异步接口模式及其应用*)

Asynchronous API Pattern and its Application

何华海　丁　柯

(中国科学院软件研究所计算机科学重点实验室　软件工程技术中心　北京100080)

Abstract In distributed systems, high efficiency can be achieved using asynchronous API between client and server. This paper provides an architectural pattern that implements asynchronous API generally. Asynchronous methods do not execute operations directly, however, they delegate the sending and receiving process to individual threads via a queue, the client deals with results by means of callback, wait or check. Synchronous API is implemented on the base of asynchronous API. Presently the asynchronous API pattern has been employed in the implementation of message queue middleware ISMQ.

Keywords Design pattern, Synchronism/Asynchronism, Concurrency, Message queue

1. 引言

随着 Internet 的飞速发展，计算机硬件在性能上不断提高，新的数字设备和便携装置不断涌现并以惊人的速度互连。在这种网络分布式环境下，人们对性能、可靠性、互连性、可扩展性等方面提出了更高的要求，这使得网络分布式软件越来越复杂，开发周期越来越长[1]。传统的分析方法和工具已经难以适应网络分布式软件的开发。

设计模式[2]能有效地减缓开发网络分布式系统所面临的压力。在分析和设计软件系统的过程中，某些问题带有普遍性，对这些问题的解决方案也需要具有一定的通用性。设计模式是对这类问题解决方案的一种抽象。通过使用设计模式，已有的软件结构可直接重用来解决同类问题，提高了软件开发效率。目前人们已经为网络分布式系统总结了若干设计模式，这些模式可应用于组件动态配置、事件处理、同步和并发控制等场合[3]。

在客户/服务器计算范型中，异步接口能有效地提高客户应用系统的执行效率。由于实现异步接口涉及到线程间交互、异步操作的执行与管理等一系列问题，不同的客户/服务器应用系统的开发者不得不重复设计和实现异步接口。我们针对异步接口相关问题的解决方法进行抽象，提出了一种实现异步接口的体系结构模式，以及通用的异步操作实现机制，并在保证系统内部执行效率的前提下同时支持同步和异步接口。同步接口可简化客户端程序设计，而灵活的异步接口则为构造高效的应用系统提供支持。

2. 异步接口模式

2.1 设计目的

在客户/服务器应用系统中，客户以接口调用的形式获得服务器提供的各种服务。由于网络传输等原因，客户从发出请求到服务器返回结果会有一定的延迟，如果客户使用同步接口，将不得不阻塞等待执行结果。即使系统内部采用了有效的异步方式，客户应用系统仍然是低效的。为了提高效率，在某些应用中需要把异步方式开放给客户。

实现异步接口的软件体系结构需要解决以下问题：1)异步操作如何执行?2)客户如何处理异步操作的执行结果?3)多个异步操作如何管理?异步请求和执行结果之间如何对应?

我们使用独立的收发线程执行具体的异步操作，发送线程往服务器传输异步请求参数，接收线程则接收返回结果并通知客户。客户线程与收发线程通过队列进行交互。对于异步操作结果，客户采用回调、等待或者查询的方式处理。每个异步请求用一个异步操作描述符表示，描述符同时负责异步操作的生命期管理。同步接口则在异步方式之上实现。在客户多线程的情况下，多个异步请求在队列上顺序化，避免了对网络通道的竞争。

2.2 异步接口模式的结构

异步接口模式的结构如图1所示，它由以下部分组成：

·接口层(API_Impl)　封装了各种同步和异步接口方法的实现。

对于同步方式，接口方法实例化一个异步操作描述符并将其插入异步操作队列，然后等待该异步操作的完成。只有当异步操作完成时，接口方法才返回。

对于异步方式，接口方法实例化异步操作描述符并插入队列，然后立即返回。这时异步操作还未完成，因此客户需要以其它方式处理结果。常用的处理方式为回调、等待或者查询。回调方式是指客户在调用异步接口时提供一个返回结果的处理方法——回调函数，当异步操作完成时，由系统自动执行这个回调函数。等待或查询方式是指客户调用异步接口之后，继续执行后继操作，然后在某个汇聚点等待或查询该异步操作的完成。

