的用于存储程序计数器的正在运行的周期样本的数据结构，用以确定程序线程的时间是花在哪些地方了。

段用来表示内存对象，这些内存对象可以被应用程序向内存管理器请求，将应用程序的地址空间映射到这个区域中来。它们记录表示磁盘上的内存对象的页的文件（或页面文件）的段。键表示的是象管理名字空间的注册表名字空间的加载点。通常只有一个名为\REGISTRY关键对象，负责链接到注册表键值和NT 名字空间的值。

对象目录和符号链接完全是本地对象管理器的 NT 名字空间的一部分。它们是类似于和它们对应的文件系统部分：目录允许要收集一些相关的对象。 符号链接允许对象名字空间来引用一个对象名字空间的不同部分中的对象的一部分的名称。

每个已知的操作系统的设备有一个或多个设备对象包含有关它的信息，并且由系统引用该设备。最后，每个已加载设备驱动程序在对象空间中有一个驱动程序对象。驱动程序对象被所有那些表示被这些驱动控制的设备的实例共享。

其他没有介绍的对象有更多特别的目的，如同内核事务的交互或 Win32线程池的工作线程工厂交互。

11.3.4 子系统、 DLL 和用户态服务

回到图11-6，我们可以看到 Windows Vista 操作系统是由内核态中的组件和用户态的组件组成的。现在我们已经介绍完了我们的内核态组件，因此，我们接下来看看用户态组件。其中对于 Windows 有三种组件尤为重要：环境子系统、DLL 和服务进程。

我们已介绍 Windows 子系统模型，所以这里不作更多详细介绍，而主要是关注原始设计的NT，子系统被视为一种利用内核态运行相同底层软件来支持多个操作系统个性化的方法。也许这是试图避免操作系统竞争相同的平台，例如在DEC的VAX上的VMS和Berkeley UNIX。或者也许在微软没有人知道OS/2是否会成为一个成功的编程接口，他们加上了他们的投注。结果，OS/2 成为无关的后来者，而Win32 API 设计为与 Windows 95结合并成为主导。

Windows 用户态设计的第二个重要方面是在动态链接库（DLL），即代码是在程序运行的时候完成的链接，而非编译时。共享的库不是一个新的概念，最现代化的操作系统使用它们。在 Windows 中几乎所有库都是DLL，从每一个进程都装载的系统库ntdll.dll到旨在允许应用程序开发人员进行代码通用的功用函数的高层程序库。

DLL通过允许在进程之间共享通用代码来提高系统效率，保持常用代码在内存中，处理减少从程序磁盘到内存中的加载时间。并允许操作系统的库代码进行更新时无需重新编译或重新链接所有使用它的应用程序，从而提高系统的使用能力。

此外，共享的库介绍版本控制的问题，并增加系统的复杂性，因为为帮助某些特定的应用而引入的更改可能会给其他的一些特定的应用带来可能的错误，或者因为实现的改变而破坏了一些其他的应用——这是一个在 Windows世界称为DLL黑洞的问题。

DLL 的实现在概念上是简单的。并非直接调用相同的可执行映像中的子例程的代码，一定程度的间接性引用被编译器引入：IAT（导入地址表）。当可执行文件被加载时，它查找也必须加载的DLL的列表（这将是一个图结构，因为这些DLL本身会指定它们所需要的其他的DLL列表）。所需的DLL被加载并且填写好它们的IAT。

现实是更复杂的。另一个问题是代表DLL之间的关系图可以包含环，或具有不确定性行为，因此计算要加载的DLL列表可以导致不能运行的结果。此外，在 Windows中DLL代码库有机会来运行代码，只要它们加载到了进程中或者创建一个新线程。通常，这是使它们可以执行初始化，或为每个线程分配存储空间，但许多DLL在这些附加例程中执行大量的计算。 如果任何函数调用的一个附加例程需要检查加载的 DLL列表，死锁可能会发生在这个过程。

DLL用于不仅仅共享常见的代码。它们还可以启用一种宿主的扩展应用程序模型。Internet Explorer 可以下载并链接到 DLL 调用 ActiveX 控件。 另一端互联网的 Web 服务器也加载动态代码，以为它们所显示的网页产生更好的Web体验。像Microsoft Office的应用程序允许链接并运行DLL，使得Office可以类似一个平台来构建新的应用程序。COM（组件对象模型）编程模式允许程序动态地查找和加载编写来提供特定发布接口的代码，这就导致几乎所有使用 COM的应用程序都以in-process的方式来托管 DLL。