第 14 章　Windows 管理规范

我们一直期望但是又害怕写这一章。Windows 管理规范（Windows Management Instrumentation，WMI）可能是微软提供给管理员使用最优秀的工具之一。但同时它也是这个公司曾经给我们造成最多问题的部分。WMI 可以从计算机中收集大量系统信息。但有时候这些信息不易看懂，另外文档也不够友好。在本章，我们将从 PowerShell 的角度介绍 WMI，以及 WMI 的工作方式和一些不完美的地方，以便全面揭示你将会遇到的问题。

需要强调的是，WMI 是一个外部技术；PowerShell 仅仅与其接口交互而已。本章的重点将放在 PowerShell 如何与 WMI 交互，而不是 WMI 的底层实现机制。如果你不想深入探讨 WMI，我们在本章结尾提供了一些建议。PowerShell 已经在最大程度减少你需要与 WMI 交互的部分做出了改进。

14.1　WMI 概要

典型的 Windows 计算机包含数万个管理信息，WMI 会将这些信息整理成易于访问的形式。

在最顶层，WMI 被组织成命名空间（namespaces）。可以把命名空间想象为关联到特定产品或技术的一个文件夹。比如，“root\CIMv2”，该命名空间包含了所有 Windows 操作系统和计算机硬件信息。而“root\MicrosoftDNS”命名空间包含了所有关于 DNS 服务器（假设你已经在计算机中安装了该角色）的信息。在客户端计算机上，“root\SecurityCenter”包含了关于防火墙、杀毒软件和反流氓软件等工具的信息。

注意：“root\SecurtityCenter”的内容根据你计算机上的已安装程序的情况而有所不同，新版本的 Windows 使用“root\SecurityCenter2”代替它，这是 WMI 使人困惑的例子之一。

图 14.1 展示了通过微软管理控制台（Microsoft Management Console，MMC）的 WMI 控制单元在我本机上产生的一些命名空间。

在命名空间中，WMI 被分成一系列的类，每个类都是可用于 WMI 查询的管理单元。比如，在“root\SecurityCenter”中的“Antivirus-Product”类被设计用于保存反间谍软件的信息；在“root\CIMv2”中的“Win32_LogicalDisk”类被设计用于保存逻辑磁盘的信息。但是即使一个计算机上存在某个类，也不代表计算机实际上安装了对应组件。比如无论是否安装了磁带驱动程序。“Win32_TapeDrive”类在所有的 Windows 版本上都存在。

图 14.1 浏览 WMI 命名空间

注意：再一次提醒，不是所有的计算机都包含相同的 WMI 命名空间或类。比如，新版本的 Windows 存在“Root\SecurityCenter2”命名空间，而不是“Root\SecurityCenter”命名空间；而前者在新版本的计算机中包含了所有信息。

下面看一下从“root\SecurityCenter2”中查询“AntiSpywareProduct”，你可以查看返回结果。

```
PS C:\> Get-CimInstance -Namespace root\securitycenter2 -ClassName antispyw
areproduct
```

注意：本示例需要 PowerShell v3 及以上版本，我们稍微介绍一下“Get-CimInstance”命令。

当你有一个或多个可管理组件时，你可以看到在对应的类中有相同数量的实例（instances）。一个实例由类代表了一个现实世界的事件。如果你的计算机只有一个单一的 BIOS，那么在“root\CIMv2”中会有一个关于“Win32_BIOS”的实例。如果计算机安装了 100 个后台服务，你会看到 100 个“Win32_Service”的实例。请注意，在

“root\CIMv2”中的类型一般以“Wim32_”（即使在 64 位系统中亦然）或“CIM_”（Common Information Model 的缩写，是 WMI 建立的标准）开头。在其他命名空间中，这些类名前缀很少出现。不同命名空间下的类型名称重复也存在可能性，虽然这种情况很少，但在 WMI 中允许存在，因为每个命名空间实际上是一种有边界的容器。当你引用一个类时，你同时需要引用其命名空间，以便 WMI 知道从哪里找到对应的类，从而避免因为多个重名但不属于同一个命名空间的类造成混乱。

所有这些实例、类和其他不可名状的东西统称为 WMI 仓库（WMI Repository）。在旧版本的 Windows 中，WMI 仓库有时会损坏而导致不可用，必须通过重建恢复。但从 Win 7 开始，这种情况越来越少见。

表面看上去，使用 WMI 十分简单：你只需要指出哪个类包含你要的信息，然后从 WMI 中查询类的实例，最后检查实例的属性得知其管理信息。有时候需要实例执行一个方法，从而启动一个动作（action）或开始一个配置变更（configuration change）。

