第 16 章　同时处理多个对象

PowerShell 存在的主要意义在于自动化管理，这通常意味着你将会在多个目标上同时执行任务。你或许希望重启多台计算机，重新配置多个服务，修改多个邮箱等。在本章，你将学到 3 种技术：批处理 Cmdlet、WMI 方法以及对象枚举，用于完成这些以及其他多目标任务。同时，你需要知道本章中的大多数示例无法在 Linux 或 macOS 平台下工作；这些示例（至少是当前）仅在 Windowsk 中有效。但是无论你使用的是哪种操作系统，这里谈到的概念与技术并无不同。

16.1　对于大量管理的自动化

我们当然知道本书不是一本关于 VBScript 的书，但我们希望使用一个 VBScript 的例子简单阐述多目标管理的方式——Don 喜欢将“批量管理”称为过去的方式（你不需要输入下面代码并运行——我们讨论的仅仅是方法，而不是结果）。

```
For Each varService in colServices
  varService.ChangeStartMode("Automatic")
Next
```

上述方法不仅仅是在 VBScript 中很流行，在编程的世界中都很流行。下面的步骤阐述了该段代码的作用。

（1）假设变量 colServices 包含多个服务。因为获取服务的方式有很多，所以先不关心 colServices 如何被赋值。重要的是，你已经获取到服务并将其存入变量。

（2）ForEach 结构将会遍历或枚举所有服务，一次一个。每次都将服务存入变量 varService。使用该结构，varService 将会仅包含一个服务。如果 colServices 包含 50 个服务，则该循环体结构将会执行 50 次，每一次 varService 变量都会只包含这 50 个服务中的一个。

(3)在循环结构中，每次都执行一个方法——在本例中是 ChangeStartMode() 方法——完成某些工作。

对于上述步骤，如果再思考一下，就会发现所采用的方法并不是一次并行执行所有服务，而是每次只执行一个。方式和使用图形用户界面（GUI）重新配置服务并无不同。唯一的区别是代码使得计算机每次只配置一个服务，而不是人为操作。

计算机擅长执行重复操作，所以上面的过程所使用的方法是可取的。但问题在于该方法需要我们给计算机提供更长、更复杂的指令。学习使用该语言编写这些指令集需要花费时间，这也是管理员会尝试避免 VBScript 和其他脚本语言的原因。

PowerShell 可以使该方法重复，因为有些时候你还是需要上述方法，我们将会在本章后面展示如何做。但利用计算机枚举对象的方式并不是使用 PowerShell 最高效的方式。实际上，PowerShell 提供了其他两种更加易于学习和减少输入的方式，并且功能更加强大。

16.2 首选方法："批处理" Cmdlet

正如我们在之前章节所说，很多 PowerShell Cmdlet 可以接受批量对象，或者称之为对象集合。比如在第 6 章，你已经学习过利用管道将一个 Cmdlet 产生的结果传输给另一个 Cmdlet，比如说下面命令（请不要运行该命令——它将使你的计算机崩溃）。

```
Get-Service | Stop-Service
```

这是一个使用批处理管理的示例。在本例中，Stop-Service 专门被设计用于从管道接受一个或多个服务对象，并停止服务。Set-Service、Stop-Service、Move-ADObject 以及 Move-Mailbox 都是接受一个或多个输入对象并执行其任务或行为的 Cmdlet 示例。你无须像我们在之前小结 VBScript 中那样使用循环结构手动枚举对象。PowerShell 知道如何使用更简单的语法规则处理批量对象。

这就是所谓的 "batch Cmdlets"（这是我们对它的命名，并不是官方术语），也是我们批量管理的首选方式。比如说，我们希望改变 3 个服务的启动模式。我们不选择 VBScript 方式的方法，而是采用下面这种。

```
Get-Service -name BITS,Spooler,W32Time | Set-Service -startuptype Automatic
```

从某种程度来说，Get-Service 也是一种批处理 Cmdlet。这是由于该命令能够从多台计算机中获取服务。假设你需要变更同样这 3 台计算机上的服务。

```
Get-Service -name BITS,Spooler,W32Time -computer Server1,Server2,Server3 |
➥Set-Service -startuptype Automatic
```

上述方法中一个潜在的问题在于，执行动作的 Cmdlet 通常不会返回表示作业状态的结果。这意味着上面两个命令都不会产生可视化结果，这非常令人不安。值得庆幸的

是，这些命令通常会有一个-passThru 参数，该参数用于打印出该命令所接受的对象。你也可以使用 Set-Service 输出其修改的服务，并使用 Get-Service 重新获取这些服务以便查看之前的命令是否生效。

