计算机软件工程

质量保证与产品可靠性评审测试技术新标准实务全书

◎ 主编 王志锦 罗乔欣

本手册不单独定价出售，随光盘附赠

计算机软件工程质量保证与产品可靠性评审测试技术新标准实务全书

主编　王志锦　罗乔欣

第三卷

水线缓冲区代替单个的缓冲区任务。稍后我们将讨论流水线缓冲区。

（2）多个任务与单一任务相互作用。

现在我们考虑多个任务与一个单一任务之间的通讯问题。如果有几个任务准备向一个任务发送数据，如图4－2－56的图（a）所示。对于这个问题如何处理呢？再一次，有两种主要的相互作用是可能的：如图（b）所示目标任务作为调用者，或者如图（c）所示发送任务作为调用者。

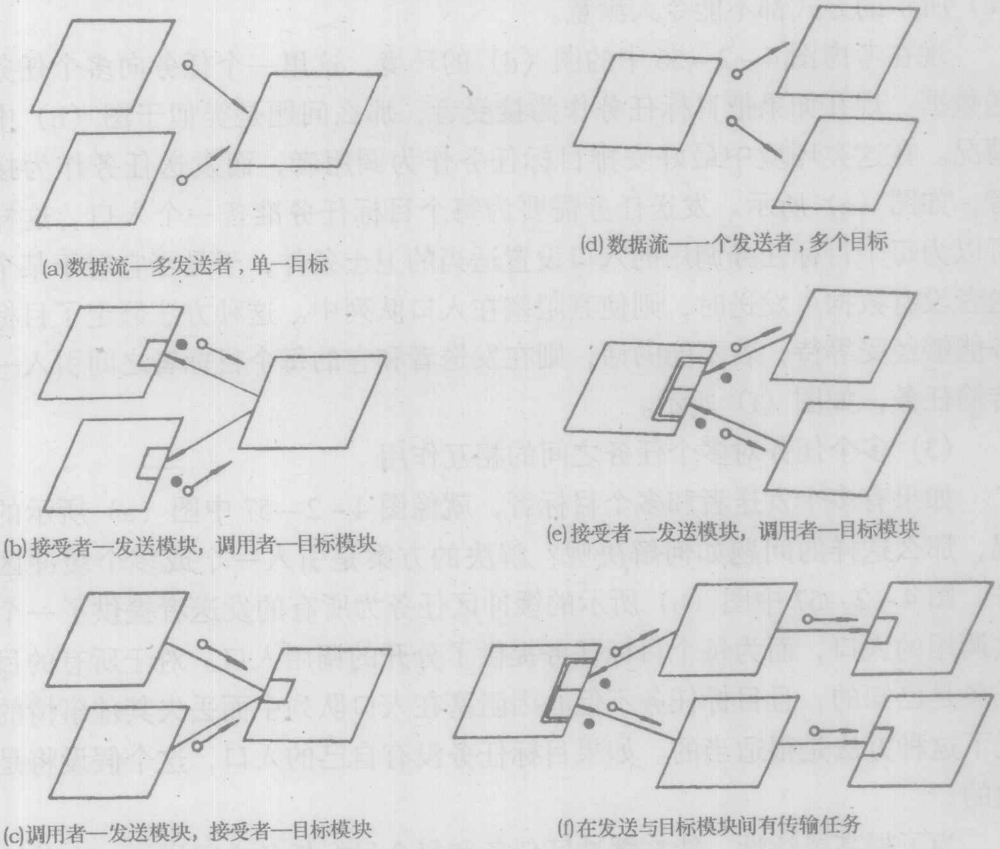


图4－2－56 多个任务与单一任务相互作用结构

这里的情形不同于单一任务对单一任务的相互作用，现在即使没有进一步的信息，这两种方式也存在着显著的差别。对于图（b），目标任务首先必须知道所有的发送者，其次它必须决定调用次序。一旦它选定了一个调用次序，当它对一个发送者作调用时，若该发送者没有准备好，那么它将阻塞在

这个发送者的入口队列中，此时即使有其他的发送者准备好了数据，它也不能去调用。另一方面对于图（c），目标者作为接受者，此时它不必知道所有的调用者，只要调用者知道它。并且任何一个发送者首先发送，它都能接受，没有被阻塞的危险。但是它应该保证它不能有其他活动而不必要地使发送者等待。这是一个比图（b）所示的更好的一个方案。无论如何，如果目标任务经常做其他工作而不能及时接受发送者的调用，那么图（b）和图（c）所示的方式都不能令人满意。

