维普资讯 http://www.cqvip.com

JAC技术实现生产者-消费者问题*)

朱 红 殷兆麟
(中国矿业大学计算机科学与技术学院 徐州 221008)

摘 要 JAC技术通过扩展JAVA注释实现并发,具有并发表达层次高、并发逻辑与应用逻辑分离、软件复用进一步加强等优点。论文利用JAC技术解决生产者-消费者问题,阐述了JAC技术的使用及其线程同步能力。

关键词 JAC,同步,生产者-消费者

To Solve the Producer-Consumer Problem Based on the JAC

ZHU Hong YIN Zhao-Lin
(Dept. of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221008)

Abstract JAC implements concurrency based on the extended JAVA comments, it introduces a higher level of concurrency, hiding threads and separating thread synchronization from application logic in a declarative fashion, thus furthering code reuse. The paper discusses JAC and its thread synchronization ability, and solves the classical process synchronization problem —the producer- consumer problem based on the JAC.

Keywords JAC, Synchronization, Producer- consumer problem

1 引言

生产者-消费者问题(producer-consumer problem)是典型的进程同步问题,人们讨论过多种不同的同步机制解决这一问题,目的是检验同步机制的能力。JAC(Java with Annotated Concurrency)技术采用注释来支持并发,使应用逻辑与并发逻辑分离,用它来解决生产者-消费者问题简洁方便。

2 JAC技术

JAC技术是由柏林自由大学 Klaus-Peter Löhr 教授、Max Haustein[1]等人开发的,主要应用于JAVA语言的并发编程。Java语言规范包括对线程和并发的明确支持,并提供了一些机制供开发人员使用以保证程序的线程安全,这些特点使得Java成为面向对象语言中对于多线程特性支持方面的佼佼者。然而随着计算机科学的发展,Java内建的并发机制已不适合实际复杂程序对并发编程的要求,它的缺陷日益突出:并发API过于简单,并发代码难以编写,更难以测试;并发模型不是面向对象的,一个Java编程语言线程实际上只是一个run()方法,由它进一步调用其它方法[2,3];应用逻辑与并发逻辑纠缠在一起,因此,导致程序逻辑结构混乱,难以复用和维护[4]。

JAC技术是基于对顺序Java代码扩展注释来描述并发特征的,它完全抛弃了Java原有的并发编程方式,隐藏了线程,减轻了继承异常。用JAC技术编写的Java程序只是纯粹的顺序化代码,并发由特殊的注释(即并发注解)来完成。普通的Java编译器忽略这些并发注解,而JAC预编译器将识别并发注解,并把它编译成Java编译器能识别的普通的Java并发代码。这样可以把应用程序逻辑和并行逻辑严格地区分开来,避免了逻辑混乱。由于不管在顺序化环境下(此时不使用JAC预编译器)还是在并发环境下代码完全相同,因此JAC技术使软件复用进一步加强。与Java不同,JAC中所有方法默认为必须互斥地执行,除非其前面有可并发执行的注解。

2.1 JAC技术的并发注解

JAC中,并发注解采用Java注释形式:

/* * @key [value]/

@表示这里是并发注解,而不是普通的Java注释。JAC预编译器将识别它。被注解的Java语言成分有:语句、方法、类。主要JAC并发注解介绍如下:

2.1.1 controlled 注解

格式:/* * @controlled */

功能:产生受并发注解控制的对象

示例:/* * @controlled */

```
Map map=new LinkedMap();
```

说明:注释规范下面的构造对象语句产生受并发注解控制的对象。

2.1.2 compatible 注解

格式:/* * @compatible SignatureList */

功能:列举出可与被注解方法并发执行的方法,表明它们可以并发执行

示例:/* * @compatible read1(), read2() */

```
public int read1() {…}
```

说明:SignatureList是由逗号隔开的类封装的可并发执行的方法表,compatible注解具有对称性,但不具有反身性。使用没有SignatureList的compatible注解,这个方法可以和所有使用同样注解的方法并发执行。

2.1.3 precondition 与 guard 注解

格式:/* * @if PreconditionExpression */

/* * @when GuardExpression */

*)中国矿业大学青年科研基金。朱 红 硕士研究生,主要研究方向为网络技术(Java并行);殷兆麟 教授,主要研究方向为网络技术(Java安全、Java软件测试、Java组件集成)。