·异步操作描述符(AsyncToken)　是异步操作的内部数据表示，同时它负责异步操作的生命期管理。如图2所示，接口层把异步操作描述符插入队列之后，异步操作进入“发送前”状态，此后接口层或客户便可以等待或者查询该异步操作的完成；发送线程把异步请求发送给服务器之后，异步操作进入“发送后”状态，这时接收线程准备接收返回结果；接收线程收到返回结果之后，异步操作进入“接收后”状态，这时接收线程根据结果处理方式通知该异步操作已经完成，或者执行该异

*)本文研究得到国家自然科学基金重点项目(69833030)、国家973基金项目(G1999035807)资助。何华海　硕士生，主要研究领域为分布式计算，中间件。丁　柯　博士生，主要研究领域为分布式计算，分布事务处理技术。

步操作的回调函数，异步操作至此结束。

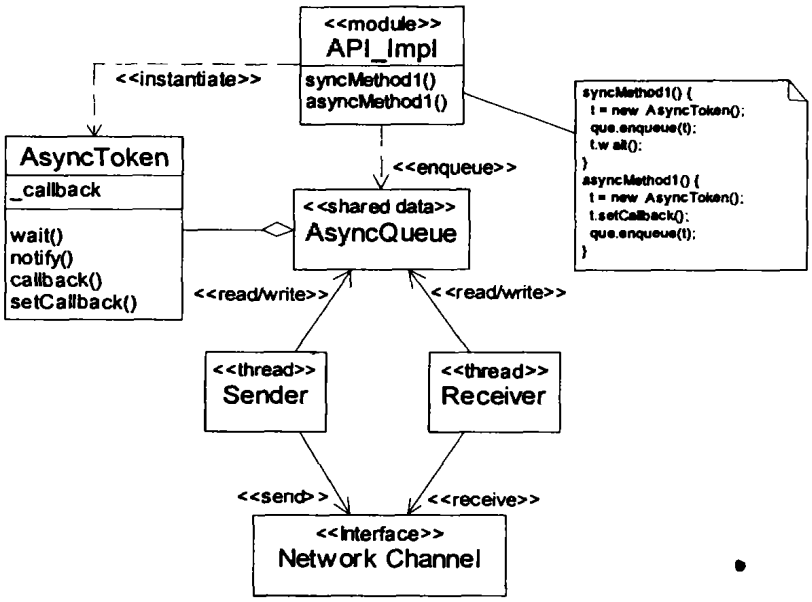

图1　异步接口模式的结构

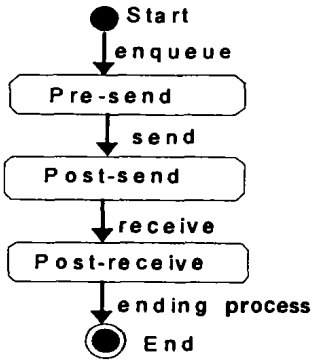

图2　异步操作的状态

·异步操作队列(AsyncQueue)　聚合了异步操作描述符，是接口层、发送线程和接收线程的联接点。在队列上，接口层插入具体的异步请求；发送线程从队列中读出异步请求并发给服务器，同时修改异步操作的状态；接收线程从服务器接收到返回结果，然后从队列中找出对应的异步操作，并进行返回结果处理。通过异步操作队列，各个独立运行的线程得到了有效的交互。

·发送线程(Sender)　执行具体的发送操作。当异步操作队列中有了新的异步请求，发送线程就从队列中读出该请求并通过网络接口发给服务器。接口层与发送线程在异步操作队列上是生产者/消费者关系，可以利用信号量机制来同步接口层和发送线程。当异步操作队列没有新的异步请求时，发送线程阻塞。