现在考虑图4-2-56中的图（d）的环境，这里一个任务向多个任务发送数据。现在如果把目标任务作为接受者，那么问题是类似于图（b）中的情况。在这类环境中最好安排目标任务作为调用者，而发送任务作为接受者，如图（e）所示。发送任务需要为每个目标任务准备一个入口，这样它可以为每个目标任务调用的入口设置适当的卫士条件，于是当它对于某个发送者没有数据可发送时，则使其阻塞在入口队列中。这种方法假定了目标任务能够经受等待，若不能的话，则在发送者和它的每个目标者之间引入一个传输任务，如图（f）所示。

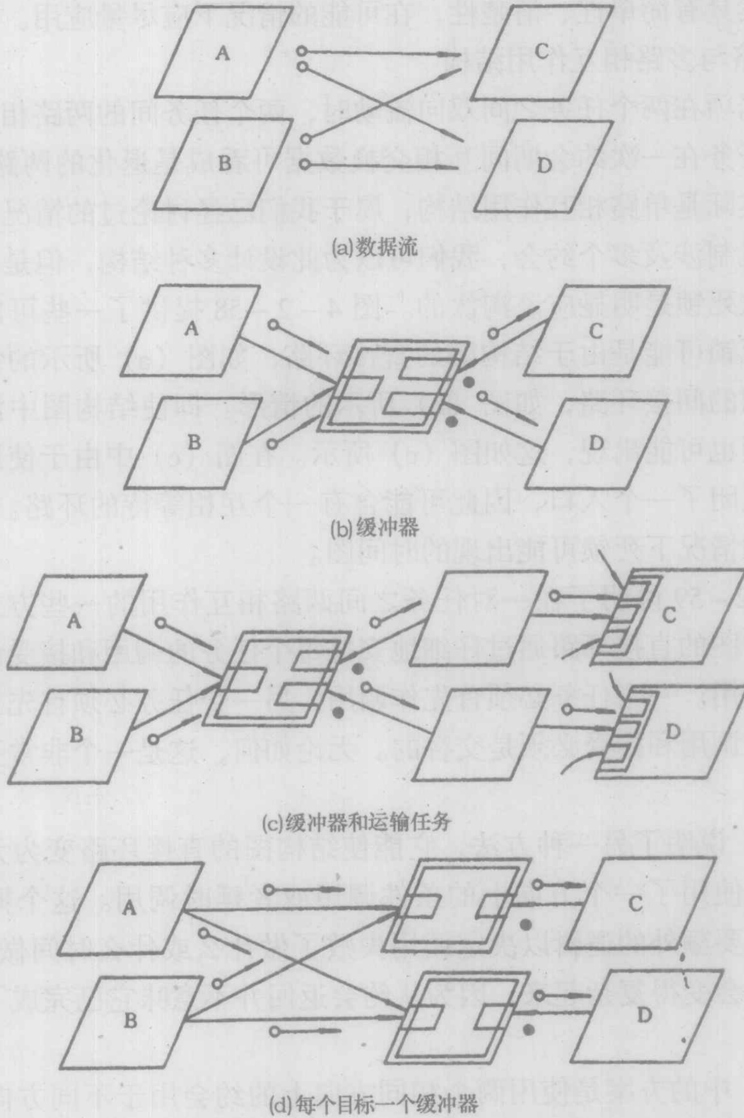
（3）多个任务对多个任务之间的相互作用

如果有多个发送者和多个目标者，就像图4-2-57中图（a）所示的情况，那么这样的问题如何解决呢？解决的方案是引入一个或多个缓冲区任务。图4-2-57中图（b）所示的缓冲区任务为所有的发送者提供了一个公共调用的入口，而为每个目标任务提供了分开的调用入口。对于所有的目标任务是已知的，且目标任务不但心因阻塞在入口队列中而丢失其他事情的情况下这种方法是很适当的。如果目标任务没有自己的入口，这个假设将是有效的。

为了提高灵活性，能在缓冲区任务和每个目标任务之间增加一个传输任务，如图4-2-57的图（c）所示。这允许目标任务等待其他入口调用而运输任务等待数据项抵达缓冲区任务。

图（d）所示的解比图（b）的多使用了任务，但都具有更好的模块化。与使用一个公共缓冲区任务不同，每个目标任务都有一个相联系的缓冲区。每个缓冲区为对应的目标任务提供了单一的管线，目标任务可以通过这个管线接受所有的对它的相互作用，因此，即使目标任务等待在入口队列中也不用担心丢失了其他事件。图（b）与图（d）的方案本质上是一样的，只是图