维普资讯 http://www.cqvip.com

功能:用来在方法前注解此方法运行的前提条件

例子:/ * * @when length>0 * /

public Object remove(){……}

说明:当 Precondition 与 Guard 表达式均成立,方法才可以执行。Precondition 表达式不成立将产生异常。如果 Guard 表达式不成立,对受控制的对象,方法体的执行将延迟到 Guard 表达式为真,对不受控制的对象 Guard 被解释为 Precondition,也就是仍将产生异常。

2.1.4 async 注解

格式:/ * * @async * /

功能:用来注解一个方法,这个方法和它的调用者异步地执行。

例子:/ * * @async * /

public T method(…) {…}

2.1.5 auto 注解

格式:/ * * @auto * /

功能:用来注解一个方法,这个方法在对象被初始化之后不断地被重新调用。

说明:如果想在自治方法的自动调用中插入暂停可使用 delay 参数,延迟单位为 millisecond. 情况如下面的方法每秒被调用:

```
/**
  * @auto delay=1000
  */
public void beep(){…}
```

如果想终止自动调用,要使用 stop 注解。如:

```
/** @stop q*/
```

其中 q 为自动调用自治方法的对象。

2.2 JAC 预编译器原理

JAC 注解的程序由 JAC 预编译器编译成普通的 Java 代码,预编译器工作原理[5]如下:

JAC 预编译器首先导入用户定义的类,为构造方法增加一个布尔型参数 controlled_,为每个成员方法分配一个标识号 ID,并且依据方法前面的注解生成新的方法 new_method,而原有方法被称为 user_mehtod。在构造对象时,如果构造对象的语句前有 controlled 注解,则构造方法中的 controlled_参数初始化为 true,否则为 false。

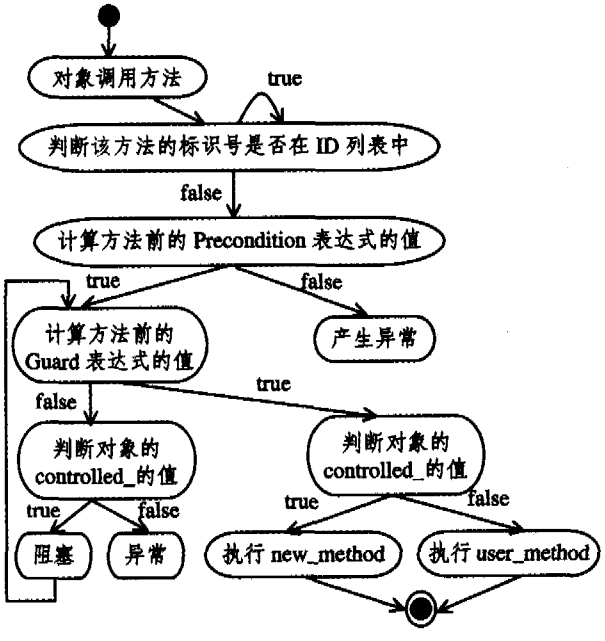


图 1 JAC 预编译器对象调用方法的过程

对象调用方法的过程如图 1 所示(程序运行时,JAC 预编译器为每个对象维护一张 ID 列表,它由不能与当前正在执行的方法并行的方法的 ID 号组成,在方法执行完毕或执行新方法时由 JAC 预编译器自动更新)。

由 auto 注解的方法叫自治方法。预编译器为自治方法产生一个线程,这个线程重复地调用此方法。由 async 注解的方法叫异步方法。预编译器为异步方法产生一个代理,并立即返回调用者。代理产生一个新线程执行异步方法,直到获得结果对象。自治方法和异步方法的执行和图 1 所示是一样的过程。

JAC 技术对于面向对象的并发的方法既具有高级别的抽象性——从应用程序逻辑中分离出并发声明,又具有轻便性——只是加一些注解给代码,这使得代码重用进一步加强。Java 线程模型不是面向对象的,一个 Java 编程语言线程实际上只是一个 run() 过程,它调用了其它的过程,这里根本没有对象、异步或同步以及其它概念。JAC 技术把并发活动的线程的分配留给了预编译器,使得面向对象 (OO) 设计人员可以完全抛弃线程,根本不从线程角度考虑问题,而只需考虑同步处理、异步处理,这样可以避免并发与对象模型之间一些概念的冲突,还可以最小化继承异常的危险。JAC 技术有广范的应用性,从高的交互系统到严格的程序到并行任务的计算。在解决经典进程同步问题上 JAC 技术同样具有显著优势。

3 JAC 技术实现经典进程同步问题

著名的生产者-消费者问题[6](producer_consumer problem)是典型的进程同步问题,JAC 技术可以解决好生产者-消费者问题就证明了它的同步能力。

