Chen Zhenqiang. Detecting Deadlock in Ada Rendezvous Flow Structure Based on Process Algebra. LNCS 2495, 262～274

8 刘园，徐宝文. Ada 并发程序的进程代数描述. 南京大学学报，2002, 38(11): 14～18