一般而言，客户和服务器之间的交互需要各自遵守某种协议，为了获得通用性，这种协议的设计应该与平台及实现语言无关。

·接收线程(Receiver)　监听网络端口并执行接收操作。接收过程依赖于具体的异步操作类型，当接收完一个返回结果时，接收线程就从异步操作队列中找出对应的异步操作描述符，根据结果处理方式通知该异步操作已完成或者执行该异步操作的回调函数。当网络端口没有数据时，接收线程阻塞。

·网络通道(Network Channel)　为客户和服务器提供网络通信接口。根据不同的应用背景，异步接口模式既可以采用面向流的通信方式，也可以采用面向数据报的通信方式。

2.3　交互过程

以回调方式的异步接口为例，异步接口模式中各个成员的交互过程如图3所示。

交互过程分为三个阶段：

·发送阶段　在发送阶段，接口层实例化一个异步操作描述符并把它插入异步操作队列，在实例化过程中，根据客户程序传入的参数设置回调函数。发送线程从异步操作队列中读出新的异步操作描述符，并根据客户/服务器交互协议把它发给服务器，同时修改异步操作描述符的状态。

如果是同步接口，接口层把描述符插入队列后阻塞调用线程，直到异步操作完成。

·接收阶段　根据客户/服务器交互协议从服务器接收返回结果，并检索异步操作队列，找出对应的异步操作描述符。

·返回结果处理阶段　接收线程根据预先设定的结果处理方式进行不同的处理。对于回调方式，接收线程直接执行或者在一个新的线程中执行回调函数；对于等待方式，接收线程通知这个异步操作已完成，使正在等待该异步操作的接口层或客户程序被唤醒，异步操作周期结束。对于查询方式，在接收线程通知前查询结果为未完成状态，通知后查询结果为完成状态。

2.4　实现中的考虑

·接口层和收发线程之间的同步　接口层和发送线程通过队列交互，可以利用信号量机制同步。异步操作描述符的等待和通知操作可以利用条件变量来实现。条件变量是操作系统提供的一种线程同步对象(在 Windows 中为 Event Object，在 UNIX 中为 Condition 变量)，它有两个状态：有信号和无信号，以及两个操作：等待和通知。等待操作在无信号时将阻塞调用线程，直到有信号时才唤醒调用线程；通知操作则把

条件变量设成有信号状态。

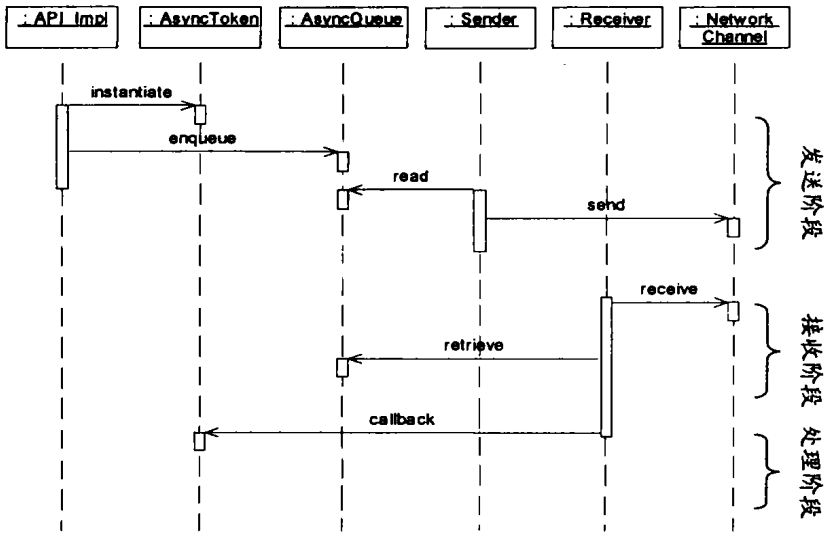

图3 异步接口模式的交互过程

·参数传递 接口层需要把异步操作参数传递给收发线程，但是收发线程都是主动对象，不可能直接通过函数调用的方式传递，此外，各种异步请求的操作参数也不相同，所以需要在异步操作描述符中定义匿名的数据结构来传递参数。