14.2 关于 WMI 的坏消息

在 WMI 大部分生命周期中，微软都没有把过多的精力放在对其内部控制上（最近有所好转）。微软为 WMI 制定了一系列的编程标准，但是产品组或多或少把精力放在如何实现类和是否对其文档化。结果就是使得 WMI 变得混乱。

例如，在“root\CIMv2”命名空间中，只有很少的类提供了让你修改配置设置的方法（methods）。因为属性是只读的，意味着你必须使用方法来修改。如果对应的方法不存在，你就不能使用 WMI 来修改这些类。当 IIS 团队采用 WMI（IIS 第六版），它们针对大量的元素进行了并行类的实现。比如一个网站，可以用一个具有典型只读属性的类表示，但是同时也提供了第二个类用于修改属性。这种情况下很容易因为文档质量不佳而导致混乱，特别是 IIS 团队倾向于使用自身提供的工具。IIS 团队已经放弃了把 WMI 作为管理接口的做法，并从 v7.5 开始把注意力集中在 PowerShell Cmdlets 上，并且用一个 PSProvider 类替代 WMI。

微软从来没规定某个产品必须使用 WMI，或者如果这个产品使用了 WMI，必须公开 WMI 的每个可能的部分。微软的 DHCP 服务可以访问到 WMI。正如旧版本的 Windows 服务器一样，你可以查询网卡的配置，但是不能查到连接速度，因为这些信息不支持通过 WMI 查询。同时，虽然大部分“Win32_”的类都有很好的文档支持，但其他命名空间下的类大部分都没有相关文档，WMI 不支持类搜索，因此查找这些类对你来说就变得费时费力（在下一节将告诉你如何减少这种影响）。

但是微软也对此进行了改善，微软正在努力使 PowerShell Cmdlets 尽可能完成更多的管理任务。比如，过去 WMI 仅用于某种特定的编程方式重启远程计算机，这个方法由“Win32_OperatingSystem”类实现。而现在，PowerShell 提供了名称为“Restart-

Computer”的 Cmdlet 来实现。在某些情况下，Cmdlets 内部会通过 WMI 实现，而无须直接调用 WMI。Cmdlets 能提供更一致的接口，并且这些接口大部分都有很好的文档支持。虽然 WMI 不会消失，但你时不时还是需要在某些场景用到 WMI。

实际上，在 PowerShell v3 及后续版本中（尤其是在新版本的 Windows 中，从 Windows 8 或 Windows Server 2012 开始），你会留意到大量“CIM”命令，如图 14.2 所示（作为“Get-Command”输出的一部分）。在大部分情况下，这些命令都是对 WMI 的某些部分进行了封装，从而提供了更加以 PowerShell 为中心的方式与 WMI 交互。你可以像使用其他 Cmdlet 一样使用这些 Cmdlet，包括对这些 Cmdlet 使用 Help 命令。这使得使用这部分 Cmdlet 和使用其他 PowerShell 中的 Cmdlet 的体验变得一致，也便于隐藏一些底层的 WMI 的复杂性。