下面是不同 Cmdlet 使用-PassThru 参数的示例。

```
Get-Service -name BITS -computer Server1,Server2,Server3 |
➥Start-Service -passthru |
➥Out-File NewServiceStatus.txt
```

该命令将会从 3 台计算机列表中获取指定的服务，然后通过管道将这些服务传递给 Start-Service。该命令不仅会启动服务，而且会将涉及的服务对象打印在屏幕上。然后这些服务对象将会通过管道传递给 Out-File，将这些被更新对象的信息存储在文本文件中。

再重申一次：这是我们使用 PowerShell 推荐的首选方式。如果存在可以通过 Cmdlet 完成的工作，请使用 Cmdlet。理想情况下，Cmdlet 的作者都会选择以对象批处理的方式处理对象，但并不总是这样（Cmdlet 作者也在学习为我们这些管理员写 Cmdlet 的最佳方式）。这是最理想的方式。

16.3 CIM/WMI 方式：调用方法

不幸的是，总有一些任务无法通过调用 Cmdlet 完成。而且有一些我们可以通过 Windows 管理规范（WMI）可以操控的条目（关于 WMI，我们将会在第 14 章讲解）。

注意： 我们将通过故事线的方式帮助你体验人们如何使用 PowerShell。这会看起来有点多余，但请记住，经验本身是无价的。

比如，WMI 中的 Win32_NetworkAdapterConfiguration 类。该类代表与网卡绑定的配置信息（网卡可以有多个配置，但目前我们假设它只有一个配置信息，这也是对于大多数计算机的常见配置）。假如说我们的目标是在计算机上所有的 Intel 网卡上启用 DHCP，但我们不希望启用 RAS 或其他虚拟网卡的 DHCP。

我们可以以查询网卡配置开始，得到如下输出结果。

```
DHCPEnabled      : False
IPAddress        : {192.168.10.10, fe80::ec31:bd61:d42b:66f}
DefaultIPGateway :
DNSDomain        :
ServiceName      : E1G60
Description      : Intel(R) PRO/1000 MT Network Connection
Index            : 7
DHCPEnabled      : True
IPAddress        :
DefaultIPGateway :
```

```
DNSDomain          :
ServiceName        : E1G60
Description        : Intel(R) PRO/1000 MT Network Connection
Index              : 12
```

为了得到上述输出结果，我们需要查询合适的 WMI 类并过滤出只有描述中包含 INTEL 的配置。下面的代码可以完成该功能（注意在 WMI 过滤中以“%”作为通配符）。

```
PS C:\> gwmi win32_networkadapterconfiguration
➥-filter "description like '%intel%'"
```

动手实验：我们欢迎你跟随本章的示例执行代码。你或许需要小幅修改命令，从而获得希望的结果。比如说，你的计算机中并没有使用 Intel 制造的网卡，则需要将过滤条件做适当的修改。

我们在管道中包含这些配置对象信息后，我们希望启用 DHCP（你可以看到其中一块网卡并没有启用 DHCP）。我们或许可以找一个名称类似“Enable-DHCP”的 Cmdlet。不幸的是，我们找不到该 Cmdlet，因此不存在该 Cmdlet。没有任何 Cmdlet 可以直接在批处理中与 WMI 对象打交道。

下一步是查看对象本身是否包含可以启用 DHCP 的方法，为了找出结果，我们将配置对象通过管道传输给 Get-Member（或者其别名 Gm）。

```
PS C:\> gwmi win32_networkadapterconfiguration
➥-filter "description like '%intel%'" | gm
```

在结果列表的开始部分，我们可以看到我们寻找的方法 EnableDHCP()。

```
TypeName: System.Management.ManagementObject#root\cimv2\Win32_NetworkAd
apterConfiguration

Name                        MemberType      Definition
----                        ----------      ----------
DisableIPSec                Method          System.Management.ManagementB...
EnableDHCP                  Method          System.Management.ManagementB...
EnableIPSec                 Method          System.Management.ManagementB...
EnableStatic                Method          System.Management.ManagementB...
```

下一步，也是很多 PowerShell 新手会尝试的方法，将配置对象通过管道传递给该方法。

```
PS C:\> gwmi win32_networkadapterconfiguration
➥-filter "description like '%intel%'" | EnableDHCP()
```

不幸的是，这是无效的。你不能将对象通过管道传输给方法，你只能将其传递给 Cmdlet。EnableDHCP 并不是一个 PowerShell 的 Cmdlet，而是直接附加在配置对象自身的行为。这种传统的、类似 VBScript 的方法和我们在本章开篇所展示给你的 VBScript 示例非常类似。但使用 PowerShell，你能够以更简单的方式改成该任务。