·返回结果与异步请求之间的对应 由于发送过程和接收过程是分离的，所以接收线程必须确定返回结果所对应的异步请求。可以用一个整数来唯一标识异步操作描述符。在发送请求时，将这个标识作为一个参数发给服务器端，服务器返回的结果中也包含这个标识，接收线程根据这个标识就可以从队列中检索出对应的异步操作描述符。

·服务器端的支持 为了在客户/服务器结构中实现完整的异步功能，服务器也需要提供支持。客户和服务器之间一般通过 Acceptor-Connector 模式[3]进行连接。对每一个客户连接，服务器都有一个连接线程监听并处理客户请求。对于同步请求，服务器的连接线程可以串行处理并阻塞，因为客户在当前请求完成之前不会发出下一个请求。但是对于异步请求，连接线程如果阻塞将无法及时处理客户发出的后继请求。在某些情况下，多个请求之间有时序依赖，例如，某个请求必须在其后继请求完成之后才可能完成，如果连接线程阻塞在该请求，将发生死锁。所以，对于可能造成阻塞的异步请求，连接线程必须考虑新的策略。

一种策略是把异步请求放在单独的异步处理线程中执行。为了提高效率，异步操作线程可以用异步操作线程池管理，连接线程遇到异步请求时从线程池中申请一个异步处理线程，把异步请求传给该线程处理，连接线程则继续监听客户请求。异步操作线程池如图4所示。

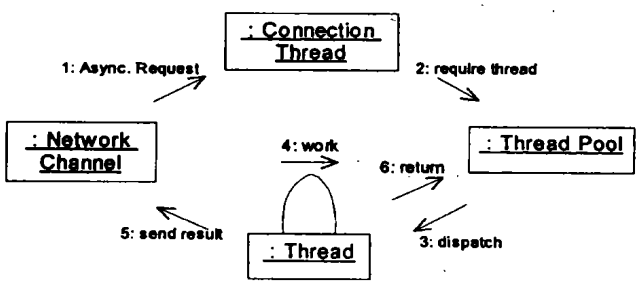

图4 异步操作线程池

2.5 优缺点分析

异步接口模式的实现不依赖于任何特定的操作系统，具有较好的通用性。客户使用同步接口能简化编程，使用异步接口能有效提高客户应用系统的执行效率。由于发送过程和接收过程分离，并通过队列缓存操作请求和返回结果，所以在客户多线程的环境下，多个异步操作可以重叠执行，保证了系统的并发性。

异步接口模式还可以实现事件方式的接口。在事件方式中，用户首先向服务器注册一个事件处理例程，此后每当服务器发生该事件时就通知客户端并发回事件数据，在客户端，接收线程充当事件源，执行事件处理例程。

异步接口模式会有一定的系统开销。发送线程和接受线程是独立运行的两个线程。队列的维护也会增加系统开销，每个异步操作必须分配一个异步操作描述符。此外，接口层将异步操作委托给收发线程分别执行，这在一定程度上破坏了异步操作的完整性。

3. 异步接口模式的应用情况

异步接口模式已被成功应用于消息队列中间件 ISMQ (Institute of Software Message Queue)[4]。消息队列中间件在异构平台上提供了一种松耦合的、可靠的消息通信机制，利用消息队列服务器的临时存储和转发功能，位于不同节点的分布组件获得了间接的通信能力而无须考虑具体的网络协议。ISMQ 是一种典型的客户/服务器结构，应用程序通过调用 ISMQ 客户端的链接库获得消息队列服务。基本的消息队列接口包括连接服务器、打开/关闭队列，发送/接收消息。接收消息有同步和异步两种方式。异步接口采用回调、等待或查询方式处理返回结果。