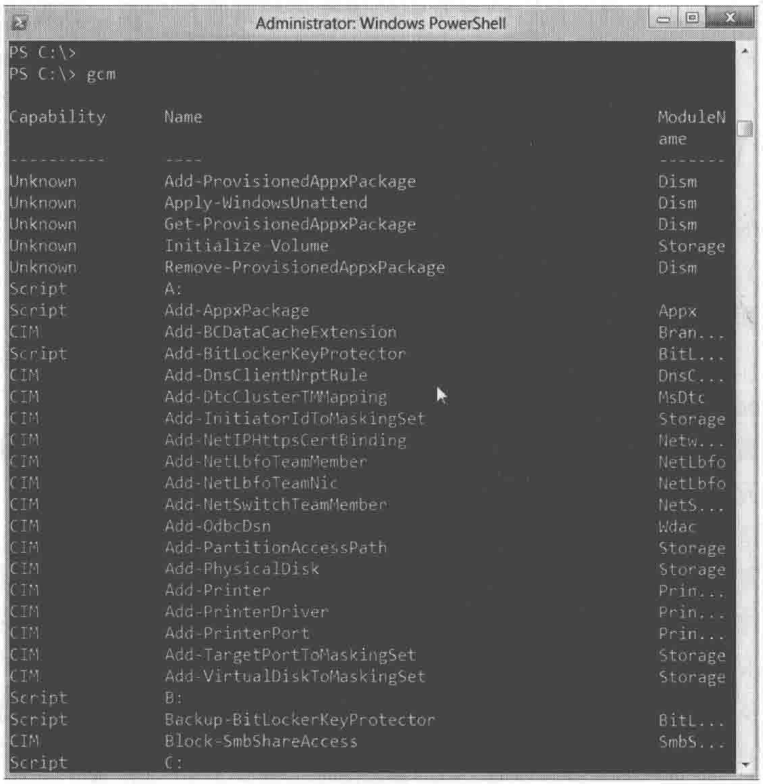

图 14.2 用于封装 WMI 类的“CIM”命令

14.3 探索 WMI

最佳的 WMI 入门恐怕是暂时抛开 PowerShell 并从 WMI 自身出发。这里我们使用 WMI 探索工具。不幸的是，这些工具就像季节更替那样经常更换，所以我们犹豫是否该告诉你某个特定工具。你可以尝试在搜索引擎中搜索 WMI explorer 并查看结果。你也

可以尝试访问 http://powershell.org/wp/2013/03/08/wmi-explorer/。我们可以从这类工具中得到大部分所需的关于 WMI 的信息。当然，这个工作要求耐心和不少的浏览量——这并不是最佳方法，但是我们最终还是选择了这个工具。

由于每台计算机上的 WMI 命名空间和类都不尽相同，所以你需要把工具直接在准备查询的机器上运行，以便看到对应机器的 WMI 仓库。

现在我们需要查询一组计算机并从中得知它们的图标间距设置。这个任务依赖于 Windows 桌面，并且是操作系统的核心部分，所以我们从“root\CIMv2”类开始，显示在 WMI 浏览器左侧的树型视图中（见图 14.3）。单击命名空间并在右侧查看对应的类，我们知道需要“Desktop”这个关键字。滚动右侧窗口的滚动条，最终锁定“`Win32_Desktop`”并单击它。此时下方窗体将展示其对应明细，然后我们选择【Properties】（属性）选项卡并查看其内容。在距离下边大约三分之一的地方，找到“`IconSpacing`”，其值为整数。

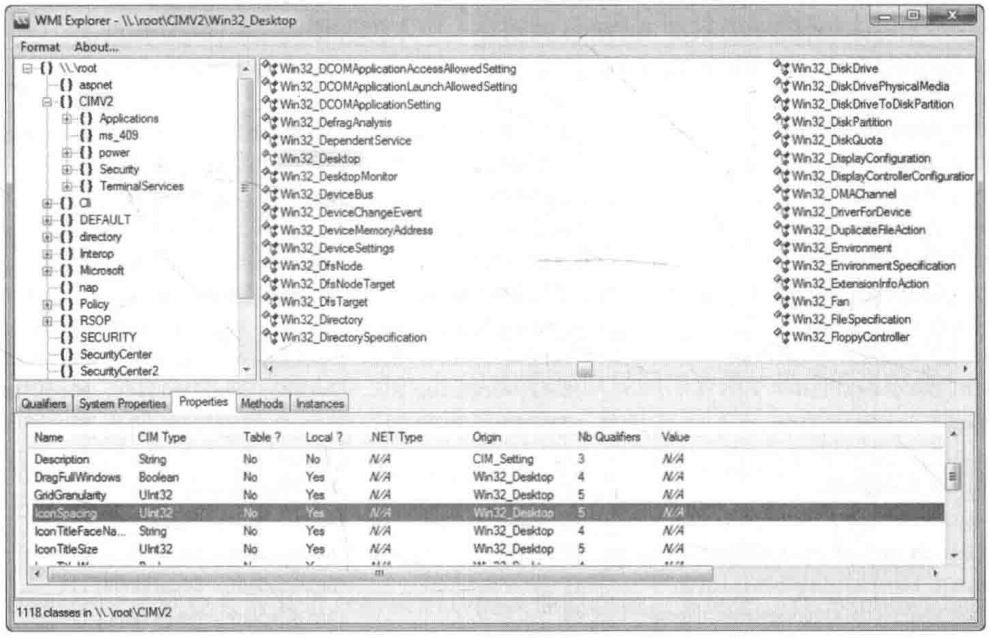

图 14.3 WMI 浏览器

显而易见，搜索引擎是查询所需类信息的另一种好方法。我们倾向在 icon spacing 之前添加 wmi 前缀作为关键字进行搜索，一般在查看几个例子之后就可以找到大概方向。这些例子可能是与 VBScript 相关的，或者是类似 C#或 Visual Basic 等.NET 语言相关的。不过这不重要，因为我们只是在查找 WMI 类名称。比如，我们在搜索引擎（例子使用 Google）中查找“wmi icon spacing”，可能会首先显示 http://stackoverflow.com/questions/202971/formula-or-api-for-calulating-desktop-icon-spacing-on-windows-xp 作为第一个结果。在该页中我们找到一些 C#代码。

```
ManagementObjectSearcher searcher = new
    ➥ManagementObjectSearcher("root\\CIMV2","SELECT * FROM Win32_Desktop");
```

