走出这个困局的一个途径是，将高速缓存文件上的改动立即传送回服务器。尽管概念上很简单，但这个方法却是低效率的。另一个解决方案是放宽文件共享的语义。一般的语义要求一个读操作要看到其之前的所有写操作的效果，我们可以定义一条新规则来取代它："在一个打开文件上所进行的修改，最初仅对进行这些修改的进程是可见的。只有在该文件关闭之后，这些修改才对其他进程可见。"采用这样一个规则不会改变在图8-38b中发生的事件，但是这条规则确实重新定义了所谓正确的具体操作行为（B得到了文件的原始值）。当客户机1关闭文件时，它将一个副本回送给服务器，因此，正如所期望的，后续的read操作得到了新的值。实际上，这个规则就是图8-36中的上传/下载模式。这种语义已经得到广泛的实现，即所谓的会话语义（session semantic）。

使用会话语义产生了新的问题，即如果两个或更多的客户机同时缓存并修改同一个文件，应该怎么办？一个解决方案是，当每个文件依次关闭时，其值会被送回给服务器，所以最后的结果取决于哪个文件最后关闭。一个不太令人满意的、但是较容易实现的替代方案是，最后的结果是在各种候选中选择一个，但并不指定是哪一个。

对会话语义的另一种处理方式是，使用上传/下载模式，但是自动对已经下载的文件加锁。其他试图下载该文件的客户机将被挂起直到第一个客户机返回。如果对某个文件的操作要求非常多，服务器可以向持有该文件的客户机发送消息，询问是否可以加快速度，不过这样做可能没有作用。总而言之，正确地实现共享文件的语义是一件棘手的事情，并不存在一个优雅和有效的解决方案。

8.4.5 基于对象的中间件

现在让我们考察第三种范型。这里不再说一切都是文档或者一切都是文件，取而代之，我们会说一切都是对象。对象是变量的集合，这些变量与一套称为方法的访问过程绑定在一起。进程不允许直接访问这些变量。相反，要求它们调用方法。

有一些程序设计语言，如C++和Java，是面向对象的，但这些对象是语言级的对象，而不是运行时刻的对象。一个知名的基于运行时对象的系统是CORBA（公共对象请求代理体系结构，Common Object Request Broker Architecture）（Vinoski，1997）。CORBA是一个客户机-服务器系统，其中在客户机上的客户进程可以调用位于（可能是远程）服务器上的对象操作。CORBA是为运行不同硬件平台和操作系统的异构系统而设计的，并且用各种语言编写。为了使在一个平台上的客户有可能使用在不同平台上的服务器，将ORB（对象请求代理，Object Request Broker）插入到客户机和服务器之间，从而使它们相互匹配。ORB在CORBA中扮演着重要的角色，以至于连该系统也采用了这个名称。

每个CORBA对象是由叫做IDL（接口定义语言，Interface Definition Language）的语言中的接口定义所定义的，说明该对象提供什么方法，以及每个方法期望使用什么类型的参数。可以把IDL的规约（specification）编译进客户端桩过程中，并且存储在一个库里。如果一个客户机进程预先知道它需要访问某个对象，这个进程则与该对象的客户端桩代码链接。也可以把IDL规约编译进服务器一方的一个框架（skeleton）过程中。如果不能提前知道进程需要使用哪一个CORBA对象，进行动态调用也是可能的，但是有关动态调用如何工作的原理则不在本书的讲述范围内。

当创建一个CORBA对象时，一个对它的引用也创建出来并返回给创建它的进程。该引用涉及进程如何标识该对象以便随后对其方法进行调用。该引用还可以传递给其他的进程或存储在一个对象目录中。

要调用一个对象中的方法，客户机进程必须首先获得对该对象的引用。引用可以直接来源于创建进程，或更有可能是，通过名字寻找或通过功能在某类目录中寻找。一旦有了该对象的引用，客户机进程将把方法调用的参数编排进一个便利的结构中，然后与客户机ORB联系。接着，客户机ORB向服务器ORB发送一条消息，后者真正调用对象中的方法。整个机制类似于RPC。

ORB的功能是将客户机和服务器代码中的所有低层次的分布和通信细节都隐藏起来。特别地，客户机的ORB隐藏了服务器的位置、服务器是二进制代码还是脚本、服务器在什么硬件和操作系统上运行、有关对象当前是否是活动的以及两个ORB是如何通信的（例如，TCP/IP、RPC、共享内存等）。

在第一版CORBA中，没有规定客户机ORB和服务器ORB之间的协议。结果导致每一个ORB的销售商都使用不同的协议，其中的任何两个协议之间都不能彼此通信。在2.0版中，规定了协议。对于用在Internet上的通信，协议称为IIOP（Internet InterOrb Protocol）。