虽然没有名为 Enable-DHCP 的“批处理”Cmdlet，但可以使用 Invoke-WmiMethod 这个通用 Cmdlet。该 Cmdlet 特别设计用于接受一批 WMI 对象，比如说我们的 Win32_NetworkAdapterConfiguration 对象，并调用附加在这些对象上的某个方法。下面是我们运行的命令。

```
PS C:\> gwmi win32_networkadapterconfiguration
➥-filter "description like '%intel%'" |
➥Invoke-WmiMethod -name EnableDHCP
```

你需要记住如下几条。

- 方法名称后无须加括号。
- 方法名称不区分大小写。
- Invoke-WmiMethod 一次只能接收一种类型的 WMI 对象。在本例中，我们只发送给 Win32_NetworkAdapterConfiguration 一种对象，这意味着命令可以如预期产生效果。当然也可以一次发送多个对象（实际上，这是重点），但所有的对象都必须是同一类型。
- 你可以针对 Invoke-WmiMethod 方法加上-WhatIf 和-Conifrm 参数。但直接由对象调用方法时，无法使用这些参数。

Invoke-WmiMethod 的输出结果有点让人困惑。WMI 总是产生结果对象，并包含大量系统对象（名称以两个下划线开始）。在本例中，命令产生如下输出结果。

```
__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 84
```

上述结果唯一有用的信息是一个没有以双下划线开头的属性：ReturnValue。该数字告诉我们操作的结果。通过 Google 搜索“Win32_NetworkAdapterConfiguration”出现文档页，我们通过单击 EnableDHCP 方法找到可能返回的值以及其代表的意义。图 16.1 展示了我们发现的结果。

0 表示成功，而 84 表示该网卡配置中未启用 IP，因此 DHCP 无法启用。但该值对应哪一个网卡配置呢？这很难说。这是由于输出结果并没有告诉你是由哪一个配置对象产生的。虽然令人遗憾，但这就是 WMI 的工作机制。

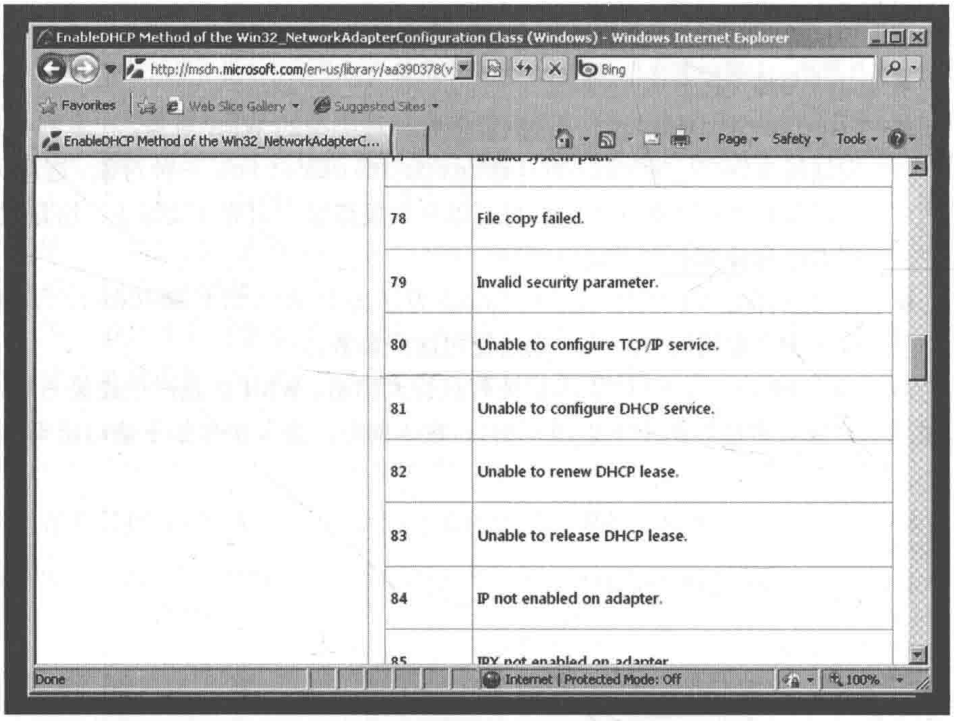

图 16.1 找 WMI 方法返回值的结果

当你有一个 WMI 对象包含可执行的方法时，大多可以使用 Invoke-WmiMethod。该命令对于远程计算机同样有效。我们的基本原则是“如果你可以使用 Get-WmiObject 获取对象，则也能够使用 Invoke-WmiObject 执行它的方法”。