试读结束：需要全本请在线购买：　www.ertongbook.com



(a)数据流

(b)缓冲器

(c)缓冲器和运输任务

(d)每个目标一个缓冲器

图 4-2-57 使用缓冲器任务的多—对—多的相互作用

(b) 中缓冲区任务的逻辑更复杂，它必须区分哪些数据项是为任务 C 的，哪些是为任务 D 的。图 (b) 和图 (d) 的方案与图 (c) 的方案比较，灵活性差一些，在图 (b) 和 (d) 中目标任务必须按缓冲区提供数据项的次序处理它们。在图 (c) 的方案中某些任务可以通过目标任务的其他入口相互作用，并通过为目标任务的入口设置卫士条件控制这个顺序。无论如何，图

（d）的方案具有简单性、清楚性，在可能的情况下应尽量应用。

4. 两路与多路相互作用结构

当数据项在两个任务之间双向流动时，两个任务间的两路相互作用出现了。两个任务在一次约会期间互相交换数据可看成是退化的两路相互作用。这种情况实际是单路相互作用结构，属于我们已经讨论过的情况。

许多机制涉及多个约会，我们可以为此设计多种结构，但是其中一些由于可能导致死锁是明显应该淘汰的。图4-2-58提供了一些可能导致死锁的例子。死锁可能是由于结构图的直接环路，如图（a）所示的情形，或者由于结构图的间接环路，如图（b）所示的情形。即使结构图中没有明显的环路，死锁也可能出现，这如图（c）所示。在图（c）中由于使用卫士条件不适当地关闭了一个入口，因此可能含有一个互相等待的环路。图（d）表明了所有的情况下死锁可能出现的时间图。

图4-2-59说明了在一对任务之间两路相互作用的一些方式。图（a）表明了结构图的直接环路通过仔细地安排每个任务的调用和接受语句顺序也能安全地使用。一个任务必须首先作调用，另一个任务必须首先接受，此后每个任务的调用和接受必须是交替的。无论如何，这是一个非常受限制的方法。

图（b）说明了另一种方法，它能使结构图的直接环路变为无死锁，在这种方法中使用了一个方向上的条件调用或者延时调用。这个概念是简单的，但是需要额外的逻辑以决定调用失败了做什么或什么时间做再次调用。整个系统也会变得复杂起来，因为从约会返回并不意味它已完成了所要做的工作。

图（c）中的方案是使用两个相同方向上的约会用于不同方向上的数据流。拾取数据项能使用轮询或条件等待方式，无论如何（b）和（c）这两种方式都不能令人满意。图（d）虽然使用了两个方向上的约会，但是通过使用一个方向上的传输任务避免了死锁。在有传输任务方向上的发送者仅仅当它有数据项发送时才接受传输任务的调用。在另一方向上的发送者直接调用其目标任务。如果这两个任务的直接调用不过分地干扰调用者的其他活动，那么这个解是好的。因此，这个调用不应涉及条件等待。这个解相对于下一个要讨论的解的优点是使用了较少的任务，而缺点是对于一个对称的问题，却引入了一个非对称的结构。另外我们可能会注意到这个图与图4-2-58

中的图（c）在结构上非常相似，但差别在于设置卫士的地点不同，这里卫士是设置在传输任务调用的入口上，而在图4-2-58中图（c）的卫士是设置在主要任务调用的入口上。

