第7章　扩展命令

可扩展性是 PowerShell 的一个主要优势。随着微软对 PowerShell 的持续投入，它为 Exchange Server、SharePoint Server、System Center 系列、SQL Server 等产品开发了越来越多的命令。通常，当你安装这些产品的管理工具时，还会安装一个或多个 Windows PowerShell 扩展的图形化管理控制台。

7.1　如何让一个 Shell 完成所有事情

我们知道你可能熟悉图形化的微软管理控制台（MMC），这就是为什么我们将使用它作为例子讲述 PowerShell 是如何工作的。它们涉及的可扩展的工作原理是一样的，部分原因是 MMC 和 PowerShell 由同一个管理框架下的团队所研发。

当你打开一个新的空白 MMC 控制台，在很大程度上，它的功能是有限的。因为 MMC 的内置功能很少，所以它基本上做不了什么事情。如果想让它强大一些，你需要在文件菜单中使用添加/删除管理单元。在 MMC 中，一个管理单元就是一个工具，这类似于活动目录用户和计算机、DNS 管理、DHCP 管理等。你可以在 MMC 中添加你喜欢的管理单元，也可以保存生成控制台，这使得下次更加方便地重新打开同一套管理单元。

这些管理单元从何而来？一旦你安装了类似 Exchange Server、Forefront 或者 System Center 产品的相关管理工具，会在 MMC 的添加、删除管理单元的对话框里面列出这些产品的管理单元。大多数产品也安装自己的预配置 MMC 控制台文件，它什么也不做，只是加载了基本的 MMC 和预加载一个或两个管理单元。如果你不想，可以不必使用这些预配置控制台，因为你总能打开一个空白的 MMC 控制台，并加载你需要的管理单元。例如，预配置的 Exchange Server MMC 控制台不包括活动目录站点和服务的管理单元，但你可以很容易地创建一个 MMC 控制台，包括交易所，也是站点和服务。

PowerShell 的工作原理方式几乎与 MMC 完全一样。安装一个给定产品的管理工具

（安装管理工具的选项通常包含在产品的安装菜单中。如果你在 Windows 7 上安装类似 Exchange Server 的产品，它的安装只提供了该管理工具）。这样做会为你提供 PowerShell 的相关扩展，它甚至可能会创建该产品特定的 Shell 管理程序。

7.2 关于产品的“管理 Shell”

这些管理特定产品的 Shell 程序的来源很混乱。我们必须澄清：只有一个 Windows PowerShell。根本就没有分 Exchange PowerShell 和活动目录 PowerShell，只有一个 Shell。

以活动目录为例，在 Windows Server 2008 R2 域控制器的开始菜单、管理工具下，你会发现一个关于活动目录组件的 Windows PowerShell。如果在这一项单击右键，然后从上下文菜单中选择属性，第一眼就可以看到类似如下的目标域：

```
%windir%\system32\WindowsPowerShell\v1.0\powerShell.exe
➥-noexit -command import-module ActiveDirectory
```

该命令运行标准的 PowerShell.exe 应用程序，而且指定命令行参数运行特定命令：Import-Module ActiveDirectory。执行的效果是可以预加载活动目录。但是，我们没有理由会认为：为什么不能打开“正常”的 PowerShell 并运行相同的命令获得相同的功能。

你可以找到同样适用于几乎所有特定于产品的“管理 Shell”：Exchange、SharePoint 等。查看这些产品开始菜单快捷方式的属性，你会发现，它们都是打开标准的 PowerShell.exe，并以传递一个命令行参数的方式来添加一个模块、增加一个管理单元或者加载一个预配置控制台文件（该控制台文件是一个包含需要自动加载管理单元的简单列表）。

SQL Server 2008 和 SQL Server 2008 R2 却是例外。它们“产品特定”Shell 叫作 Sqlps。它是一个经过特殊编译专门运行 SQL Server 扩展的 PowerShell。通常称之为 mini-Shell。微软第一次在 SQL Server 尝试这种方法。但这种方法已经不流行了，并且微软不会再使用这种方法了：SQL Server 2012 使用的是 PowerShell。

你不只局限于使用预先设定的扩展。当你打开 Exchange 的管理 Shell 程序，你可以运行 Import-Module ActiveDirectory 并假设该活动目录模块已经存在于你的电脑，添加活动目录功能到 Shell 中。你也可以打开标准的 PowerShell 控制台并手动添加你想要的扩展。