当你回忆第 14 章所学内容时，你会发现 Get-WmiObject 与 Invoke-WmiMethod 都是“遗留”用于操作 WMI 的 Cmdlet；这两个命令的接替者为 Get-CimInstanc 和 Invoke-CimMethod。它们的工作方式或多或少有些相同。

```
PS C:\> Get-CimInstance -classname win32_networkadapterconfiguration
➥ -filter "description like '%intel%'" |
➥ Invoke-CimMethod -methodname EnableDHCP
```

在第 14 章中，我们提供了何时使用 WMI 或 CIM 的建议，该建议在此同样适用：虽然 WMI 所需的 RPC 网络通信难以穿透防火墙，但 WMI 能够适用的计算机数量最多（当前来说）；CIM 只需要更新更简单的 WS-MAN 通信，但在老版本的 Windows 默认情况下，并没有安装 WS-MAN。

但请等一下，还有一件事，我们在本小节讨论了 WMI，并在第 14 章中提到微软做了很多工作，也就是将 WMI 功能封装进了 Cmdlet，以至于无意中对你隐藏了 WMI 的存在（技术角度来讲，是 CIM 功能，但也很接近）。请尝试在 PowerShell 中运行 Help Set-NetIPAddress。在较新版本的 Windows 中，你将会发现这个强大的 Cmdlet 掩盖了大量底层 WMI/CIM 的复杂性。我们可以使用该 Cmdlet 变更 IP 地址，而无需一大堆 WMI/CIM。这是一个真实的教训：即使你在网上阅读了关于该主题的一些资料，也并不意味着新版本的 PowerShell 没有提供更好的方式。大多数发布在网上的资料都是基于 PowerShell v1 和 v2，但 v3 和更新的版本中提供的 Cmdlet 至少比之前的好 4～5 倍。

16.4 后备计划：枚举对象

不幸的是，我们遇到的一些情况是 Invoke-WmiObject 无法执行某个方法——执行时不断返回奇怪的错误信息（Invoke-CimMethod 更可靠）。我们还遇到的一些情况是虽然某个 Cmdlet 可以产生对象，但我们知道并没有可以通过管道接收这些对象并进行操作的批处理 Cmdlet。无论是上述哪种情况，你依然可以完成任务，但你必须回到传统的 VBScript 风格的方法来指挥计算机枚举对象并一次执行一个对象。PowerShell 提供了两种方法：第一种是使用 Cmdlet，另一种是使用脚本结构。我们在本章主要关注第一种技术，并在第 21 章阐述第二种。在第 21 章中，我们将会深入 PowerShell 内置的脚本语言。

我们使用 Win32_Service 这个 WMI 类作为示例。更详细地说，我们将使用 Change() 方法。这是一个可以一次性变更某个服务中多个元素的复杂方法。图 16.2 展示了其在线文档（通过搜索“Win32_Service”并单击 Change 方法找到）。

通过阅读该页，你会发现无须为该方法的每一个参数赋值。你可以将你希望忽略的参数指定为 Null（PowerShell 中有一个特殊的内置$null 变量）。

对于本例来说，我们希望变更服务的启动密码，也就是第 8 个参数。为了完成该工作，我们需要将前 7 个参数指定为$null。这意味着我们的方法执行代码看上去像下面这样。

```
Change($null, $null, $null, $null, $null, $null, $null, "P@ssw0rd")
```