对此，我们并不知道上述代码的意思，但是“Win32_Desktop”看上去像是个类名称。接下来我们就要查询该类名，这样的查询通常会帮助我们找到一些存在的文档。我们会在本章后续介绍一些关于文档的问题。

另外一个途径是使用 PowerShell 本身。比如，假设我们想知道一些关于磁盘的信息，那么需要从猜测正确的命名空间开始。但是我们已经知道“root\CIMv2”包含了所有 OS 核心和硬件设备的信息，所以可以使用下面的命令。

```
PS C:\> Get-WmiObject -Namespace root\CIMv2 -list | where name -like '*dis *'

   NameSpace:ROOT\CIMv2
Name                                Methods              Properties
----                                -------              ----------
Win32_DisplayConfiguration          {}                   {BitsPerPel,Caption,...}
Win32_DisplayControllerConfigura... {}                   {BitsPerPixel,Caption,...}
CIM_DiskSpaceCheck                  {Invoke}             ...}
CIM_DiscreteSensor                  {SetPowerState, R... {AcceptableValues, Availability,...
CIM_Display                         {SetPowerState, R... {Availability, Caption,...
CIM_DiskDrive                       {SetPowerState, R... {Availability, Capabilities,...
Win32_DiskDrive                     {SetPowerState, R... {Availability, BytesPerSector,...
CIM_DisketteDrive                   {SetPowerState, R... {Availability, Capabilities,...
CIM_LogicalDisk                     {SetPowerState, R... {Access, Availability, BlockSize,...}
Win32_LogicalDisk                   {SetPowerState, R... {Access, Availability, BlockSize,...}
Win32_MappedLogicalDisk             {SetPowerState, R... {Access, Availability, BlockSize,...}
CIM_DiskPartition                   {SetPowerState, R... {Access, Availability, BlockSize,...}
Win32_DiskPartition                 {SetPowerState, R... {Access, Availability, BlockSize,...}
Win32_LogicalDiskRootDirectory      {}                   {GroupComponent, PartComponent}...
Win32_DiskQuota                     {}                   {DiskSpaceUsed, Limit,...}
Win32_LogonSessionMappedDisk        {}                   {Antecedent, Dependent}...
CIM_LogicalDiskBasedOnPartition     {}                   {Antecedent, Dependent,...
Win32_LogicalDiskToPartition        {}                   {Antecedent, Dependent,...
CIM_LogicalDiskBasedOnVolumeSet     {}                   {Antecedent, Dependent,...
Win32_DiskDrivePhysicalMedia        {}                   {Antecedent, Dependent}...
CIM_RealizesDiskPartition           {}                   {Antecedent, Dependent,...
Win32_DiskDriveToDiskPartition      {}                   {Antecedent, Dependent}
Win32_OfflineFilesDiskSpaceLimit    {}                   {AutoCacheSizeInMB,...
Win32_PerfFormattedData_Counters... {}                   {Caption, Description,...
Win32_PerfRawData_Counters_FileS... {}                   {Caption, Description,...
Win32_PerfFormattedData_Distribu... {}                   {AckMessagesReceivedPersecond,...
Win32_PerfRawData_DistributedRou... {}                   {AckMessagesReceivedPersecond,...
Win32_PerfFormattedData_MSDTC_Di... {}                   {AbortedTransactions,...
Win32_PerfRawData_MSDTC_Distribu... {}                   {AbortedTransactions,
Win32_PerfFormattedData_MSSQLSER... {}                   {Caption, Description,...
Win32_PerfRawData_MSSQLSERVER_SQ... {}                   {Caption, Description,...
```

```
Win32_PerfFormattedData_PeerDist... {}          {BITSBytesfromcache,...
Win32_PerfRawData_PeerDistSvc_Br... {}          {BITSBytesfromcache,...
Win32_PerfFormattedData_PerfDisk... {}          {AvgDiskBytesPerRead,...
Win32_PerfRawData_PerfDisk_Logic... {}          {AvgDiskBytesPerRead,...
Win32_PerfFormattedData_PerfDisk... {}          {AvgDiskBytesPerRead,...
Win32_PerfRawData_PerfDisk_Physi... {}          {AvgDiskBytesPerRead,...
```