生产者-消费者问题表述如下:有 n 个生产者和 m 个消费者,共享 $k+1$ 个单位的缓冲区。其中,p_i 和 c_j 都是并发进程,只要缓冲区未满,生产者进程 p_i 生产的产品就可投入缓冲区;类似地,只要缓冲不空,消费者进程 c_j 就可从缓冲区取走并消耗产品。

解决生产者-消费者问题有很多传统的方法,如信号量和 PV 操作、管程、消息传递等,但它们是较低级的同步机制。用 JAC 技术解决生产者-消费者问题程序如下:

(1)定义类可并发的方法

```
public class ProducerConsumer
{
    public final int k;
    protected final Object[] buffer; //产品缓冲区
    protected int front,rear; // front,rear 分别指示放置产品和取出
                                 产品的位置
    public ProducerConsumer(int k)
    {
      this. k=k;
      buffer=new Object[k+1]; //构造有 k+1 个单元的缓冲区
      }
    /**
       * @if x! =null //参数 x 不为空,x 为产品
       * @when front! =(rear+1)%(k+1) //缓冲区不满
       */
       public void producer(Object x)
       {
         buffer[rear]=x; //生产者将产品放入缓冲区
         rear=(rear+1)%(k+1); //放置产品的位置向前推进
       }
/**
    * @compatible producer(Object) //producer()与 consumer()并
                                      发执行
    * @when front! =rear //缓冲区不空
    */
    public Object consumer()
    {
      Object x=buffer[front]; //消费者取走产品消费
      front=(front+1)%(k+1); //缓冲区取出产品位置向前推进 1
      return x; //返回 x
```

维普资讯 http://www.cqvip.com

```
    }
}
```

方法 consumer()前有/* @compatible producer(Object)/注解，而且 compatible 注解具有对称性，所以方法 consumer()和方法 producer(Object)之间可以相互并发地执行。compatible 注解没有反身性，所以方法 consumer()与自身不能并发地执行。

如果缓冲区既不空也不满，并发执行这两个方法没有任何问题。但是，如果缓冲区是空的，执行方法 consumer()时，在检测到 Guard 表达式 front! =rear 不成立之后，方法 consumer()被阻塞之前，执行了方法 producer(Object x)，并在方法 consumer()被阻塞之前结束，那么新放到缓冲区的产品将被忽略。JAC 技术不会产生这样的问题，因为外部调用引起的状态变化会使被阻塞的方法的 Guard 表达式重新计算，也就是说，当执行了方法 producer(Object x)，返回它的调用者(对象)之后，被阻塞的方法 consumer()的 Guard 表达式 front! =rear 要被重新计算。同样道理，当缓冲区满时，方法 consumer()和方法 producer(Object)并发地执行也不会出现任何问题。

(2)JAC 技术实现生产者-消费者问题

下面的代码能正确地实现生产者-消费者问题：生产者不断地向缓冲区中写数据，消费者不断地从缓冲区中读数据。

```
public class PCTest
{
    ProducerConsumer q;
    int val =0
    public PCTest()
    {
        /** @controlled */
        q = new ProducerConsumer(8); //生产者-消费者对象
    }
        public static void main(String[] args)
    {
        /** @controlled */
        PCTest t = new PCTest();//构造受控制的 PCTest 类对象
        /** @sleep 6000 */
        System.out.println("done");
        /** @stop t */
        t=null; //此语句是技术上的需要，不是程序必需
    }
        /**
    * @auto delay Math.random() * 200 //生产者速率随机改变
    * @compatible
    */
    public void write()
    {
        System.out.println("writing:" + val);
        q.producer(new Integer(val++));
    }
    int oldread =-1;
    /**
     * @auto delay Math.random() * 200 //消费者速率随机改变
     * @compatible
     */
    public void read()
    {
        int read = ((Integer)q.consumer()).intValue();
        System.out.println("read: " +read + (read! =oldread+1?
        " ERROR":"") );
        oldread =read;
    }
}
```

预编译器为每个自治方法建立一个线程，所以上述程序建立了三个线程，实现生产者-消费者问题活动图如图 2 所示。

方法 write()和方法 read()都有注解/* @auto/，所以它们在对象被初始化之后不断地被重新调用，间隔时间是随机的。由于它们都有/* @compatible/注解，因此它们并发地执行。该程序向有 9 个单元的缓冲区不断地写数据 0,1,2,3,…，同时不断地读出来，约持续 6 秒。