顺便提一下，无论是 Get-Service 还是 Set-Service，都无法显示或设置某个服务的登录密码。但 WMI 可以完成该工作，所以我们使用 WMI。

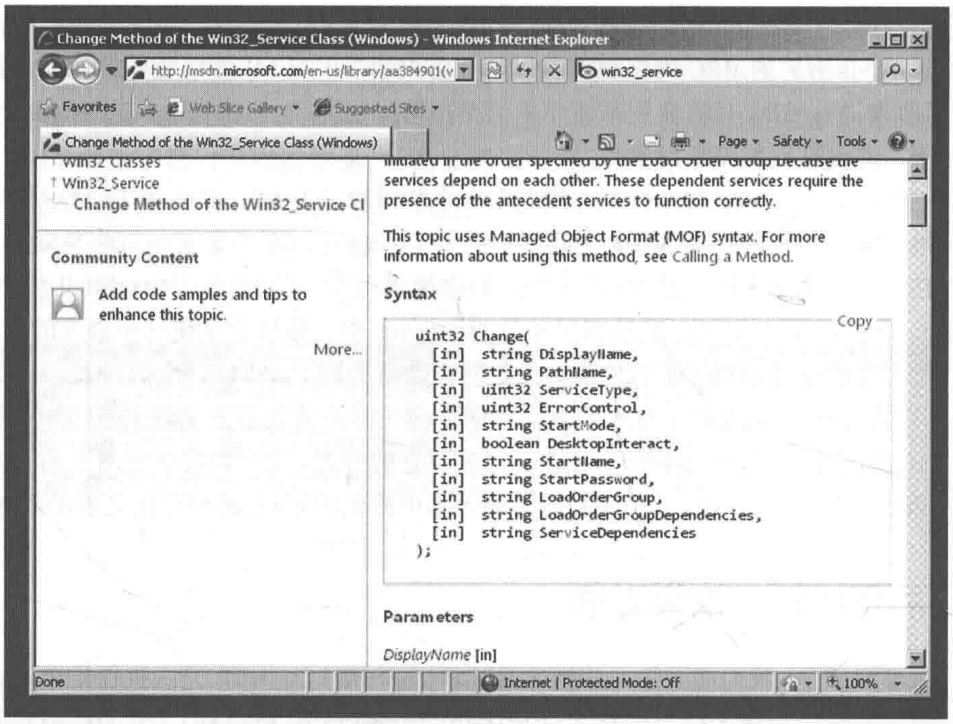

图 16.2 Win32_Service 的 Change() 方法的文档页

由于我们无法使用首选的 Set-Service 这个批处理 Cmdlet，让我们尝试第二种方式，也就是使用 Invoke-WmiMethod。该 Cmdlet 包含一个参数：-ArgumentList，可以利用该参数为方法指定参数。下面的示例是我们进行的尝试以及接收的结果。

```
PS C:\> gwmi win32_service -filter "name = 'BITS'" | invoke-wmimethod -name
➥ change -arg $null,$null,$null,$null,$null,$null,$null,"P@ssw0rd"
Invoke-WmiMethod : Input string was not in a correct format.
At line:1 char:62
+ gwmi win32_service -filter "name = 'BITS'" | invoke-wmimethod <<<<  -nam
e change -arg $null,$null,$null,$null,$null,$null,$null,"P@ssw0rd"
    + CategoryInfo          : NotSpecified: (:) [Invoke-WmiMethod], Forma
   tException
    + FullyQualifiedErrorId : System.FormatException,Microsoft.PowerShell
   .Commands.InvokeWmiMethod
```

注意：我们这里使用的是 Get-WmiObject，但 Get-CimInstance 的语法与其几乎相同。

此时，我们必须做出决定。有可能我们没有用正确的方式运行命令，所以我们必须决定是否花一些时间找出原因。还有一种可能是 Invoke-WmiMethod 与 Chang() 方法的兼容性存在问题。如果是这个问题的话，就需要我们花费大量时间在我们无法控制的事情上。

对于这种情况，我们通常会尝试其他方式：我们将会要求计算机（好吧，是 Shell）枚举所有服务对象，每次一个，并对每个对象执行 Change() 方法。我们将使用 ForEach-Object 这个 Cmdlet 完成这项工作。

```
PS C:\> gwmi win32_service -filter "name = 'BITS'" | foreach-object {$_.cha
➥nge($null,$null,$null,$null,$null,$null,$null,"P@ssw0rd") }
__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
```

在文档中，我们发现 ReturnValue 为 0 表示成功。这意味着我们已经实现了目标。但让我们将命令格式化得更美观，更仔细地看这个命令。

```
Get-WmiObject Win32_Service -filter "name = 'BITS'" |
➥ForEach-Object -process {
➥ $_.change($null,$null,$null,$null,$null,$null,$null,"P@ssw0rd")
➥}
```

该命令中包含很多内容。第一行看起来很合理：我们使用 Get-WmiObject 获取所有满足过滤条件的 Win32_Service 实例，也就是名称包含“BITS”的服务（照例，我们选择 BITS 服务是由于相比其他服务来说，该服务并没有那么重要，该服务停止运行不会导致计算机崩溃）。然后我们将 Win32_Service 对象传递给 ForEach-Object 这个 Cmdlet。

让我们把之前示例中的代码分解为模块。

- 首先，你将看到 Cmdlet 名称：ForEach-Object。
- 接下来，使用-Process 参数指定脚本段。我们原先并没有输入-Process 的参数名称，这是由于该参数为位置参数。但脚本段中，所有在花括号中的代码都是-Process 参数的值。所以我们接下来将参数名称包含在内，并更好地格式化，以方便阅读。
- ForEach-Object 将会对于每一个通过管道传输给 ForEach-Object 的对象执行脚本段。每次脚本段执行后，下一个通过管道传输进来的对象都会被置于特殊的$_容器。
- 通过在$_后输入一个“.”，告诉 Shell 我们需要访问当前对象的属性或方法。