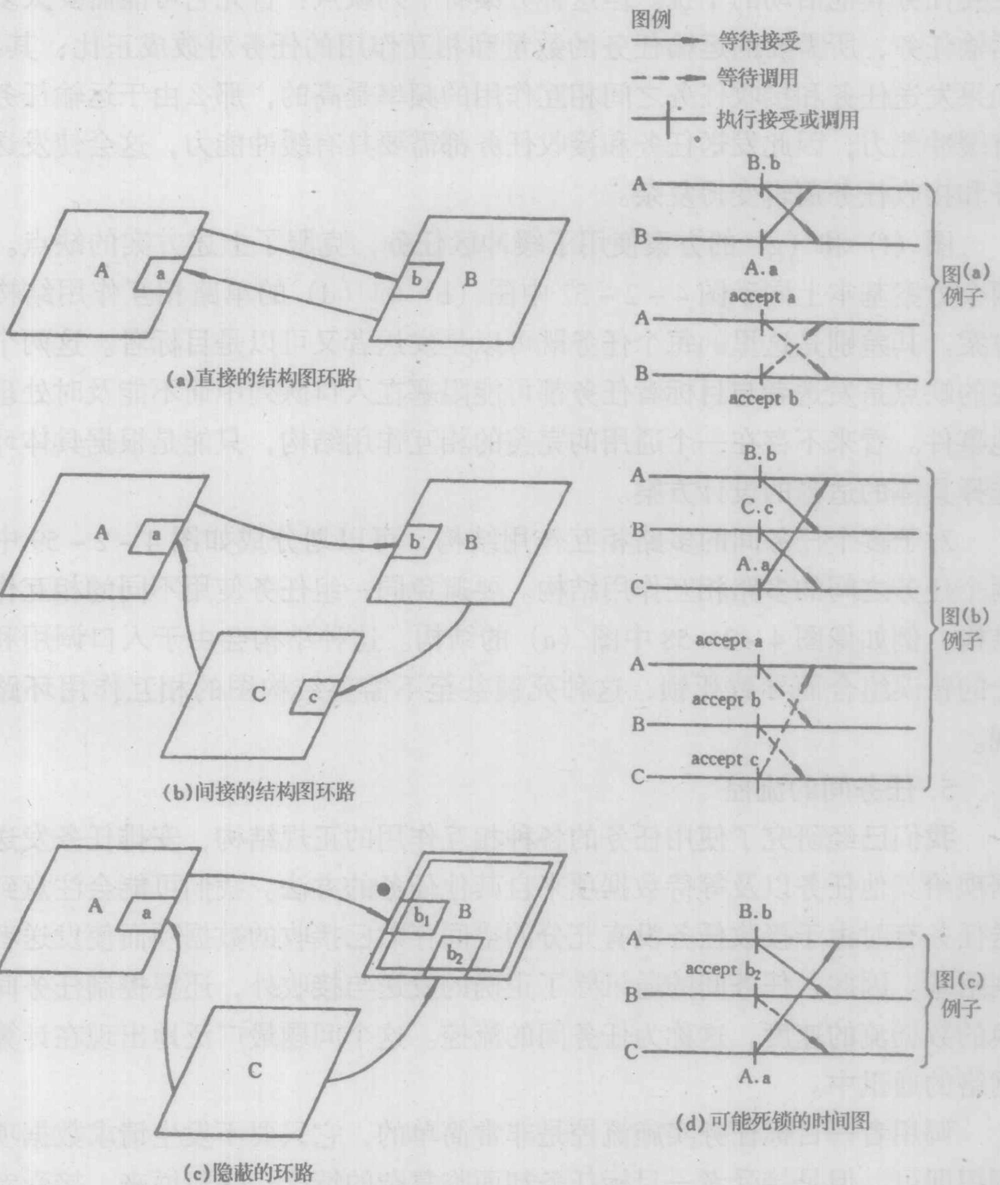


图4-2-58　约会死锁

图（e）中的方案使用了两个传输任务，每个数据流方向上一个。传输任务有条件地等待要发送的数据项。这个解是相当灵活的，它没有限制主要任务和其他任务的相互作用。像所有传输任务的解决方案一样，它最小化对主要任务其他活动的干扰。但这种方案有下列缺点：首先它可能需要太多的运输任务，所需要的运输任务的数量和相互作用的任务对数成正比；其次，如果发送任务和接收任务之间相互作用的频率是高的，那么由于运输任务没有缓冲能力，因此发送任务和接收任务都需要具有缓冲能力，这会使发送任务和接收任务逻辑变得复杂。

图（f）和（g）的方案使用了缓冲区任务，克服了上述方案的缺点。这两个方案基本上同于图 4－2－57 中图（b）和（d）的单路相互作用结构的方案。其差别是这里的每个任务既可以是发送者又可以是目标者。这两个方案的缺点是发送者与目标者任务都可能阻塞在入口队列中而不能及时处理其他事件。看来不存在一个通用的完美的相互作用结构，只能是根据具体环境选择具体的适当的设计方案。

对于多个任务间的多路相互作用结构，可以划分成如图 4－2－59 中的两个任务之间的多路相互作用结构。要避免同一组任务使用不同的相互作用结构，例如像图 4－2－58 中图（c）的结构。这种结构会由于入口调用和卫士的错误组合而导致死锁，这种死锁甚至不需要结构图的相互作用环路出现。

5. 任务间的流控

我们已经研究了使用任务的各种相互作用的正规结构，安排任务发送数据项给其他任务以及等待数据项来自其他任务的方法。我们可能会注意到发送任务有时由于接收任务没有充分的空间存贮已接收的数据项而使发送者受到阻塞。因此，任务间的通讯除了正确的发送与接收外，还要控制任务间交换的数据流的速度，这称为任务间的流控。这个问题最广泛地出现在计算机网络的通讯中。