如图5所示，ISMQ 应用结构包括三个部分：ISMQ 应用程序、ISMQ 链接库和 ISMQ 服务器。ISMQ 应用程序通过 ISMQ 链接库使用消息队列服务。ISMQ 链接库在初始化时建立与服务器的连接并创建收发线程。API_Impl 是 ISMQAPI 的实现，它把接口参数直接插入异步操作队列，发送线程从异步操作队列读出异步请求并往服务器发出请求，接收线程从网络接口接收返回结果，根据异步操作的标识从异步操作队列中找到对应的异步操作，然后进行返回结果处理。

如果异步操作是回调方式，则执行回调函数。如果异步操作是等待/查询方式，则通知该异步操作的条件变量。在 ISMQ 服务器端，消息队列管理器为每个客户建立一个连接线程，接收客户请求并执行。对于异步请求，连接线程采用图4所示的异步操作线程池处理。

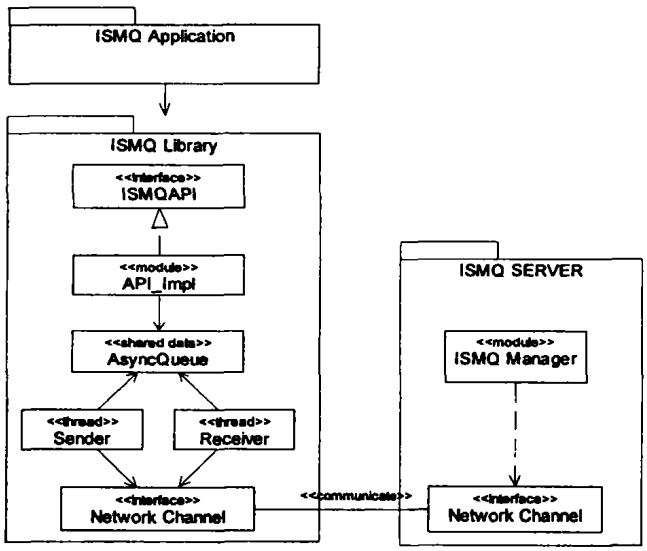

图5　异步接口模式在 ISMQ 中的应用

4. 相关模式

异步接口模式工作在多线程环境下，是一种并发模式。目前人们已经提出了若干种并发模式，Active Object 模式[5]使客户可以访问运行在独立线程中的对象，方法的调用发生在调用者所在线程，而方法的执行则发生在对象自己的线程，这样可以提高并发度，同时简化了对象之间的同步。Half-Sync/Half-Async[6]模式把同步操作和异步操作结合在同一个并发系统中，在系统高层使用同步操作，在底层使用异步操作，同步操作和异步操作通过队列交互，这样既简化了用户编程又保证了系统的执行效率。Leader/Followers 模式[7]为多事件驱动系统提供了一种有效的并发模型，在该模型中，多个线程轮流监听事件源集合，这样多个服务请求可以同时被有效地处理。

异步接口模式的结构与 Half-Sync/Half-Async 模式有若干的共同点，两者都分为上下两层，上下层通过队列进行交互。但是 Half-Sync/Half-Async 模式的目的是将 I/O 中断之类的异步操作转换成同步接口，为客户提供编程简洁性。而异步接口模式则将异步方式开放给客户，提高客户应用系统的执行效率。Half-Sync/Half-Async 模式主要应用于各种操作系统，异步接口模式可以应用于中间件等客户/服务器软件。

异步接口模式的实现依赖于其它设计模式。异步操作队列采用 Mediator 模式[2]协调接口层与收发线程之间的交互；客户与服务器之间用 Acceptor-Connector 模式建立连接。

小结　网络分布环境对软件开发方法和技术提出了更高的要求，设计模式旨在重用对某类问题的通用解决方案，能有效提高分布式软件的开发效率。在网络分布计算中，客户/服务器之间采用异步方式可以有效提高系统的执行效率。本文

（下转第182页）

怎样编写中、英文摘要

摘要是科技论文的重要组成部分，是一种以提供文献内容梗概为目的，不加评论和补充解释，简明、确切地记述文献重要内容的短文．其基本要素包括研究的目的、方法、结果和结论。摘要可大致分为 3 种类型：报道性摘要、指示性摘要、报道一指示性摘要。