最终我们找到“Win32_LogicalDisk”。

注意：这些名称以“CIM_”开头的类通常是基类，你通常不能直接使用这些类。“Win32_”版本的类是 Windows 特有的，并且这种前缀仅存在于特定命名空间——其他命名空间不使用以 CIM_作为前缀这种命名方式。

14.4 选择你的武器：WMI 或 CIM

在 PowerShell v3 及后续版本中，有两种与 WMI 交互的方式。

- 所谓的“WMI Cmdlets”，例如“Get-WmiObject”与“Invoke-WmiMethod”——这些都是遗留命令，意味着它们依旧能工作，但是微软不会对它们进行后续开发投入。它们与远程过程调用（RPC）交互，也就是说，只有在防火墙支持状态审查时才能通过防火墙（实际上很难）。
- 新版的“CIM Cmdlets”，例如“Get-CimInstance”与“Invoke-CimMethod”——它们或多或少等价于旧版本的“WMI Cmdlets”，但是它们通过 WS-MAN（由 Windows 远程管理服务实现）交互，替代原有的 RPCs。这是微软的主方向，执行“Get-Command-noun CIM*”可以显示很多微软提供的这类命令的功能。

毫无疑问，这些命令的后端同样是 WMI，其差异在于如何交互和如何被使用。在没有安装 PowerShell 的旧版本系统中，或者没有启用 Windows 远程管理功能的系统中，WMI Cmdlets 依旧能工作（这个功能从 Windows NT 4.0 SP3 开始引入）。对于已经装有 PowerShell 和启用了 Windows 远程管理服务的新系统，CIM Cmdlets 提供最佳体验——微软也会对其进行持续的功能及性能改进。实际上，在 Windows Server 2012 R2 以及更新版本中，旧版的 WMI 默认为禁用状态，因此尽可能使用 CIM。除此之外，CIM cmdlet 可以使用旧版的 RPC（或 DCOM）协议通讯，因此即使与老机器进行通讯时，你也可以仅使用 CIM cmdlet。

14.5 使用 Get-WmiObject

通过“Get-WmiObject”命令，你可以指定一个命名空间、一个类名称甚至远程计算机的名称以及备用凭据名。如果需要，还可以从指定的计算机中查询该类的所有实例。

如果需要减少类实例的返回结果，甚至可以提供筛选条件实现，可以使用下面的语法获取一个命名空间中的类列表。

```
Get-WmiObject -namespace root\cimv2 -list
```

注意，命名空间名字使用的是反斜杠，不是斜杠。

也可以通过指定命名空间和类型查询一个类。

```
Get-WmiObject -namespace root\cimv2 -class win32_desktop
```

其中“root\CIMv2”命名空间是 Windows XP SP2 及后续版本的系统默认命名空间，所以如果你的类在该命名空间中，可以不显式指定。同时，“-class”是位置参数，也就是说，如果你把类名称放到第一个位置，它依旧能正常工作。

这里有两个例子，其中一个使用 Gwmi 别名代替完整的 Cmdlet 名称。

```
PS C:\> Get-WmiObject win32_desktop
PS C:\> gwmi antispywareproduct -namespace root\securitycenter2
```

动手实验：从现在开始，你应该动手运行每个我们展示的命令。对于涉及远程计算机名称的，如果没有另外一台机器可供测试，可以用 localhost 替代。

对于许多 WMI 类，PowerShell 的默认配置已经设定了需要展示的属性。“Win32_OperatingSystem”是一个很好的例子，因为它默认仅在列表中展示了 6 个属性。请记住，你总能把 WMI 对象用管道传输到“Gm”或“Format-List *”中，以便查看所有可用的属性。“Gm”总是列出所有可用的方法。下面是一个示例。

```
PS C:\> Get-WmiObject win32_operatingsystem | gm

   TypeName: System.Management.ManagementObject#root\cimv2\Win32_Operating
System

Name                                MemberType     Definition
----                                ----------     ----------
Reboot                              Method         System.Managemen...
SetDateTime                         Method         System.Managemen...
Shutdown                            Method         System.Managemen...
Win32Shutdown                       Method         System.Managemen...
Win32ShutdownTracker                Method         System.Managemen...
BootDevice                          Property       System.String Bo...
BuildNumber                         Property       System.String Bu...
BuildType                           Property       System.String Bu...
Caption                             Property       System.String Ca...
CodeSet                             Property       System.String Co...
CountryCode                         Property       System.String Co...
CreationClassName                   Property       System.String Cr...
```