调用者—目标任务实施流控是非常简单的，它只要不发生请求数据项的调用即可。但是接受者—目标任务却面临复杂的情况。一般说来，接受者—目标任务可以使用下列方式实施流控：

（1）使用卫士阻塞发送任务发送数据项；

（2）在约会中拒绝接收数据项，但是从约会中释放发送任务；

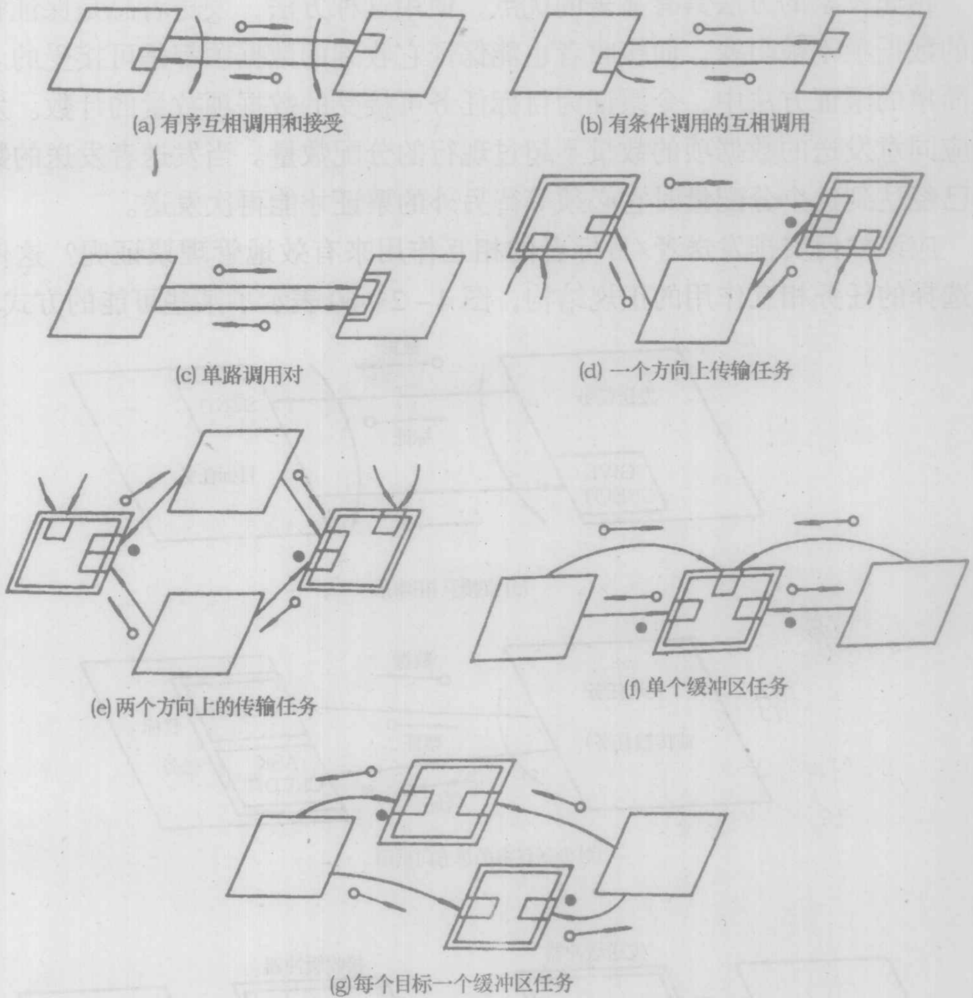


图 4－2－59 两路相互作用结构

(3) 在约会期间丢弃数据项；

(4) 调用可能的发送者告诉它们实施流控的时间；

(5) 在发送之前，给发送者票证，让发送者按规定的数量发送。

前三种方法要求发送者首先发送，然后要么发送者进行等待，或者该项发送失败。这些流控方式要么可能不必要地消耗了资源，要么可能丢失数据。第四种方式也不是令人满意的。因为通知可能没有及时抵达，并且因为并行的相互调用可能导致死锁。

预先发票的方法具有显著的优点，使用这种方法，发送者总是保证它发送的数据项不被阻塞，而接收者也能保证它收到的数据项都是可接受的。在最简单的票证方法中，令票证为目标任务可接受的数据项数量的计数。发送者应同意发送的数据项的数量不超过现行的分配数量。当发送者发送的数据项已经达到这个分配量时它必须等待另外的票证才能再次发送。