报道性摘要　是指明一次文献的主题范围及内容梗概的简明摘要，相当于简介。报道性摘要一般用来反映科技论文的目的、方法或定量的信息，充分反映该研究的创新之处。

指示性摘要　是指明一次文献的论题及取得的成果的性质和水平的摘要，其目的是使读者对该研究的主题的主要内容（即作者做了什么工作）有一个轮廓性的了解。

报道-指示性摘要　这种摘要界于上述两者之间，以报道性摘要的形式表述一次文献中信息价值较高的部分，而以指示性摘要的形式表述其余部分。

1、中文摘要一般不宜超过 200-300 字，外文摘要不宜超过 250 个实词。如遇特殊需要字数可以略多（为了扩大国际影响，英文摘要尽量写长一些，行文须符合英语习惯，不必与中文摘要一一对应）。

2. 除了实在无变通办法可用以外，摘要中不用图、表、化学结构式、非公知公用的符号和术语。

3. 英文时态以简练为佳、常用一般现在时、一般过去时。

4. 摘要中过去多用第三人称 This paper…等开头，现在倾向于采用更简洁的被动语态或原形动词开头。

5. 行文时最好不用第一人称，以方便文摘刊物的编辑刊用。

总之，好的摘要既能使读者了解论文的主要内容，又能为科技情报人员和计算机检索提供方便。可以这样说，摘要质量的高低，直接影响着论文的被录用情况和期刊的知名度。

送的消息。一开始，他们被初始化为 v_i 在第一阶段中从列的方向上得到的消息（如果是在列上做线形 gossiping，则将他们初始化为 vi 在第一阶段中从行的方向上得到的消息）。我们只关注左边半行的执行，右边半行是相似的。算法在左半部分的执行可以看作由三个子操作完成，一个子操作是 $v_0, v_1, \cdots, v_{c-1}$ 的消息传送到 v_c（途经所有的中间结点），第二个子操作是 $v_{c-1}, \cdots, v_1$ 的消息传送到 v_0（途经所有的中间结点），第三个子操作是 v_c 积聚的 $v_c, v_{c+1}, \cdots, v_{n-1}$ 传送到 $v_0, v_1, \cdots, v_{c-1}$。

对于 n 为奇数的情况，由于在每一列中奇结点和偶结点的个数不同，导致第一阶段结束后行上的结点从列的方向上得到的消息个数是不同的：或者偶结点得到的消息数比奇结点得到的消息数多一个，或者相反。我们来研究前一种情况，（后者情况基本相似），偶结点有 $(n+1)/2$ 个消息，奇结点有 $(n-1)/2$ 个消息。

```
◇Ri=Li={vi 在第一阶段中从列的方向上得到的消息}0≤i≤n-1;
◇node vi,i<c,位于中央结点左边的结点
  Repeat(以下的两个 if 语句并行执行)
  ·If Ri≠ϕ
        将 Ri 中的第一个消息发送到 vi+1并且将这个消息从 Ri 中删除
   else
        从 vi+1(Li+1)得到一个消息并将这个消息加入 Li(if i>0);
  ·if vi-1传来一个消息
   将此消息加入 Ri;
  until done;
◇node vi,i>c,位于中央结点右边的结点
  repeat
  (与位于中央结点左边的结点的算法相似，只是所有方向相反)
  until done;
◇中央结点 Vc
  (以下两个 repeat 语句并行执行)
  ·repeat
        从左边获取一个消息并将这个消息加入 Rc;
   until 没有可以获得的消息(Rc-1=ϕ)
   ·repeat
        从右边获取一个消息并将这个消息加入 Lc;
   until 没有可以获得的消息(Lc+1=ϕ)
```