正如这一节前面提到的，这是一个让人感到非常困惑的知识点，包括有些人认为多个版本的 PowerShell 不能交叉利用彼此的功能。Don（作者）甚至在他的博客（http://windowsitpro.com/go/DonJonesPowerShell）中进行了讨论。PowerShell 团队成员介入并支持他，所以，请相信我们：你可以在一个 Shell 中包含所有你想要的功能，而在开始菜单中，特定产品的快捷方式不会以任何方式限制或暗示你这些产品存在特殊版本的 PowerShell。

7.3 扩展：找到并添加插件

PowerShell 存在两种类型的扩展：模块和管理单元。首先讲述管理单元。

一个适合管理单元 PowerShell 的名字是 PSSnapin，用于区别这些来自管理单元的图形 MMS。PSSnapins 在 PowerShell v1 版本的时候就已经存在了。一个 PSSnapin 通常包含一个或多个 DLL 文件，同时包含配置设置的 XML 文件和帮助文档。PSSnapins 必须先安装和注册，然后 PowerShell 才能识别它的存在。

注意： PSSnapin 的概念逐步被微软移除了，将来可能会越来越少出现。在内部，微软的重点是提供扩展的模块。

你可以通过在 PowerShell 中运行 Get-PSSnapin -registered 命令获取到一个可用的管理单元列表。因为我在域控制机器上安装了 SQL Server 2008，所以执行命令返回的结果如下：

```
PS C:\> get-pssnapin -registered
Name        : SqlServerCmdletSnapin100
PSVersion   : 2.0
Description : This is a PowerShell snap-in that includes various SQL
              Server Cmdlets.
Name        : SqlServerProviderSnapin100
PSVersion   : 2.0
Description : SQL Server Provider
```

上面的信息说明我的机器上安装了两个可用的管理单元，但是并没有加载。你可以通过运行 Get-PSSnapin 命令来查看加载的列表。该列表包含所有的核心，自动加载的管理单元包含 PowerShell 中的本机功能。

通过运行 Add-PSSnapin 并指定管理单元名的方式来加载某一个管理单元：

```
PS C:\> add-pssnapin sqlserverCmdletsnapin100
```

类似常用的 PowerShell 命令，你不必担心大小写是否正确，Shell 是忽略大小写的。

当一个管理单元加载成功了，你可能想知道 Shell 到底增加了什么功能。PSSnapin 可以增加 Cmdlets 命令、提供 PSDrive，或者两者都增加。使用 Get-Command（或者别名：Gcm）命令找出增加的 Cmdlets 命令：

```
PS C:\> gcm -pssnapin sqlserverCmdletsnapin100

CommandType     Name                          Definition
-----------     ----                          ----------
Cmdlet          Invoke-PolicyEvaluation       Invoke-PolicyEvaluation...
Cmdlet          Invoke-Sqlcmd                 Invoke-Sqlcmd [[-Query]...
```

我们在这里必须指出，输出的结果中只包含了 SqlServerCmdletSnapin100 这个管理单元，并且只有两行记录。是的，这就是 SQL Server 在管理单元中增加的所有内容，而且只有一个可以执行 Transact-SQL(T-SQL)的命令。因为你可以通过 T-SQL 命令在 SQL Server 上实现几乎所有的操作，Invoke-Sqlcmd 这个 Cmdlet 命令同样可以完成所有的操作。

运行 Get-PSProvider 可以查看一个管理单元提供哪些新的 PSDrive，你不能在该 Cmdlet 命令指定某个管理单元，所以你必须熟悉哪些提供程序已经存在，并通过查看列表方式发现新增内容。下面是返回结果：

```
PS C:\> get-psprovider

Name             Capabilities             Drives
----             ------------             ------
WSMan            Credentials              {WSMan}
Alias            ShouldProcess            {Alias}
Environment      ShouldProcess            {Env}
FileSystem       Filter, ShouldProcess    {C, A, D}
Function         ShouldProcess            {Function}
Registry         ShouldProcess, Transa... {HKLM, HKCU}
Variable         ShouldProcess            {Variable}
Certificate      ShouldProcess            {cert}
```