应该如何安排发送者/目标者的相互作用来有效地管理票证呢？这依赖于选择的任务相互作用的正规结构，图 4－2－60 表示了某些可能的方式。

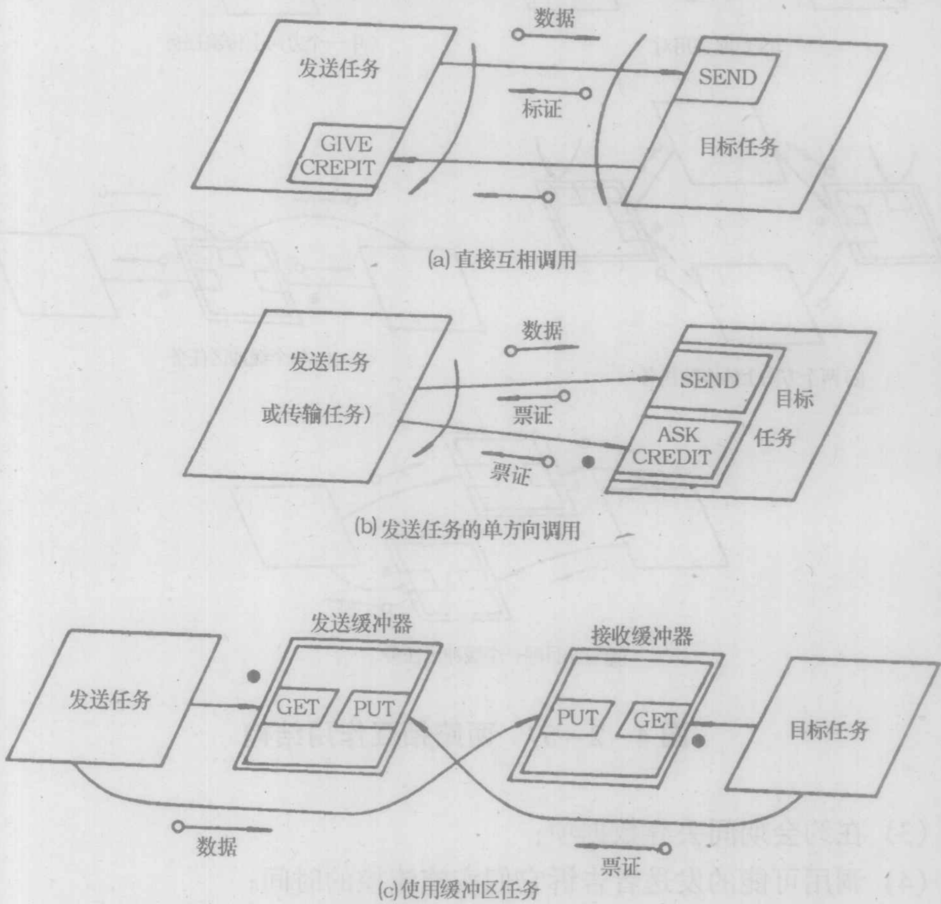


图 4－2－60　使用票证流控的任务相互作用结构

如果相互作用是直接的或者是通过传输任务，那么该图中图（a）和（b）的方法是可行的。在图（a）和（b）中假定当票证是可用的时候，它是作为 SEND 入口的输出参数返回给发送者。系统在开始第一次通讯时，发送

试读结束：需要全本请在线购买： www.ertongbook.com

者任务并不立即发送数据，而仅仅是为了取得票证。在后来的调用中是既发送数据，又取回可用的票证。当通过 SEND 入口没有票证可用时，发送任务等待目标任务调用它以发送票证或者主动通过专用入口请求票证。

图（a）尽管通过有次序的调用与接受，保证了这个结构是无死锁的，无论如何这在发送者和目标者之间引入了过分紧的耦合，因而是非模块化的。

如图（b）所示，目标任务能够提供一个分开的有卫士的入口 ASK－CREDIT，在发送者任务已经发送了一个数据项之后，如果不再有票证可用时，发送者可以等待在此处。如果发送者不能直接等待，可引入一个传输任务代替发送任务在 ASR－CREDIT 处等待。

如果选择缓冲区任务作为相互作用的机构，那么图（c）所示的方法是适当的。PUT 入口可以为许多发送者使用，而每个 GET 入口仅由拥有该缓冲区任务的目标任务使用。每一个发送者也可以是一个目标者，在这种情况下可以实现双向通讯。

