作业是一种可命名、可保护、可共享的内核对象，借此可以用“组”的方式控制一个或多个进程。作业对象的基本功能在于，可以将一组进程作为一个单元加以管理和操作，一个进程可以是任意数量个作业的成员(Win 8开始)，不过通常只会是一个作业的成员。进程与作业对象的关联无法打断，由一个进程及其后代创建的所有进程会关联至相同作业对象(除非使用CREATE\_BREAKAWAY\_FROM\_JOB (标志创建子进程并且作业本身对此无限制)。作业对象还会记录与作业相关联的所有进程(包括所有曾经与该作业关联，但后来被终止的进程)的基本审计信息。

作业还可以关联至I/O完成端口对象，而此时可能还有其他线程正在等待此对象，为此需要使用 Windows的GetQueuedCompletionStatus 函数，或使用线程池API(原生函数TpAllocJobNotification)。借此可以让对此有兴趣的其他方 (通常为作业的创建者)监视对各种限制的违反情况，以及可能影响到作业安全性的事件，例如新进程的创建或进程的异常退出。

作业在很多系统机制中扮演了重要的角色，如下。

1. 管理现代化应用(UWP 进程)，实际上，每个现代化应用都是在一个作业中运行的。
2. 借助一种名为Server Silo的机制，可实现 Windows 对容器的支持
3. 是Desktop Activity Moderator (DAM)为Wn32应用程序和服务管理限制(throttling).计时器虚拟化、计时器冻结以及其他闲置行为的主要方式。
4. 可用于为动态公平共享调度(Dynamic Fair-Share Scheduling，DFSS)定义并管理调度组。
5. 可用于实现自定义内存分区
6. 用于实现诸如 Run As(辅助登录)应用程序装箱(boxing)和程序兼容性助手等功能。
7. 为诸如 Google Chrome 和 Microsoft Office 文档转换器等应用程序提供安全沙盒的部分功能，并可通过 Windows Management Instrumentation (WM请求针对拒绝服务(DoS)攻击提供缓解措施。

作业的限制

我们可以为作业指定如下这些与 CPU、内存以及IO 有关的限制。

1. **活动进程最大数量。**限制了作业中可同时包含的进程数量。如果达到该限制，将无法继续创建分配给该作业的进程。
2. **作业范围内的用户模式 CPU 时间限制。**限制了作业中进程(包括曾运行但已退出的进程)可使用的用户模式 CPU 时间最大值。一旦达到该限制，默认情况下作业中的所有进程将被终止并返回错误代码，该作业中无法继续创建新进程(除非该限制被重置)。作业对象是有信号的，因此此时任何等待该作业的线程都将被释放。若要更改这一默认行为，可调用SetInformationJobObject设置JOBOBJECTEND\_OFJOB\_TIMEINFORMATION结构的 EndOfJobTimeAction 成员，为其传递 JobObjectEndOfJobTimeInformation信息类，并请求通过作业的完成端口发送通知。
3. **每个进程的用户模式CPU时间限制。**可以限制作业中的每个进程总共最多只能使用固定数量的用户模式CPU 时间。当达到该限制后，进程将被终止(没有进行清理的机会)。
4. **作业的处理器相关性。**可以为作业中的每个进程设置处理器相关性掩码。(单个线程可以将自己的相关性调整为作业相关性的任何子集，但进程无法调整自己的进程相关性设置。)
5. **作业组的相关性。**用于设置作业中进程可分配到的组列表。随后，相关性的任何变化都将受到该限制所决定的组选择方式的影响。该限制也被视为作业处理器相关性限制(老版本)的一种可以感知组的新版本，可用于取代老版本的限制。
6. **作业进程优先级类。**可为作业中的每个进程设置优先级类。线程无法增加自己相对于该类的优先级(通常是可以的)。增加线程优先级的企图会被忽略。(调用SetThreadPriority时不返回错误信息，但优先级无法提高。)
7. **默认工作集的最小值和最大值。**定义了作业中每个进程的工作集最小值和最大值(该设置无法用于整个作业范围，每个进程都有自己的工作集最小值和最大值。)
8. **进程和作业已提交虚拟内存的限制。**定义了一个进程或整个作业可提交的虚拟地址空间数量最大值
9. **CPU速率控制。**定义了在被强制限制 (throttling)前，作业可以使用的CPU间数量最大值。
10. **网络带宽速率控制。**定义了在进行限制前，整个作业可使用的出站带宽最大值为了实现 QoS，还可借此为作业所发送的每个网络数据包设置区分服务代码(Differentiated Services Code Point，DSCP)标志，但只能按照层次结构为整个作业设置并且会影响到该作业及其全部子作业。
11. **磁盘I/O带宽速率控制。**与网络带宽速率控制类似，但影响的是磁盘IO，可用于控制带宽本身，或每秒执行的I/O操作数量(Input/Output Operations Per Second, IOPS)可针对系统中的特定磁盘卷或全部卷进行设置。