为了节省空间，这里截断了部分输出结果。如果想看完整结果，请自行执行命令。

另外，“-filter”参数允许你通过指定的规则查询特定实例。该参数使用起来有点棘手。下面的示例可以看出其最坏情况下的结果。

```
PS C:\> gwmi -class win32_desktop -filter "name='COMPANY\\Administrator'"

__GENUS                : 2
__CLASS                : Win32_Desktop
__SUPERCLASS           : CIM_Setting
__DYNASTY              : CIM_Setting
__RELPATH              : Win32_Desktop.Name="COMPANY\\Administrator"
__PROPERTY_COUNT       : 21
__DERIVATION           : {CIM_Setting}
__SERVER               : SERVER-R2
__NAMESPACE            : root\cimv2
__PATH                 : \\SERVER-R2\root\cimv2:Win32_Desktop.Name="COMPANY
                         \\Administrator"
BorderWidth            : 1
Caption                :
CoolSwitch             :
CursorBlinkRate        : 530
Description            :
DragFullWindows        : False
GridGranularity        :
IconSpacing            : 43
IconTitleFaceName      : Tahoma
IconTitleSize          : 8
IconTitleWrap          : True
Name                   : COMPANY\Administrator
Pattern                : 0
ScreenSaverActive      : False
ScreenSaverExecutable  :
ScreenSaverSecure      :
ScreenSaverTimeout     :
SettingID              :
Wallpaper              :
WallpaperStretched     : True
WallpaperTiled         : False
```

对于该命令和输出结果，需要注意下面事项。

- 筛选条件通常被双引号包住。
- 筛选比较操作符并不使用 PowerShell 的常规操作符“-eq”或“-like”，而使用更加传统、更加编程化的操作符，比如=、>、<、<=、>=和<>。可以使用关键字“LIKE”作为操作符，但在匹配值时必须使用“%”作为字符通配符，如“NAME LIKE '%administrator%'”。注意，这里不能像 PowerShell 的其他地方一样使用*作为通配符。

- 字符串匹配是以单引号包住，这也是筛选表达式的最外层的引号是双引号的原因。
- 避免在 WMI 中使用反斜杠。当你需要使用文本的反斜杠时，你必须使用两个反斜杠替代。
- Gwmi 的输出结果总会包含大量系统属性。PowerShell 的默认显示配置通常会隐藏这些属性，如果你故意列出的这些值或类没有默认配置，那么这些值会被显示出来。系统属性名称以双下划线开始。这里有两个非常有用的属性。

  __SERVER：包含被查询的实例所在的计算机名称。当所查询的 WMI 信息来自于多台计算机时非常有用，该属性与易于记忆的“PSComputerName”属性功能一致。

  __PATH：是实例本身的绝对引用。如果需要的话，可以用来查询实例。

该 Cmdlet 不仅可以从远程计算机中查询信息，也可以从多台计算机中检索，可以使用任何可以生成一个包含计算机名称或 IP 列表的技术。

```
PS C:\> Gwmi Win32_BIOS -comp server-r2,server3,dc4
```

计算机名称按顺序连接，如果某一台计算机不可用，该 Cmdlet 会产生一个错误，并跳过这台计算机，并转向下一台计算机。对于不可用的计算机，Cmdlet 通常需要等待直到发生超时，意味着 Cmdlet 可能会暂停 30～45 秒之后才决定放弃这台计算机，然后产生错误并继续向后连接。

一旦你查询到一个 WMI 实例的集合后，就可以把它们用管道连接到任何以“-Object”Cmdlet、“Format-”Cmdlet 或“Out-”“Export-”“ConvertTo-”开头 Cmdlet 中。你可以使用自定义表格显示“Win32_BIOS”类的信息。

```
PS C:\> Gwmi Win32_BIOS | Format-Table SerialNumber,Version -auto
```

在第 10 章中，我们已经介绍了如何使用“Format-Table”Cmdlet 生成定制列。当你想从一台给定计算机中查询一些 WMI 类并集成到一个表时，该技术在这里就可以派上用场。此时你可以创建一个关于表的自定义列，并用列的表达式执行一个全新的 WMI 查询。语法如下，虽然看上去有点头大，但是结果却能让人满意。

```
PS C:\> gwmi -class win32_bios -computer server-r2,localhost |
➥ format-table @{label='ComputerName';expression={$_.__SERVER}},
➥ @{label='BIOSSerial';expression={$_.SerialNumber}},
➥] @{label='OSBuild';expression= {gwmi -class \win32_operatingsystem
➥ -computer $_.__SERVER | select-object -expand BuildNumber}} -autosize

ComputerName BIOSSerial                                              OSBuild
------------ ----------                                              --------
SERVER-R2    VMware-56 4d 45 fc 13 92 de c3-93 5c 40 6b 47 bb 5b 86 7600
```