- 在示例中，我们访问 Change() 方法。注意，方法的参数以逗号分隔列表方式存在，并被包在括号内。我们使用$null 作为我们不希望变更的参数传入，并将新密码作为第 8 个参数。该方法可以接受更多参数，但由于我们不希望修改第 9 个、第 10 个或第 11 个参数，我们可以完全忽视它。（我们也可以将最后三个参数指定为$null。）

我们当然传达了一个复杂的语法。图 16.3 将帮助你分解它。

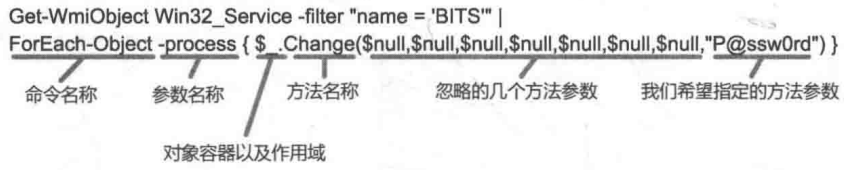

图 16.3 分解 ForEach-Object Cmdlet

你可以对 WMI 方法使用完全同样的模式。为什么你从不使用 Invoke-WmiMethod 来替代上面的方法呢？好吧，该命令通常会起作用，并更容易输入和阅读。但如果你更倾向于只记住一种方式，那就是 ForEach-Object 方式。

我们不得不警告你，在网上看到的示例可能或更难以阅读。PowerShell 专家更倾向于使用别名、位置参数以及最短的参数名称，这会降低可读性（但节省输入）。下面是同样的命令，但以最短的形式。

```
PS C:\> gwmi win32_service -fi "name = 'BITS'" |
➥ % {$_.change($null,$null,$null,$null,$null,$null,$null,"P@ssw0rd") }
```

让我们查看一下我们所做的变更。

- 我们使用 Gwmi 而不是 Get-WmiObject。
- 我们将-filter 简写为-fi。
- 我们使用%这个别名代替 ForEach-Object。是的，百分号符号是该 Cmdlet 的别名。我们发现该别名难以阅读，但很多人这么用。
- 我们再次删除了-Process 的参数名称，这是由于该参数是位置参数。

在博客或其他地方分享脚本时，我们并不喜欢使用别名和简写的参数名称。这是由于该方法使得其他人难以阅读。如果你将一些代码存入脚本，花费一些时间将代码输入完整是值得的（或者使用 Tab 自动补全功能让 Shell 帮你输入）。

如果你希望使用本例，下面是一些你希望改变的地方（见图 16.4）。

- 你或许希望改变 WMI 名称或者过滤条件，以取得你希望取得的 WMI 对象。
- 你可以将方法名称从 Change 修改为你希望执行的方法名称。
- 你可以修改方法的参数（也被称为“argument”）列表为任何你的方法期望的参数列表。参数列表总是一个逗号分隔的列表，并包裹在圆括号内。对于没有任何参数的方法，圆括号内可以为空，比如我们在本章开篇介绍的 EnableDHCP() 方法。

这是否是实现我们目标的最佳方式？通过查看 Set-Service 的帮助文档，我们发现该命令并没有提供修改密码的方式，而 Get-WmiObject 和 Get-CimInstance 这两个命令都可以完成该功能。这使得我们可以做出总结：即使是 PowerShell v3，对于这个任务，WMI 依然是一种值得使用的方式。

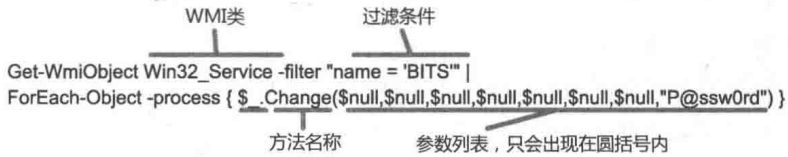

图 16.4 可以对之前示例所做的变更，以便执行不同的 WMI 方法

16.5 常见误区

我们本章中所涵盖的技术是 PowerShell 中最难的技术，这些技术是在我们班级中导致最多困惑的技术。让我们来看一些学生们经常遇到的问题，并提供一些替代的阐述方式。我们希望能够帮助你避免同样的问题。

16.5.1 哪一种是正确的方式

我们使用术语“批处理 Cmdlet”或“行为 Cmdlet”指代一个针对一组对象或对象集合操作的 Cmdlet。你可以将一组对象发送给 Cmdlet 并由 Cmdlet 对循环进行处理，而不是指示计算机“遍历列表中的东西，并对列表中的每一个东西执行某些行为”。