对于上述很多限制，作业的所有者可以设定指定的阈值，借此在到达阈值后发送通知(或者如果没有注册通知的话，则会直接终止该作业)。此外还可通过速率控制实现一定的容差范围和容差区间，例如每5min 里，允许一个进程的网络带宽用量超限20%并最多持续10s。这样的通知是通过将相应的消息在作业I/O完成端口上排队实现的，详见WindowsSDK文档。

最后，我们还可以通过作业对进程施加用户界面方面的限制。例如禁止作业中的进程打开指向该作业之外其他线程所拥有的窗口句柄，禁止读取或写入剪贴板，禁止通过Windows的SystemParametersInfo 函数更改各种用户界面系统参数等。此类用户接口方面的限制是由Windows子系统GDI/USER 驱动程序 Win32k.sys管理的，并通过注册到进程管理器中一个特殊的调出--“作业调出”来强制实现。通过调用 UserHandleGrantAccess函数，即可允许作业中的所有进程访问特定的用户句柄(例如窗口句柄)，但该函数仅限目标作业之外的其他进程调用。

使用作业

我们可以用 CreateJobObject API创建作业对象。刚创建好的作业不包含任何进程。若要将进程加入某个作业，需要调用 AssignProcessToJobObject，通过多次调用即可将多进程加入同一个作业，或将同一个进程加入多个作业。将进程加入作业的另一种方法是使用进程创建属性PS\_CPJOBLIST手动指定到作业对象的句柄。我们可以指定到作业对象的一个或多个柄，进而实现连接。

对于作业来说，最有趣的API是 SetInformationJobObject，该API可用于设置前文提到的多种限制和设置，并可借此包含供诸如容器 (silo)、DAM或Windows UWP 应用程序等机制使用的内部信息类。这些值可使用QueryInformationJobObject 重新读回，进而相关方了解有关该作业的各种限制。为了让调用方精确了解到底违反了哪些限制，还有必要调用所设置的限制通知。另一个较为有用的函数是 TerminateJobObject,它可以终止作业中的所有进程(等同于对每个进程调用了 TerminateProcess)。

嵌套的作业

从Windows 8和Windows Server 2012开始一个进程可以关联给多个作业，进而让作业产生层次结构。

子作业包含父作业的进程子集。一旦一个进程被添加至多个作业中，系统便会尽可能尝试着产生层次结构。这方面目前有一个限制: 如果有任何一个作业被设置有任何UI方面的限制(SetInformationJobObject 配合 JobbjectBasicUIRestrictions 参数)，那么就无法用这样的作业产生层次结构。

子作业的作业限制，在宽容度方面不能超过其父作业的限制，但可比父作业的限制更严格。举例来说，如果某个父作业设置了内存限制为 100MB，那么任何子作业都无法设置更大的内存限制(这种请求会直接失败)。任何以该作业I/O 完成端口为目标的通知都将发送给该作业，及其全部的祖先作业。(如果只需要向祖先作业发送通知，则该作业本身可以不具备IO完成端口。

对父作业进行的资源记账也会包含由其直接管理的进程,以及子作业中的所有进程使用的资源。当作业被终止 (TerminateJobObject)后，该作业及其子作业中的所有进程均会被终止，并且此时会按照层次结构从最底层的子作业开始终止。

Windows 容器( Server Silo)

创建Silo的能力主要与 SetJobObjectInformationAPI 中一系列未文档化的子类有关换句话说，Silo 其实是一种超级作业 (super-job)，除了前文介绍过的规则和能力外，还有一些额外的其他规则与能力。实际上，作业对象可用于实现我们介绍过的隔离和资源管理能力，并可用于创建Silo。此类作业被系统称为混合作业(hybridjob)。

Silo的隔离

服务器 Silo的第一个基本要素是必须具备自定义的对象管理器根目录 。所有对应用程序可见的命名对象(例如文件、注册表键、事件、互斥体、RPC 端口等)都承载于根命名空间中，借此应用程序就可以创建、定位，并在相互之间共享这些对象。服务器 Silo 有自己的根，这意味着对任何命名对象的访问均可加以控制。这是通过下列3 种方式之一实现的。

1. 创建现有对象的新副本，借此供 Silo 内部进行替代访问
2. 创建到现有对象的符号链接，借此提供直接访问。
3. 创建仅在 Silo内部存在的全新对象，例如可供容器化的应用程序使用的对象。

随后，这些基本能力与(Docker 所用的) Virtual Machine Compute (Vmcompute)服务相结合，并与其他组件相互交互，就提供了完整隔离的下列层。