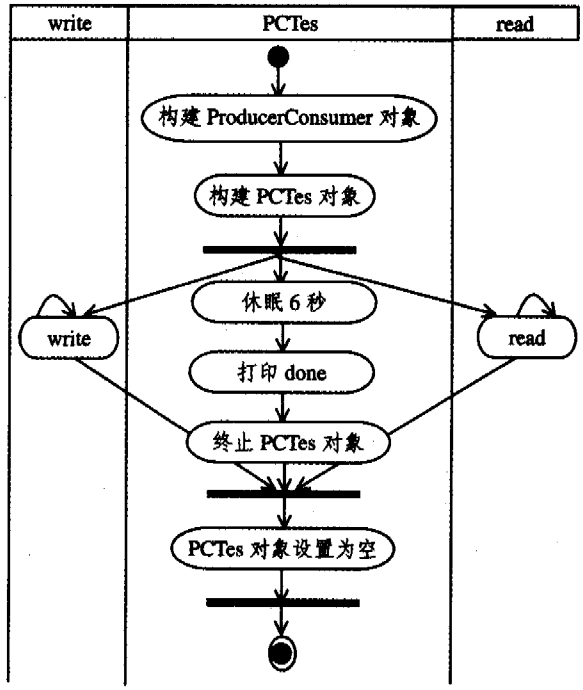


图 2 实现生产者-消费者问题活动图

如果不使用 JAC 技术，由于向缓冲区写数据和从缓冲区读数据的速率是随机改变的，很有可能它们访问缓冲区的速率不匹配，即生产者消费者进程的相对速度不协调，从而出现不正确的结果。使用 JAC 技术，ProducerConsumer 类的 Producer()和 Consumer()前有 Guard 表达式，它可以协调向缓冲区写数据和从缓冲区读数据的速率，从而正确地实现同步。

从程序运行结果可以看出，虽然向缓冲区写数据和从缓冲区读数据的速率是随机改变的，但写数据和读数据的顺序没有改变，读出的数据永远不会超过写入的数据，而且程序没有因向缓冲区写数据和从缓冲区读数据的速率的随机改变而发生任何错误，符合解决生产者消费者问题的要求。

如果有 n 个生产者，我们可以对 write()方法的 auto 注解进行如下扩充：/* @auto delay Math.random() * 200 thread n, val */，其中 n 表示自动生成的方法数，val 表示要修改的变量。这样在对象初始化之后，会产生 write1(), write2(), …, writen()共 n 个自治方法，它们不断地被重新调用，并且并发地执行。原来 write()方法内的局部变量 val 会依次转换为 val1, val2, …, val$n$，并且初始值均为 0。如果有 m 个消费者，扩充方法是相同的。

```
public class PCTest
{ ProducerConsumer q;
  int val =0;
  public PCTest()
  {
    /** @controlled */
    q = new ProducerConsumer(8); //产生一个受控制的生产者消
                                   费者对象
  }
     public static void main(String[] args)
  {
    /** @controlled */
    PCTest t = new PCTest();
    /** @sleep 6000 */
    System.out.println("done");
    /** @stop t */
    t = null;//此语句是技术上的需要，不是程序必需
  }
    /**
  * @auto delay Math.random() * 200 thread n,val
  * @compatible
  */
  public void write()
```

维普资讯 http://www.cqvip.com

```
{
  System.out.println("writing:" + val);
  q.producer(new Integer(val++));
}
/**
 * @auto delay Math.random() * 200 thread n
 * @compatible
 */
public void read()
{
  int read = ((Integer)q.consumer()).intValue();
}
}
```

从前面分析来看，任何 write()方法与任何 read()方法并发执行都是没有问题的，所以上面的程序中 read()方法中的检测语句 System.out.println("read:" + read + (read! = oldread+1? "ERROR":"")); 与 oldread = read; 去掉了。

那么 write1(), write2(), …, writen()之间并发执行会有问题吗？JAC 技术的方法如果没有注明是可并发执行的，默认为它们必须互斥地执行。由于 write1(), write2(), …, writen()之间并发执行实际上是并发地调用 producer(Object x)方法，producer(Object x)方法前没有 /* @compatible producer(Object)/ 注解，说明 producer(Object x)方法自身必须互斥地执行，所以不会出现这样的情况：一个 write 方法调用 producer(Object x)方法还没结束，另一个 write 方法又调用它，从而使得 producer(Object x)并发地执行。所以 write1(), write2(), …, writen()之间并发执行不会有问题。对于 read 方法也是一样。