6. 主动包—流水线缓冲区

在发送任务和目标任务之间引入一个缓冲区任务虽然能够对发送者和目标者的发送与接收的速度变化进行缓冲，但是由于任务一次只能和一个任务约会，因此，当缓冲区任务和其中之一进行约会时，则阻塞了另一个任务的运行。现在假定我们要设计一个缓冲区机构使发送者和目标者之间取得更松的耦合，我们可以采用如图 4－2－61 所示的使用包封装的流水线缓冲区。这个流水线缓冲区由两个缓冲区任务和一个传输任务构成，为了给其用户提供一个基本的程序单元，我们用包将其封装起来。当这个流水线缓冲区即不满也不空时，它允许数据的发送者和目标者同时存取它的两个过程接口。这个特征使发送者与目标者之间的耦合变得更松，因而是更理想的。它的 Ada 程序框架如下：

```
package PipeLineBuffer is
    …
    procedure PUT（…）
    procedure GET（…）;
end;
package body PipeLineBuffer is
```

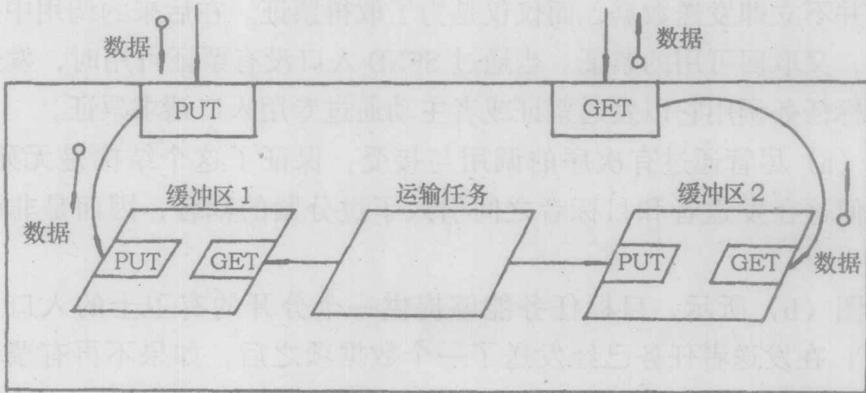


图 4-2-61　主动包：流水线缓冲区

```
        task type Buffers is
            entry PUT (…);
            entry GET (…);
    end;
    Bufferl, Buffer2: Buffers;
    task body Buffers is
    begin
        loop
            accept PUT (…) do
            …
            end;
            accept GET (…) do
            …
            end
        endloop;
    end Buffers;
        task transfer;
        task body fransfer is
            …
        begin
```

```
      loop
          Buffer1.GET (…);
          Buffer2.PUT (…);
      endloop;
    end transfer;
    procedure PUT (…) is
      begin
      Buffer1.PUT (…)
    end;
    procedure GET (…) is
    begin
      Buffer2.GET (…);
    end;
end PipeLineBuffer;
```

在上述程序框架中我们省去了传送数据项的类型细节。包的客户调用包所提供的接口 PUT 与 GET，将数据项送入流水线缓冲区或从流水线缓冲区把数据取走。具体操作顺序是发送者调用包的 PUT 过程，由 PUT 过程调用缓冲区 1 的 PUT 入口，将数据置入缓冲区 1 中，传输任务取走该数据放入缓冲区 2 中，目标任务调用包的过程 GET，由 GET 调用缓冲区 2 的 GET 入口取得数据。尽管数据是顺序地从缓冲区 1 到缓冲区 2，但是所有任务的操作却是并行的。

三、对象设计

面向对象的系统设计很像建筑行业中建筑物的平面图设计。平面图规定了每个房间的用途和各房间相连接的机制以及建筑物的外部环境。现在是提供建造每个房间所需细节内容的时间了。

在 OOD 的意义上，对象设计是集中在“房间”上，我们必须开发组成每个类的属性和操作的详细设计，以及它发出的消息的彻底的规范。这些消息将类与其协助者联接起来。

（一）对象描述

一个对象的设计描述能够取两种形式：

（1）协议描述，通过定义对象能够接收的每个消息和当对象接收到消息时它执行的有关的操作。

（2）实现描述，它表明一个对象收到的消息所涉及操作的实现细节。实现细节包括了对象私有部分的信息，即关于描述对象属性的数据结构的内部细节和描述操作的过程细节。

协议描述不过是一组消息和关于每个消息的对应注释。例如，对于（早先描述的）运动传感器对象的协议部分描述可以是：

MESSAGE（运动传感器）→读：RETURNS 传感器 ID，传感器状态；

MESSAGE（运动传感器）→置：SENDS 传感器 ID，传感器状态；

前者描述了读运动传感器所需要的消息；后者描述了设定运动传感器所需要的消息。对于一个大的系统，可能需要许多消息，为了便于管理，可将消息划分成一些不同的种类。例如，对于房屋安全系统中系统对象的消息种类可分为系统配置消息、监视消息、事件消息等等。