如果将上面的代码复制到 PowerShell ISE 中并格式化，则会更容易理解：

```
gwmi -class win32_bios -computer server-r2,localhost |
format-table
@{label ='ComputerName';expression={$_.__SERVER}},
@{label ='BIOSSerial';expression={$_.SerialNumber}},
@{label ='OSBuild';expression={
    gwmi -class win32_operatingsystem -comp $_.__SERVER |
    select-object -expand BuildNumber}
  } -autosize
```

下面是上述示例所产生的结果。

- “Get-WmiObject”从两台计算机中查询“Win32_BIOS”信息。
- 结果被管道传输到“Format-Table”。“Format-Table”被要求创建三个自定义列。

  第一列：名称为 ComputerName，使用“Win32_BIOS”实例中的“__SERVER”系统属性得出。

  第二列：名称为 BIOSSerial，使用“Win32_BIOS”实例中的“SerialNumber”属性得出。

  第三列：名称为 OSBuild。该列执行一个新的“Get-WmiObject”查询，从“Win32_BIOS”实例的“__SERVER”属性中查询“Win32_OperatingSystem”类。然后把结果用管道传输到“Select-Object”中，这些信息来自于“Win32_OperatingSystem”实例的“BuildNumber”属性的内容，并用于填充 OSBuild 列的值。

语法有点复杂，但是提供了满意的结果。并且作为一个很好的例子展示了如何通过一些精心挑选的 PowerShell Cmdlet 实现你要的结果。

我们已经提醒过，一些 WMI 类包含方法。你可以在第 16 章中看到如何使用这些方法。这些方法相对难懂，所以我们独立出一章来介绍。

14.6 使用 Get-CimInstance

Get-CimInstance 是 PowerShell v3 引入的新命令，与“Get-WmiObject”有很多相似的地方，但是也有几个语法上的差异。

- 你需要使用“-ClassName”代替“-Class”（虽然你只需要输入-Class，但是如果你只记住了该参数名称的话，这没有问题）。
- 没有用于列出命名空间中的所有类的“-List”参数。而是使用“Get-CimClass”并搭配“-Namespace”参数获取类列表。
- 没有“-Credential”参数；如果你需要从远程计算机查询并被要求提供替代凭据，需要通过“Invoke-Command”（前面章节已介绍）发送“Get-CimInstance”。

比如：

```
PS C:\> Get-CimInstance -ClassName Win32_LogicalDisk

DeviceID   DriveType  ProviderName    VolumeName          Size        FreeSpace
--------   ---------  ------------    ----------          ----        ---------
A:         2
C:         3                                              687173...   580806...
D:         5                           HB1_CCPA_X64F...   358370...   0
```

如果你需要使用替代凭据查询远程计算机，可以使用类似下面的命令。

```
PS C:\> invoke-command -ScriptBlock { Get-CimInstance -ClassName win32_proc ess }
    ➥-ComputerName WIN8 -Credential DOMAIN\Administrator
```

14.7 WMI 文档

前面提到过，搜索引擎通常是查找已有 WMI 文档的最佳方式。虽然“Win32_”类在微软的 MSDN 网站上有很好的文档记录，但是搜索引擎依旧是查询正确页面最容易的方式。你只需要把类名在 Google 或者 Bing 上搜索，通常第一条就会导航到 http://msdn.microsoft.com/。

14.8 常见误区

之前的 10 章介绍了如何使用内置的 PowerShell 帮助，所以你可能更倾向于在 PowerShell 中运行类似“help win32_service”的命令。不幸的是，在这里行不通。操作系统本身不包含任何 WMI 信息，所以 PowerShell 的帮助功能不能实现你的期望。你可能希望从网上的其他管理员和程序员分享的经验中而不是从微软信息中得到大部分你要的信息，比如查询“root\SecurityCenter”。不幸的是，你不会在结果中找哪怕一个微软的文档页。

WMI 的筛选规则的差异也是其中一个误区。不管任何时候，你需要在所有可用实例中筛选信息时都应该提供过滤条件。但是别忘了，筛选语法存在不同。筛选语法是传递到 WMI，而不是由 PowerShell 处理，所以你必须使用 WMI 规定的语法去替代内置的 PowerShell 操作符。