结束语 JAC 技术是一种并行表达层次较高的并发模型，它只是加一些并发注解给代码，使得代码重用进一步加强，同时可以避免并发与对象模型之间一些概念的冲突，而且它做到了应用程序逻辑与并发逻辑的分离。生产者-消费者问题(producer consumer problem)是典型的进程同步问题，用 JAC 技术解决生产者-消费者一类的进程同步问题简洁、清晰、程序可读性强。JAC 技术也适合解决其它的进程同步问题，如：读者写者问题、哲学家进餐问题、理发师问题等，同样简洁、方便。

感谢德国柏林自由大学数字信息系教授 Klaus-Peter Löhr 为我们的合作研究提供帮助。

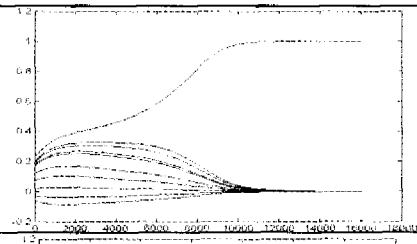
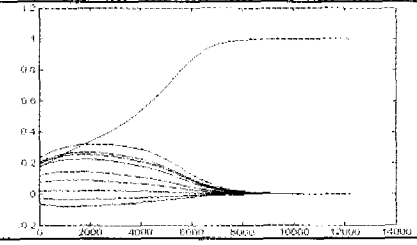
参 考 文 献

1 Haustein M. Java with annotated concurrency[EB/OL]. http://page.mi.fu-berlin.de/~haustein/jac/
2 Holub A. Taming Java Threads[M]. Apress, San Francisco, CA, 2000
3 Holub A. If I were King: A proposal for fixing the Java programming language's threading problems[EB/OL]. www-128.ibm.com/developerworks/library/j-king.html? dwzone = java , Oct 2000
4 邓辉，孙明. 构建 Java 并发模型框架[EB/OL]. http://www.uml.org.cn/sjms/200410221.htm
5 Haustein M, Löhr K-P. JAC-Declarative Java Concurrency. Concurrency and Computation: Practice and Experience, 2006, 18
6 孙钟秀，等. 操作系统教程[M]. 北京：高等教育出版社，2003

(上接第 254 页)

余注意参数；如果待识别模式的性别特征不明显，则所有注意参数取值相同。取平衡注意参数时，对文中列出的 10 个原型模式，用 20 幅试验模式，只有一幅识别错误。表 2 显示，对于平衡注意参数时的误识别，取非平衡注意参数时能正确识别试验模式。

表 2 两种情况注意参数的不同结果

注意参数 ξ_k	试验模式	被识别出的模式	ξ_k 的演化过程(横轴表示迭代步数，纵轴表示 ξ_k)
平衡注意参数			
非平衡注意参数			

结束语

本文针对静态人脸识别设计了一种简单易行的协同算法。它不像传统方法一样需要面部特征的准确配准；当系统中的图片有增减时，算法可以快速获得新的原型向量和伴随向量；对注意参数 λ_k 的两种情况的设置使算法的识别效果较单一平衡注意参数更好。仿真实验中得到了较好的识别效果。

参 考 文 献

1 Yang M-H, Kriegman D J, Narendra Ahuja. Detecting Faces in Images: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(1): 34~58
2 Zhao W, Chellappa R, Phillips P J, et al. Face Recognition: A Literature Survey. ACM Computing Surveys, 2003, 35(4): 399~458
3 周杰，卢春雨，张长水，李衍达. 人脸自动识别综述. 电子学报，2000，28(4)
4 崔国勤，高文. 基于双层虚拟视图和支持向量的人脸识别方法. 计算机学报，2005(3)：368~370
5 王和勇，姚正安，李磊. 基于聚类的核主成分分析在特征提取中的应用. 计算机科学，2005，32(4)：64~66
6 毛缜红，范伟，谭铁牛. 融合全局与局部特征的子空间人脸识别算法. 计算机学报，2005(3)：1657~1663
7 山世光，高文，唱铁钲，等. 人脸识别中的"误配准灾难"问题研究. 计算机学报，2005(5)：782~791
8 Haken H. 杨家本，译. 协同计算机和认知，清华大学出版社，1994
9 陈丽，戚飞虎. 基于梯度动力学的协同神经网络学习算法的改进. 计算机工程与科学，2005，27(11)
10 胡栋梁，戚飞虎. 协同模式识别中不平衡注意参数的研究. 电子学报，1999，27(5)：15~17
11 何光渝，高永利. Visual Fortran 常用数值算法集. 科学出版社，2002
12 Face Recognition by goksel@gazi.edu.tr