微软在其产品中提供这类 Cmdlet 方面做得越来越好，但并没有 100%覆盖所有功能（很可能以后很多年也覆盖不了，这是由于存在大量复杂的微软产品）。但当存在一个我们所需的 Cmdlet 时，我们更倾向使用 Cmdlet。即便如此，其他 PowerShell 的开发人员根据他们先学到的和他们更容易记住的倾向于选择其他替代办法。下面所有的命令实现的功能完全相同。

```
Get-Service -name *B* | Stop-Service                          <-❶ 批处理 cmdle
Get-Service -name *B* | ForEach-Object { $_.Stop()}           <-❷ ForEach-Object
Get-WmiObject Win32_Service -filter "name LIKE '%B%'" |       <-❸ WMI
➥Invoke-WmiMethod -name StopService
Get-WmiObject Win32_Service -filter "name LIKE '%B%'" |       <-
➥ForEach-Object { $_.StopService()}                              ❹ WMI 和 ForEach-Object
Stop-Service -name *B*                                        <-❺ Stop-Service
```

让我们来看一下每种方式的工作机制。

- 第一种方式是使用批处理 Cmdlet❶。这里，我们使用 Get-Service 获取所有名称包含“B”的服务，并停止这些服务。

- 第二种方式类似。但使用 ForEach-Object 替代批处理 Cmdlet，并要求每个服务执行 Stop() 方法❷。
- 第三种技术是使用 WMI，而不是 Shell 的原生管理 Cmdlet❸。我们接收到需要的服务（也就是名称包含字母“B”的服务），并通过管道传递给 Invoke-WmiMethod。我们告诉该命令调用 StopService 方法，这是 WMI 服务对象使用的方法名称。
- 第四种方式是使用 ForEach-Object 而不是 Invoke-WmiMethod 实现完全相同的工作❹。这种方式结合了方式 2 和方式 3，并不是一种全新的方式。
- 第五种方式是直接使用 Stop-Service❺，但其-Name 参数(在 PowerShell v3)接受通配符。

其实还有第六种方式——使用 PowerShell 的脚本语言完成工作。你将会发现 PowerShell 中每一项工作都可以使用多种方式完成，且没有哪一种方式是错误的。某些方式比其他方式更易于学习、记忆以及重复，这也是为什么我们按照所做的顺序关注我们可以使用的技术。

我们的例子还阐述了使用原生 Cmdlet 和 WMI 的重要区别。

- 原生 Cmdlet 过滤条件通常使用“*”作为通配符，而 WMI 过滤使用百分比符号（%）——请不要将百分比符号和 ForEach-Object 别名搞混。这个百分比符号封装在 Get-WmiObject 的-filter 参数内，它并不是一个别名。
- 原生对象通常和 WMI 有同样的功能，但语法或许会有不同。在本例中，由 Get-Service 产生的 ServiceController 对象有 Stop() 方法；而我们通过 WMI 的 Win32_Service 类访问同样的对象时，方法名称变为 StopService()。
- 原生过滤通常使用原生的比较操作符，比如说-eq；WMI 使用类似编程语言风格的操作符，比如说=或者 Like。

我们该使用哪一种方式?这无所谓，因为并没有一种所谓“对”的方式。你甚至会根据环境以及 Shell 能够提供给你的功能混合使用这两种方式。

16.5.2 WMI 方法与 Cmdlet 对比

你何时该使用 WMI 方法或 Cmdlet 来完成一个任务呢?这个选择十分简单。

- 如果你通过 Get-WmiObject 获取对象，你将需要通过使用 WMI 方法来执行行为。你可以使用 Invoke-WmiMethod 或 ForEach-Object 方式执行方法。
- 如果你通过非 Get-WmiObject 的方式获取对象，你将需要对获取到的对象使用原生 Cmdlet。除非你获取到的对象只有方法而没有能够完成任务所需的 Cmdlet，你可能会使用 ForEach-Object 方式执行方法。

注意，到这里的最低标准是 ForEach-Object：它的语法或许是最难的，但你可以使用它完成几乎所有你需要完成的工作。

无论何时都无法将任何对象通过管道传递给一个方法。你只能利用管道将一个 Cmdlet 产生的对象传递给另一个 Cmdlet。如果完成任务所需的 Cmdlet 不存在，但存在这样的方法，那么你就可以将其通过管道传递给 ForEach-Object 并执行对象的方法。

例如，假设你通过 Get-Something 这个 Cmdlet 获取到对象，你希望删除这些对象，但不存在 Delete-Something 或 Remove-Something 这样的 Cmdlet。但该对象包含 Delete 方法，那么你就可以这么做。

```
Get-Something | ForEach-Object { $_.Delete() }
```

16.5.3 方法文档

请记住，通过管道将对象传递给 Get-Member，可以查看该对象包含的方法。我们在此使用 Get-Something 这个 Cmdlet 作为示例。

```
Get-Something | Get-Member
```