对象的实现描述为实现提供了所需要的内部的（隐藏的）的细节，即对象的设计者必须提供实现描述和生成对象的内部细节。无论如何使用这个对象的其他对象的设计者和实现者则仅仅需要知道协议描述而不必知道实现描述。

实现描述是由下列信息组成：①对象与类的名字规范；②指明数据项和类型的私有数据结构规范；③每个操作的过程描述。实现描述必须含有对协议描述中所有消息的适当处理的充分信息。

我们能够用术语服务的“用户”与服务的“提供者”描述包含在协议描述和实现描述中信息的差别。使用对象服务的用户必须熟悉引用服务的协议，即规定需要什么服务；而服务的提供者（对象自身）必须关心服务将如何提供给用户，即关心实现细节。这是我们先前讨论的封装概念的目标。

（二）定义算法和数据结构

在分析模型和系统设计中的各种表示为属性和操作的设计提供了规范。

设计算法和数据结构的方法是稍微不同于传统的设计方法。

为实现每个操作的规范设计算法。在许多情况下，算法是一个简单的计算或过程序列并能实现为一个自包含的模块。但是如果一个操作的规范是复杂的，可能必须对该操作进行模块化，方法是使用传统的过程设计技术。

数据结构的设计应该先于算法的设计，至少是和算法设计同时进行。因为操作总是操纵类的属性，因此最好地表示属性的数据结构的设计将会对相应操作的算法设计有强烈的影响。

尽管有许多不同类型的操作存在，但是它们一般可划分为3种更广的范畴：②以某种方式操纵数据的操作（如填加、删除、格式化和选择）；②执行计算的操作；③监控对象的受控事件出现的操作。例如，房屋安全系统的处理叙述含有语句“为传感器指定序号和类型”和“程序设计主口令以开启和关闭系统”。这两个短语表明了一些事情：

·和传感器对象有关的赋值操作；

·应用于系统对象的程序设计操作；

·应用于系统对象的启动与关闭操作，系统状态可以最终定义为（使用数据字典符号）：

系统状态 = ［开启的|关闭的］

操作程序设计是在 OOA 期间确定的，它将被精炼成一些更特殊的操作以配置系统。例如，当设计者和分析员讨论之后，他可能细化原来的有关程序设计的处理描述如下：

一旦房屋安全系统安装之后，程序设计允许该系统的用户配置该系统。用户能：①设置电话号码；②为报警定义延迟时间；③建立含有每个传感器ID，类型和位置的传感器表；④装入主口令。

设计者已经精炼了单一的操作程序设计，并用操作：设置、定义、建立和装入代替它。这些新操作都变成对象系统的一部分，有实现对象属性的内部数据结构的知识，并可通过下列形式的消息进行引用：

对象名·消息名（实参数表）

其中对象名为接收消息的对象；消息名为打算引用的方法名，实参数表为操作输入数据或取得操作输出数据的常数或变量。例如下列消息：

system·Install（“1 2 3 4 5 6 7 8”）

将引用对象系统的设置操作，设置紧急电话号码：12345678。

在上面的例子中，我们把一个复杂的操作程序设计分解成一些更特殊，但同时是更简单的一些操作。

（三）程序部件和接口

软件设计质量的一个重要方面是模块化——即一些程序部件（模块）相结合以形成程序。面向对象的方法定义对象作为程序部件，对象自身又联接到其他部件（如私有数据、操作）。在设计期间我们必须标识出对象之间的接口和对象的总体结构（体系结构）。

程序部件是一个设计抽象，它的描述最好使用其实现的语言。如果设计用 Ada 实现，那么我们可以直接用 Ada 语言描述。Ada 不仅是一个程序设计语言，而且是一个系统设计语言，可以方便地描述系统的设计。如果系统不准备用 Ada 语言实现，可以采用下面所示的 Ada 样的程序描述语言（PDL，Program Description Language）建模。无论如何，所使用的程序语言应该能够容易地实现下述按 Ada 方式建模的程序部件。

```
PACKAGE program - component - name IS
    TYPE Specification of data objects…
    PROC Specificotion of related operations…
PRIVATE
    data structure details for objects
END
PACKAGE BODY program - components - name IS
    PROC Operation.1 (interface description) IS
    …
    END
    …
    PROC Operation.n (interface description) IS
    …
    END
END program - component - name
```

使用上边所示的 Ada 样的 PDL，通过规定数据对象和操作二者描述程序

试读结束：需要全本请在线购买： www.ertongbook.com