源编号低的资源。所以，若一个进程起初请求9号和10号资源，而随后释放两者，那么它实际上相当于从头开始，所以没有必要阻止它现在请求1号资源。

尽管对资源编号的方法消除了死锁的问题，但几乎找不出一种使每个人都满意的编号次序。当资源包括进程表项、假脱机磁盘空间、加锁的数据库记录及其他抽象资源时，潜在的资源及各种不同用途的数目会变得很大，以至于使编号方法根本无法使用。

条　件	处理方式
互斥	一切都使用假脱机技术
占有和等待	在开始就请求全部资源
不可抢占	抢占资源
环路等待	对资源按序编号

图6-14　死锁预防方法汇总

死锁预防的各种方法如图6-14所示。

6.7　其他问题

在本节中，我们会讨论一些和死锁相关的问题，包括两阶段加锁、通信死锁、活锁和饥饿。

6.7.1　两阶段加锁

虽然在一般情况下避免死锁和预防死锁并不是很有希望，但是在一些特殊的应用方面，有很多卓越的专用算法。例如，在很多数据库系统中，一个经常发生的操作是请求锁住一些记录，然后更新所有锁住的记录。当同时有多个进程运行时，就有出现死锁的危险。

常用的方法是两阶段加锁（two-phase locking）。在第一阶段，进程试图对所有所需的记录进行加锁，一次锁一个记录。如果第一阶段加锁成功，就开始第二阶段，完成更新然后释放锁。在第一阶段并没有做实际的工作。

如果在第一阶段某个进程需要的记录已经被加锁，那么该进程释放它所有加锁的记录，然后重新开始第一阶段。从某种意义上说，这种方法类似于提前或者至少是未实施一些不可逆的操作之前请求所有资源。在两阶段加锁的一些版本中，如果在第一阶段遇到了已加锁的记录，并不会释放锁然后重新开始，这就可能产生死锁。

不过，在一般意义下，这种策略并不通用。例如，在实时系统和进程控制系统中，由于一个进程缺少一个可用资源就半途中断它，并重新开始该进程，这是不可接受的。如果一个进程已经在网络上读写消息、更新文件或从事任何不能安全地重复做的事，那么重新运行进程也是不可接受的。只有当程序员仔细地安排了程序，使得在第一阶段程序可以在任意一点停下来，并重新开始而不会产生错误，这时这个算法才可行。但很多应用并不能按这种方式来设计。

6.7.2　通信死锁

到目前为止，我们所有的工作都着眼于资源死锁。一个进程需要使用另外一个进程拥有的资源，因此必须等待直至该进程停止使用这些资源。有时资源是硬件或者软件，比如说CD-ROM驱动器或者数据库记录，但是有时它们更加抽象。在图6-2中，可以看到当资源互斥时发生的资源死锁。这比CD-ROM驱动器更抽象一点，但是在这个例子中，每个进程都成功调用一个资源（互斥锁之一）而且死锁的进程尝试去调用另外的资源（另一个互斥锁）。这种情况是典型的资源死锁。

然而，正如我们在本章开始提到的，资源死锁是最普遍的一种类型，但不是惟一的一种。另一种死锁发生在通信系统中（比如说网络），即两个或两个以上进程利用发送信息来通信时。一种普遍的情形是进程A向进程B发送请求信息，然后阻塞直至B回复。假设请求信息丢失，A将阻塞以等待回复，而B会阻塞等待一个向其发送命令的请求，因此发生死锁。

仅仅如此并非经典的资源死锁。A没有占有B所需的资源，反之亦然。事实上，并没有完全可见的资源。但是，根据标准的定义，在一系列进程中，每个进程因为等待另外一个进程引发的事件而产生阻塞，这就是一种死锁。相比于更加常见的资源死锁，我们把上面这种情况叫做通信死锁（communication deadlock）。

通信死锁不能通过对资源排序（因为没有）或者通过仔细地安排调度来避免（因为任何时刻的请求都是不被允许延迟的）。幸运的是，另外一种技术通常可以用来中断通信死锁：超时。在大多数网络通信系统中，只要一个信息被发送至一个特定的地方，并等待其返回一个预期的回复，发送者就同时启动