PowerShell 的内置帮助系统并未记录 WMI 方法的文档。你需要使用搜索引擎（通常搜索 WMI 类的名称）来找到 WMI 方法的指南和示例。你也无法在 PowerShell 内置的帮助系统中找到非 WMI 对象的文档。比如说，如果你获取一个服务对象的成员列表，你将会发现存在名称为 Stop 和 Start 的方法。

```
TypeName: System.ServiceProcess.ServiceController

Name                      MemberType     Definition
----                      ----------     ----------
Name                      AliasProperty   Name = ServiceName
RequiredServices          AliasProperty   RequiredServices = ServicesDepe...
Disposed                  Event           System.EventHandler Disposed(Sy...
Close                     Method          System.Void Close()
Continue                  Method          System.Void Continue()
CreateObjRef              Method          System.Runtime.Remoting.ObjRef ...
Dispose                   Method          System.Void Dispose()
Equals                    Method          bool Equals(System.Object obj)
ExecuteCommand            Method          System.Void ExecuteCommand(int ...
GetHashCode               Method          int GetHashCode()
GetLifetimeService        Method          System.Object GetLifetimeService()
GetType                   Method          type GetType()
InitializeLifetimeService Method          System.Object InitializeLifetim...
Pause                     Method          System.Void Pause()
Refresh                   Method          System.Void Refresh()
```

```
Start                Method          System.Void Start(), System.Voi...
Stop                 Method          System.Void Stop()
ToString             Method          string ToString()
WaitForStatus        Method          System.Void WaitForStatus(Syste...
```

如果希望找到该对象的文档，请重点关注 TypeName，在本例中也就是 System.Service Process.ServiceController。在搜索引擎中搜索完整的类型名称，你通常可以找到完整的官方开发文档，并可以根据文档找出你所需的特定方法的文档。

16.5.4 ForEach-Object 相关误区

ForEach-Object 这个 Cmdlet 的语法中包含大量标点符号，再加上方法自带的语法，会导致出现难以阅读的命令行。我们准备了一些小技巧帮你打破僵局。

- 多使用 ForEach-Object 的完整名称，而不是使用%或 ForEach 这样的别名。完整名称更易于阅读。如果你使用别人写的示例，请将别名替换为完整名称。
- 花括号内的代码段对于每一个通过管道传入的对象执行一次。
- 在代码段内，$_代表通过管道传入的对象之一。
- 使用$_本身控制所有通过管道传入的对象；使用$_后的加“.”控制单独的方法或属性。
- 即使方法不需要任何参数，方法名称之后也总是跟随圆括号。当需要参数时，通过逗号将参数分隔放在括号内。

16.6 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 的计算机。

尝试回答接下来的问题并完成指定任务。这是一个重要的实验，因为该实验需要利用你在之前章节所学的技巧。随着你读完本书剩下的内容，你还需要不断巩固这些技巧。

1. 哪一个 ServiceController 对象（由 Get-Service 产生）的方法将会暂停服务，而不是完全停止服务？

2. 哪一个 Process 对象的方法（由 Get-Process 产生）可以终止指定的进程？

3. 哪一个 WMI 对象 Win32_Process 的方法将会终结一个给定进程？

4. 写 4 个不同命令，利用该命令可以终结所有名称为“Notepad”的进程。在此假设多个进程以同样的进程名称运行。

5. 假设你有一个计算机名称的文本列表，但希望以大写的方式展示。该使用哪种 PowerShell 表达式。

16.7 动手实验答案

1. 找到类似如下的方法：get-service | Get-Member -MemberType Method，你应该能够找到 Pause() 方法。

2. 找到类似如下的方法：get-service | Get-Member -MemberType Method，你应该能够找到 Kill() 方法。你可以通过检查该进程对象类型对应的 MSDN 文档进行确认。当然你并不应该需要调用方法，这是由于已经存在一个名称为 Stop-Process 的 cmdlet，该 cmdlet 可以实现该功能。

3. 你可以在 MSDN 文档中搜索 Win32_Process 类。或者你可以使用 CIM 的 cmdlet，这是由于这些 cmdlet 可作用于 WMI 用于列出所有可能的方法。

```
Get-CimClass win32_process | select -ExpandProperty methods
```

无论是哪种方法，你应该都能看到 Terminate() 方法。

4.

```
get-process Notepad | stop-process
stop-process -name Notepad
get-process notepad | foreach {$_.Kill()}
Get-WmiObject win32_process -filter {name='notepad.exe'} |
Invoke-WmiMethod -Name Terminate
```

5.

```
Get-content computers.txt | foreach {$_.ToUpper()}
```