另外一些在我们的学生中常见的关于 WMI 的误区是，虽然 PowerShell 提供了从 WMI 查询信息的简易途径，但是 WMI 并不集成在 PowerShell 中。WMI 是一个外部技术，有自己的规则和工作方式。WMI 虽然可以在 PowerShell 内部使用，但和其他 Cmdlets 不一样，因为这些 Cmdlets 完全集成在 PowerShell 内。所以请注意 WMI 的这些误区。

14.9 动手实验

注意：对于本次动手实验来说，你需要运行 PowerShell v3 或更新版本 PowerShell 的 Windows 计算机。

花点时间完成下面的动手任务。使用 WMI 的难处主要在于如何找到你所需要的类的信息，所以本实验中大部分时间会花在查找正确的类上面。尝试花点儿时间在思考关键字上（我们会给一些提示），并使用 WMI explorer 快速查找这些类（WMI Explorer 按字母顺序排列类，便于我们验证自己的猜测）。记住，PowerShell 的帮助系统并不能帮助你查找 WMI 类。

1．使用什么类可以查看一个网卡的当前 IP 地址？这个类是否有什么方法可用于释放 DHCP 租期？（提示：network 是一个不错的关键字。）

2．创建一个显示计算机名称、操作系统版本号、操作系统描述（标题）和 BIOS 序列号的表格。（提示：你已经见过这个技术，但你需要首先查询 OS 类，然后再查询 BIOS。）

3．使用 WMI 查询关于热修复补丁（hotfixes）的列表。（提示：微软通常把这些引用为 quick fix engineering。）该列表内容与“`Get-Hotfix`”的输出结果有何不同？

4．显示服务列表，在列表中包含它们的当前状态、启动模式和启动账号信息。

5．使用 CIM cmdlet，显示在 SecurityCenter2 命名空间内，并且以 Product 作为路径列表一部分的所有可用列。

6．当你发现所需使用的名称后，使用 CIM cmdlet 显示所有反间谍软件。你也可以再查询反病毒产品。

动手实验：当你完成这个实验后，尝试完成附录中的实验回顾 2。

14.10 进一步学习

WMI 是一个巨大的、复杂的技术，其中一些技术足以编写一整本书。实际上确实有人这样做了：*PowerShell and WMI*，作者是微软 MVP Richard Siddaway(Manning，2012)。该书提供了大量示例，并讨论了关于 PowerShell v3 引入关于 CIM Cmdlets 的新功能。如果有深入学习 WMI 的意愿，我强烈建议阅读这该书。

如果你发现 WMI 实在很难理解，别担心。这是正常反应。但是我们有一些好消息：在 PowerShell v3 及后续版本中，你可以在没有显式调用的情况下使用 WMI。因为微软已经开发了数百个 Cmdlets 用于封装 WMI。这些 Cmdlets 提供了帮助信息、可发现性、示例和所有其他 Cmdlets 能提供给你的好处，但是它们内部实现还是使用 WMI。这样能更好地使用 WMI 的强大功能，又避免处理一些使人困惑的元素。

14.11 动手实验答案

1. 你可以使用 Win32_NetworkAdapterConfiguration 类。

   如果对该类运行 Get-Wmiobject 并通过管道传输给 Get-Member，你可以看到大量 DHCP 相关的方法。你还可以使用 CIM cmdlet 发现这些。

   ```
   Get-CimClass win32_networkadapterconfiguration | select -expand
   methods | where Name -match "dhcp"
   ```

2. ```
   get-wmiobject win32_operatingsystem | Select BuildNumber,Caption,
   @{l='Computername';e={$_.__SERVER}},
   @{l='BIOSSerialNumber';e={(gwmi win32_bios).serialnumber}}|ft-auto
   ```

   或者使用 CIM cmdlets：

   ```
   get-ciminstance win32_operatingsystem | Select BuildNumber,Caption,
   @{l='Computername';e={$_.CSName}},
   @{l='BIOSSerialNumber';e={(get-ciminstance win32_bios).serialnumber
   }} | ft -auto
   ```

3. ```
   get-wmiobject win32_quickfixengineering
   ```

4. 你可以看到结果是类似的。

5. ```
   get-wmiobject win32_service | Select Name,State,StartMode,StartName
   ```

   或

   ```
   get-ciminstance win32_service | Select Name,State,StartMode,StartName
   ```

6. ```
   get-cimclass -namespace root/SecurityCenter2 -ClassName *product
   get-ciminstance -namespace root/SecurityCenter2 -ClassName
   AntiSpywareProduct
   ```