1. **称为基准OS的基准 Widows 映像 (Windows Image，WIM)文件**。提供了操作系统的单独副本。目前微软提供了 Server Core 映像以及Nano Server 映像。
2. **主机OS的Ntdll.dll 库**。用于覆盖基准OS 映像中的对应内容。这样的做法源自前文提到的情况: 服务器Silo需要使用相同的主机内核和驱动程序，但因为要由Ntdll.dll处理系统调用，所以这个用户模式的组件必须从主机操作系统中实现复用。
3. **由Wcifs.sys筛选器驱动程序提供的沙盒虚拟文件系统**。借此，容器对文件系统进行的临时改动才不会影响到底层的NTFS 驱动器，并可在容器关闭后将改动彻底清除
4. **由VReg 内核组件提供的沙盒虚拟注册表**。借此提供临时使用的注册表根键(及额外的一个命名空间层隔离，因为对象管理器根命名空间只能隔离注册表根，无法隔离注册表根键本身)。
5. **会话管理器** (Smss.exe)。可以用它创建额外的服务会话或控制台会话，这是为了支持容器所需的全新能力。该机制对 SMSS 进行了扩展，使其不仅可以处理额外的用户会话，还可以处理每个已启动容器所需的会话。

Silo的隔离边界

上述组件提供了用户模式的隔离环境。然而因为使用了主机中的Ntdll.dll组件，需要与主机内核与驱动程序通信，所以还必须创建额外的隔离边界，借此由内核提供给不同的Silo。因此每个服务器 Silo 还包含下列这些相互隔离的内容。

1. **微共享(micro shared)的用户数据(符号中的SILO\_USER\_SHARED\_DATA)**。包含自定义系统路径、会话 ID、前台 PID 以及产品类型/套件。这些均为原始 KUSERSHARED\_DATA 的组成要素，但无法从主机提供，因为主机提供的此类信息引用了与主机 OS 映像有关的信息，而非与基准 OS 有关的信息，所以需要单独提供。在查找此类数据时，各种组件与API通过修改将能读取 Silo共享的数据，而非用户共享的数据。注意原始KUSER\_SHAREDDATA 依然存在于原地址中，并借此呈现了主机的相关细节信息而主机状态也可通过这种方式“泄露”到容器状态中。
2. **对象目录根命名空间。**具备自己专用的\SystemRoot符号链接、\Device 目录(所有用户模式组件正是通过这种方式间接访问设备驱动程序的)、设备地图以及 DOS 设备映射(举例来说，用户模式应用程序正是通过这种方式访问网络映射驱动器的)、\Sessions目录等。
3. **API集映射**。基于基准OSWIM的API集架构，而非主机OS文件系统中所存储的架构。加载程序会使用API集映射来判断是否需要，以及需要使用哪个DLL来实现某些函数。但不同 SKU中的这些信息可能有所差异，因此应用程序必须通过基准OS的SKU，而非主机的SKU来做此判断。
4. **登录会话。**主要与SYSTEM和Anonymous 本地唯一ID(LocalUnigue ID，LUID)以及用于在 Silo中描述用户的虚拟服务账户的LUID 有关。从本质上来说，这代表了将要在Smss 所创建的容器服务会话中运行的服务和应用程序所用的令牌
5. **ETW 跟踪和日志记录上下文**。主要用于将 ETW 操作隔离在 Silo 内部，确保不会在容器或主机OS 本身之间暴露或泄露状态信息

Silo 的上下文

内核中的其他组件以及驱动程序(包括第三方驱动程序)可以使用 PsCreateSiloContextAPI设置与Silo有关的自定义数据或将现有对象与 Silo 关联，借此为 Silo 添加上下文数据。每个此类 Silo 上下文会使用一种 Silo插槽索引(slot index)，并将其插入所有运行中的，以及随后创建的服务器 Silo 中借此存储到上下文的指针中。

在创建每个服务器Silo 时，Silo 会收到自己的 Silo 本地存储 (Silo-local Storage，SLS)数组，有些类似TLS。在这个数组中，会有不同项对应于已分配的、用于存储 Silo 上下文的插槽索引。对于相同的插槽索引，每个 Silo 会有不同指针，但索引中始终存储了相同的上下文。

与处理服务器 Silo 共享的用户数据时的做法类似，各种组件和API 通过更新已经可以通过从相关 Silo 的上下文，而非原本使用的全局内核变量来获取数据，借此访问所需数据。

通过存储一种名为PspHostSiloGlobals 的全局内核变量实现的，该变量有自己的插槽本地存储数组，以及被内置内核组件使用的其他 Silo 上下文。当使用 NULL 指针调用不同 Silo API 时，这个“NULL”会被视作“无 Silo”，即使用主机Silo。

