另一方面，我们默认在自动挂载时所有可选的文件系统都是完全相同的。由于NFS不提供对文件或目录复制的支持，用户需要自己确保所有这些文件系统都是相同的。因此，自动挂载多数情况下被用于包含系统代码的只读文件系统和其他很少改动的文件。

第二个NFS协议是为访问目录和文件设计的。客户端可以通过向服务器发送消息来操作目录和读写文件。客户端也可以访问文件属性，如文件模式、大小、上次修改时间。NFS支持大多数的Linux系统调用，但是也许很让人惊讶的是，open和close不被支持。

对open和close的省略并不是意外事件，而纯粹是有意为之。没有必要在读一个文件之前先打开它，也没有必要在读完后关闭它。读文件时，客户端向服务器发送一个包含文件名的lookup消息，请求查询该文件并返回一个标识该文件的文件句柄（即包含文件系统标识符i节点号以及其他数据）。与open调用不同，lookup操作不向系统内部表中复制任何信息。read调用包含要读取的文件的文件句柄，起始偏移量和需要的字节数。每个这样的消息都是自包含的。这个方案的优势是在两次read调用之间，服务器不需要记住任何关于已打开的连接的信息。因此，如果一个服务器在崩溃之后恢复，所有关于已打开文件的信息都不会丢失，因为这些信息原本就不存在。像这样不维护打开文件的状态信息的服务器称作是无状态的。

不幸的是，NFS方法使得难以实现精确的Linux文件语义。例如，在Linux中一个文件可以被打开并锁定以防止其他进程对其访问。当文件关闭时，锁被释放。在一个像NFS这样的无状态服务器中，锁不能与已打开的文件相关联，这是因为服务器不知道哪些文件是打开的。因此，NFS需要一个独立的，附加的机制来处理加锁。

NFS使用标准UNIX保护机制，为文件属主、组和其他用户使用读、写、执行位（rwx bits）（在第1章中提到过，将在下面详细讨论）。最初，每个请求消息仅仅包含调用者的用户ID和组ID，NFS服务器用它们来验证访问。实际上，它信任客户端，认为客户端不会进行欺骗。若干年来的经验充分表明了这样一个假设。现在，可以使用公钥密码系统建立一个安全密钥，在每次请求和应答中使用它验证客户端和服务器。启用这个选项后，恶意的客户端就不能伪装成另一个客户端了，因为它不知道其他客户端的安全密钥。

3. NFS实现

尽管客户端和服务器代码实现独立于NFS协议，但大多数Linux系统使用一个类似图10-36所示的三层实现。顶层是系统调用层，这一层处理如open、read和close之类的调用。在解析调用和参数检查结束后，调用第二层——虚拟文件系统（VFS）层。

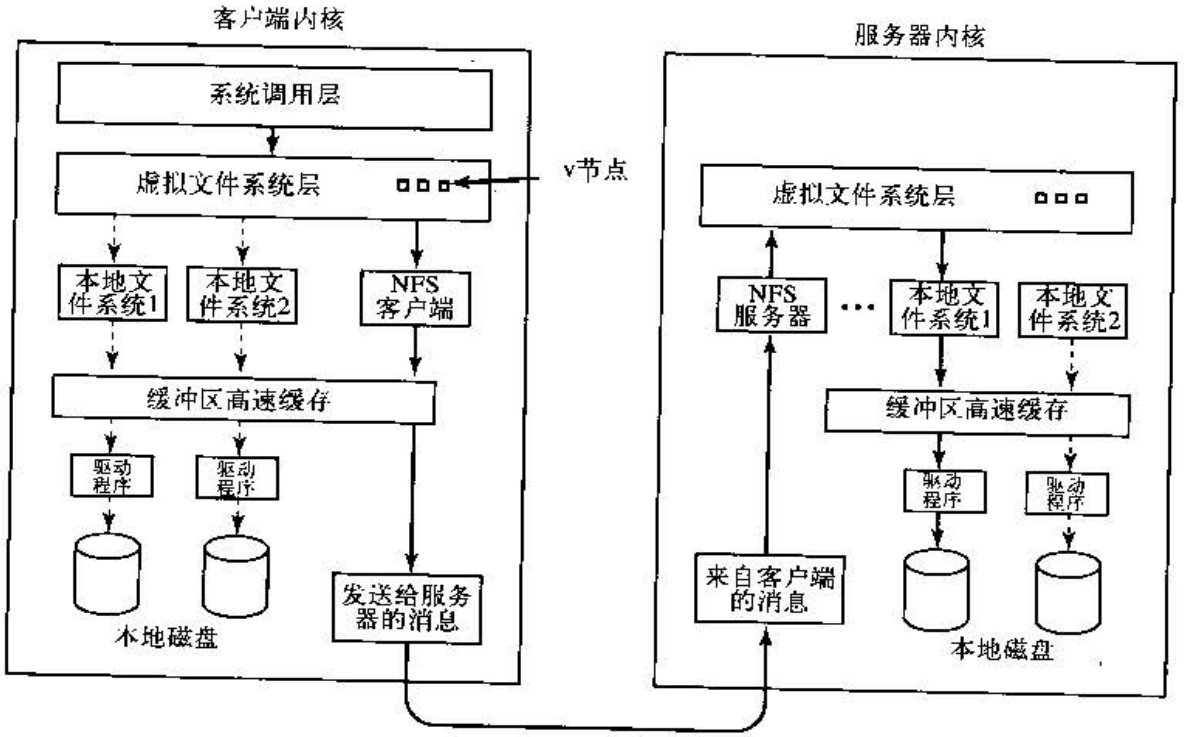

图10-36 NFS层次结构