看起来没有任何新增内容。我们并不感到惊讶，因为管理单元是通过 SqlServer CmdletSnapin100 名称来加载的。如果你回忆一下，我们的可用管理单元同样包含了 SqlServerProviderSnapin100，这意味着微软出于某些原因，把它的 Cmdlets 命令和 PSDrive 分开打包。让我们尝试添加第二个：

```
PS C:\> add-pssnapin sqlserverprovidersnapin100
PS C:\> get-psprovider

Name             Capabilities             Drives
----             ------------             ------
WSMan            Credentials              {WSMan}
Alias            ShouldProcess            {Alias}
Environment      ShouldProcess            {Env}
FileSystem       Filter, ShouldProcess    {C, A, D}
Function         ShouldProcess            {Function}
Registry         ShouldProcess, Transa... {HKLM, HKCU}
Variable         ShouldProcess            {Variable}
Certificate      ShouldProcess            {cert}
SqlServer        Credentials              {SQLSERVER}
```

回顾一下之前的输出结果，可以发现 SQL Server 驱动器已经被添加到我们的 Shell 当中，由 SQL Server 的 PSDrive 提供驱动。新增的该驱动意味着可以运行命令：cd SQL server 切换到 SQL Server 驱动器，接着可以开始探索数据库。

7.4 扩展：找到并添加模块

PowerShell 提供的第二种扩展方式叫作模块。模块被设计得更加独立，因此更加容

易分发，但是它的工作原理类似于 PSSnapins。但是，你需对它们有更多了解，这样才能找到和使用它们。

模块不需要复杂的注册。反而，PowerShell 会自动在一个特定的目录下查找模块。PSModulePath 这个环境变量定义了 PowerShell 期望存放模块的路径：

```
PS C:\> get-content env:psmodulepath
C:\Users\Administrator\Documents\WindowsPowerShell\Modules;C:\Windows
\system32\WindowsPowerShell\v1.0\Modules\
```

在前面的例子中可以发现，路径中包含了两个默认的位置：其中一个是存放系统模块的操作系统目录，另外一个是存放个人模块的文档目录。只要你知道一个模块的完整路径，你也可以从任何其他的位置添加模块。

注意： PSModulePath 并不能在 PowerShell 中修改，它是你操作系统环境变量的一部分。你可以在系统控制面板对它进行修改，或者通过组策略。

在 PowerShell 中，该路径很重要。如果你有位于其他位置的模块，你应该把模块所在的路径加入到 PSModulePath 这个环境变量中。图 7.1 展示了如何通过系统控制面板而不是 PowerShell 去修改该环境变量。

图 7.1 修改 Windows 下的 PSModulePath 环境变量

为什么 PSModulePath 这个环境变量的路径如此重要？因为通过它，PowerShell 可以自动加载位于你计算机上的所有模块。PowerShell 会自动发现这些模块。换句话说，它看起来好像是所有的模块都已被加载了。查看一个模块的帮助，会发现你不需要手动加载它。运行任何的命令，PowerShell 都会自动加载该命令相关的模块。PowerShell 的 Update-Help 命令同样使用 PSModulePath 发现存在的任何模块，然后针对每个模块搜索需要更新的帮助文档。

例如，运行 Get-Module | Remove-Module 移除所有加载的模块。接着运行下面的命令（你返回的结果可能会有细微的差异，这取决于你所使用的 Windows 版本）：

```
PS C:\> help *network*

Name                              Category   Module
----                              --------   ------
Get-BCNetworkConfiguration        Function   BranchCache
Get-DtcNetworkSetting             Function   MsDtc
Set-DtcNetworkSetting             Function   MsDtc
Get-SmbServerNetworkInterface     Function   SmbShare
Get-SmbClientNetworkInterface     Function   SmbShare
```

正如你所看到的，PowerShell 发现了几个命令名中包含“network”关键字的命令（在函数分类里）。即使你没有加载该模块，你也可以查看它们中任何一个的帮助信息：

```
PS C:\> help Get-SmbServerNetworkInterface

名称
   Get-SmbServerNetworkInterface
语法
   Get-SmbServerNetworkInterface [-CimSession <CimSession[]>]
    [-ThrottleLimit <int>] [-AsJob] [<CommonParameters>]
```

