1) Write：把一个记录项放入JavaSpace。

2) Read：将一个与模板匹配的记录项复制出JavaSpace。

3) Take：复制并移走一个与模板匹配的记录项。

4) Notify：当一个匹配的记录项写入时通知调用者。

write方法提供记录项并确定其租约时间，即何时应该丢弃该记录项。相反，Linda的元组则一直停留着直到被移出为止。在JavaSpace中可以保存有同一个记录项的多个副本，所以它不是一个数学意义上的集合（如同Linda那样）。

read和take方法为要寻找的记录项提供了一个模板。在该模板的每个域中有一个必须匹配的特定值，或者可以包含一个“不在乎”的通配符，该通配符可以匹配所有合适的类型的值。如果发现一个匹配，则返回该记录项，而在take的情形下，该记录项还被移出了JavaSpace空间。这些JavaSpace方法中的每一个都有两个变种，在没有匹配到记录项时，它们之间有所差别。其中一个变种即刻返回一个失败的标识。而另一个则一直等到时间段（作为一个参数给定）到期为止。

notify方法用一个特殊模板注册兴趣。如果以后进来了一个相匹配的记录项，就调用调用者的notify方法。

与Linda中的元组空间不同，JavaSpace支持原子事务处理。通过使用原子事务处理，可以把多个方法聚集在一起。它们要么全部都执行，要么全部都不执行。在该事务处理期间，在该事务处理之外对JavaSpace的修改是不可见的。只有在该事务处理结束之后，它们才对其他的调用者可见。

可以在通信进程之间的同步中运用JavaSpace。例如，在生产者-消费者的情形下，生产者在产品生产出来之后可以把产品放进JavaSpace中。消费者使用take取走这些产品，如果产品没有了就阻塞。JavaSpace保证每个方法的执行都是原子性的，所以不会出现当一个进程试图读出一个记录项时，该记录项仅仅完成了一半进入的危险。

8.4.7 网格

如果没有谈及最新的发展，即在未来有可能变得非常重要的网格，那么，对于分布式系统的论述将是不完整的。所谓网格（grid），是一个大的、地理上分散的、通常是由私有网络或因特网连接起来的异构机器的集合，向用户提供一系列服务。有时候网格也被比作虚拟超级计算机，但其实还不只是这样。它是很多独立计算机的集合，一般位于多个管理域中，所有的这些管理域都会运行中间件的一个公共的中间件层以便用户和程序可以通过方便和一致的方式访问所有资源。

构建网格的初始动机是为了CPU的时钟周期共享。当时的想法是：当一个机构不需要它的全部的计算能力时（例如在夜间），另一个机构（可能相隔好几个时区）就可以利用这些时钟周期，并且12小时之后也对外提供这样的帮助。现在，网格研究人员也在关注其他资源的共享，尤其是专门硬件和数据库。

典型地，网格的工作原理是：在每个参与的机器中运行一组管理机器并且把它加入到网格中的程序。这个程序通常需要处理认证及远程用户登录、资源发布及发现、作业调度及分配等。当某个用户有工作需要计算机来做时，网格软件决定哪里有空闲的硬件、软件和数据资源来完成这项工作，然后将作业搬运过去，安排执行并收集计算结果返回给用户。

在网格世界中，一个流行的中间件叫Globus Toolkit，它在很多平台上都是可用的并且支持很多（即将出现的）网格标准（Foster，2005）。Globus通过灵活和安全的方式提供一个供用户共享计算机、文件以及其他资源的平台，同时又不会牺牲本地的自治性。网格正在成为很多分布式应用的构建基础。

8.5 有关多处理机系统的研究

在本章中，我们考察了四类多处理器系统：多处理器、多计算机、虚拟机和分布式系统。下面简要地介绍在这些领域中的有关研究工作。

在多处理器领域中的多数研究与硬件有关，特别是与如何构建共享存储器和保持其一致性（如Higham等人，2007）有关。然而，还有一些关于多处理器的其他研究，特别是片上多处理器，包括编程模型和随之带来的操作系统问题（Fedorova等人，2005；Tan等人，2007）、通信机制（Brisolara等人，2007）、软件的能源管理（Park等人，2007）、安全（Yang和Peng，2006）还有未来的挑战（Wolf，