图2　第二阶段线形 Gossiping 算法

图3显示了在 n=7时第2阶段的执行情况，我们只列出了左半行的执行情况，因为左右是完全对称的。在该图中，大的长方形含有 $(n+1)/2$ 个消息，小的长方形含有 $(n-1)/2$ 个消息。从图中可以计算出第2阶段所用的时间片为：

$$(n+1)^2/2+(n-1)^2/2+(n-1)/2-1=(n^2+n-2)/2=S_{odd}$$

上面式子中的第一项对应图中的大长方形，第二项对应图中的小长方形，第三项对应空心圆点。

用同样的方法，我们可以计算出 n 为奇数时另一种情况下（偶结点在第一阶段得到的消息比奇结点得到的少一个）第2阶段所需的时间片：$(n^2+n-4)/2=S'_{odd}$

明显的，$S_{odd}>S'_{odd}$，所以第二阶段线形算法在所有行和列上执行所需的时间片为 S_{odd}。

在 n 为偶数的情况下，第一阶段结束后，偶结点和奇结点得到的消息数是一样的，都为 n/2个。因此，我们可以很容易得到 n 为偶数时第二阶段的执行时间为：

$$n*n/2+n/2-1=(n^2+n-2)/2=S_{even}=S_{odd}。$$

综合以上两个阶段，该算法总共所需的时间片为 $S_m=S_{第一阶段}+S_{第二阶段}=(n^2+3n-4)/2$。.

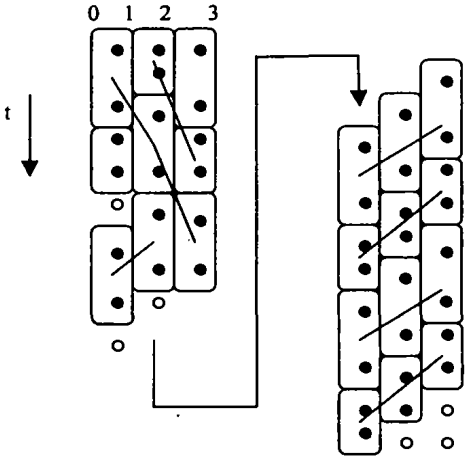

图3　n=7时第2阶段执行情况（只显示左半边）

总结　在本文中，我们给出了 square mesh 中的 gossiping 算法，时间复杂度为 $1/2O(n^2)$，与我证明的时间复杂度是一样的。但是这个算法还不是最优的，主要因为第一和第二阶段的耦合不够紧密。

参考文献

1　Bermond J-C, Gargano L, Rescigno A A, Vaccaro U. Fast gossiping by short messages. SIAM J. on computing, 1998, 27: 917～941

2　Krumme D W, Fast Gossiping for the Hypercube, SIAM J. Computing, 1992, 21(2): 365～380

（上接第180页）

提出的异步接口模式抽象了异步接口的一般性实现方法，在独立于特定操作系统的前提下能构造出同时支持同步和异步接口的客户端软件。在异步接口模式中，异步请求通过队列委托给独立的收发线程执行，对于返回结果，客户端可采用回调、等待或查询等方式处理。异步接口模式可用于构造灵活高效的异步编程接口。目前，异步接口模式已经在消息队列中间件的实现中得到了成功的应用。

参考文献

1　Coulouris G, Dollimore J, Kindberg T. Distributed Systems: Concepts and Design. Addison-Wesley, 2000

2　Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1995

3　Schmidt D C, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects. John Wiley & Sons, 2000

4　中国科学院软件研究所软件工程技术中心报告. 消息队列中间件 ISMQ 的设计和实现. 2000

5　Lavender R G, Schmidt D C. Active Object: an Object Behavioral Pattern for Concurrent Programming. in Pattern Languages of Program Design (J. O. Coplien, J. Vlissides, and N. Kerth, eds.), Addison-Wesley, 1996

6　Schmidt D, Cranor C. Half-Sync/Half-Async -- An Architectural Pattern for Efficient and Well-structured Concurrent I/O. In: Proc. of the 2nd Pattern Languages of Programs conf. 1995

7　Schmidt D C, et al. Leader/Followers: A Design Pattern for Efficient Multi-threaded Event Demultiplexing and Dispatching. In: proc. of the 7th Pattern Languages of Programs Conf. Aug. 2000