Silo 监视器

Silo 监视器提供的一组 API 可以接收通知信息，进而了解什么时候创建或终止了服务器Silo (PsRegisterSiloMonitor、PsStartSiloMonitor、PsUnregisterSiloMonitor)，以及有哪些经存在的 Silo。随后，每个 Silo 监视器可以调用 PsGetSiloMonitorContextSlot 获取自己的插槽索引，并按需将其与PsInsertSiloContext、PsReplaceSiloContext和PsRemoveSiloContext 函数配合使用。借助PsAllocSiloContextSlot 还可分配更多插槽，但只有当某个组件由于某些情况需要存储两个上下文时才有必要这样做。另外，驱动程序也可以借助 PsInsertPermanentSiloContext或PsMakeSiloContextPermanent API来使用“永久”的Silo 上下文，但并不进行引用计数也不会与服务器 Silo 的寿命或所获得的 Silo 上下文数量绑定。一旦被插入，即可使用PsGetSiloContext或PsGetPermanentSiloContext 获取此类 Silo 上下文。

服务器 Silo的创建

在创建服务器 Silo时，首先会使用一个作业对象，因为如前所述，Silo 是作业对象的一个功能。这是通过标准的CreateJobObject API实现的，该API可以关联作业ID，即JID。JID 与进程和线程的ID(PID和TID)来自同一个“数据池”，即客户端ID(Client ID，CID)表。

随后将使用 SetInformationJobObject API创建 Silo信息类。借此即可在代表该作业的EJOB 执行体对象内部设置 Siloflag，并分配之前我们在 EJOB 的 Storage 成员中看到的SLS插槽数组。至此我们就有了一个应用程序 Silo。

随后将使用另一个信息类创建根对象目录命名空间并调用至 SetInformationJobObject。这个新类需要具备 Trusted Computing Base(TCB)特权。由于Silo通常只能由 Vmcompute服务创建，因此这确保了虚拟对象命名空间不会被恶意地用于让应用程序产生混淆并造成潜在破坏。当该命名空间创建完成后，对象管理器会在真正的主机根(\)下创建或打开一个新的Silo目录，并附加JID进而创建新的虚拟根(如\Sios\148\)。随后还会创建KerelObiects.ObiectTypes、GLOBALROOT和DosDevices 对象。接着会将该根存储为 Silo上下文，并使用PsObjectDirectorySiloContextSlot 中定义的任何插槽索引，该索引是由对象管理器在启动时分配的。

接下来需要将该 Silo转换为服务器 Silo，这是通过再次调用 SetInformationJobObiect并使用另一个信息类实现的。随后会运行内核中的 PspConvertSiloToServerSilo函数，初始化ESERVERSILO\_GLOBALS 结构。借此将初始化 Silo的共享用户数据、API集映射、SystemRoot 以及各种Silo上下文，例如SRM用于识别 Lsass.exe 进程的上下文。进行这个转换的同时，Silo监视器将完成注册操作并开始回调，进而准备好接收通知，这一过程也可能添加自己的 Silo上下文数据。

最后一步，需要“启动”这个服务器 Silo，为此需要为它初始化一个新的服务会话。这个会话可以看作该服务器 Silo的会话0。这是通过Smss SmApiPort发送ALPC 消息实现的，消息中包含了到 Vmcompute 所创建作业对象的句柄，当然该对象现在已经成为一个服务器 Silo作业对象。和创建真正的用户会话时类似，Smss 为自己创建一个克隆的副本，不过这次会在创建时将副本关联给作业对象。借此即可将新的 Smss 副本附加给服务器 Silo 内所有容器化的元素。Smss 会认为这就是会话0，并开始承担自己的本职工作，例如启动 Csrss.exe、Wininit.exe、Lsass.exe 等。“启动”过程将照常进行，先启动Wininit.exe，随后启动服务控制管理器 (Services.exe)，接着启动所有自动启动的服务，以此类推。

辅助功能

为了让设备驱动程序能够正常访问，驱动程序还必须了解并注册自己的 Silo监视器,随后使用通知创建自己的每 Silo 设备对象。内核提供的 API PsAttachSiloToCurrentThread(以及对应的PsDetachSiloFromCurrentThread)可以暂时设置Silo的ETHREAD对象字段传入作业对象。但这会导致所有访问，包括对象管理器的访问，都被视作来自 Silo 本身例如命名管道驱动程序可以使用这种功能在\Device命名空间下创建NamedPipe对象，但实际上会被包含在\Silos\JID\中。

容器模板

可使用一种特殊容器模板文件来定义 Silo命名空间、注册表和文件系统。在通过添加/删除Windows功能对话框启用Windows容器功能后，该文件将默认保存在%SystemRoot%\System32\Containers\wsc.def 目录下。该文件描述了对象管理器、注册表命名空间以及相关规则，并可按需定义到主机上真实对象的符号链接。此外该文件还描述了要使用的作业对象、卷挂载点和网络隔离策略。