如果你想，你甚至可以运行该命令，PowerShell 确保会自动为你加载该模块。这个自动发现和自动加载的功能非常有用，甚至帮你发现和使用你在启动 Shell 时没有出现的命令。

提示： 你也可以使用 Get-Module 命令检索一个远程服务器的可用模块列表，还可以使用 Import-Module 加载一个远程模块到当前 PowerShell 会话。你将在第 13 章中的远程控制中学习到如何使用该功能。

即使在模块还没有显式地加载到内存的情况下，PowerShell 依然可以自动发现模块让 Shell 完成命令名称自动补全（在控制台使用 Tab 按钮，或者使用 ISE 的智能提醒）、显示帮助和运行命令。该特性使得保持 PSModulePath 环境变量的完整和最新很有必要。

如果一个模块不在被 PSModulePath 引用的任何一个目录下，你应该使用 Import-Module 命令并指定模块的完整路径，如 C:\MyPrograms\Something\MyModule。

如果在开始菜单有一个特定产品 Shell 的快捷方式，比如说 Share Point Server，而你却不知道该产品安装 PowerShell 模块的路径，打开快捷方式图标的属性，像本章之前教你的方法，在快捷方式的目标属性中会包含使用 Import-Module 命令需要的模块名和路径。

模块还可以添加 PSDrive。你必须使用在 PSSnapins 中相同的技巧来确定有哪些新的提供者：运行 Get-PSProvider 命令。

7.5 命令冲突和移除扩展

仔细看看我们为 SQL Server 和活动目录增加的命令。注意到什么特别的命令名了吗？大多数的 PowerShell 扩展（Exchange Server 是一个明显的例外）都在它们命令名的名词部分增加了一个短的前缀，如 Get-ADUser 和 Invoke-SqlCmd。这些前缀看起来有些多余，但是它们可以防止命令名的冲突。

例如，假设你加载的两个模块中都包含了 Get-User 这个 Cmdlet 命令。这样两个命令名称相同，且被同时加载。你运行 Get-User 时，PowerShell 应该执行哪个呢？事实上是执行最后一个加载模块的命令。但是另外一个相同的命令却无法被访问。为了明确所需运行的具体命令，你需要使用看起来有点多余的命名规则，它包括管理单元名称和命令名称。如果 Get-User 来自一个叫作 yCoolPowerShellSnapin 的模块单元，你需要使用下面的方式运行：

```
MyCoolPowerShellSnapin\Get-User
```

这需要输入很多内容，这就是为什么微软建议添加特定产品前缀，如在每个命令的名词中加入 AD 或者 SQL。增加前缀可以防止冲突，并且使命令更容易识别和使用。

如果你已经对冲突不厌其烦，你可以随时选择删除冲突的扩展名。你需要运行 Remove-PSSnapin 或 Remove-Module，并指定管理模块或模块命令的名称，从而卸载某个扩展。

7.6 玩转一个新的模块

让我们开始对刚刚学习到的新知识加以实践。假设你使用最新版本的 Windows 系统，并希望你能跟随我们目前在本节的命令。更重要的是，我们希望你跟随该过程并思考我们将要解释的内容，因为这是我们教自己如何使用遇到的新命令而没有冲出去为每个单独的产品和功能买一本新书的方法。在本章的后面的动手实验，我们会让你自己重复该过程来学习一个全新的命令集。

我们的目标是清除我们计算机上的 DNS 名称解析缓存。我们还不知道 PowerShell 是否能做到这一点，所以我们先要在帮助系统中寻找一些线索：

```
PS C:\> help *dns*

Name                                Category     Module
----                                --------     ------
dnsn                                Alias
Resolve-DnsName                     Cmdlet       DnsClient
Clear-DnsClientCache                Function     DnsClient
Get-DnsClient                       Function     DnsClient
Get-DnsClientCache                  Function     DnsClient
Get-DnsClientGlobalSetting          Function     DnsClient
Get-DnsClientServerAddress          Function     DnsClient
Register-DnsClient                  Function     DnsClient
Set-DnsClient                       Function     DnsClient
Set-DnsClientGlobalSetting          Function     DnsClient
Set-DnsClientServerAddress          Function     DnsClient
Add-DnsClientNrptRule               Function     DnsClient
Get-DnsClientNrptPolicy             Function     DnsClient
Get-DnsClientNrptGlobal             Function     DnsClient
Get-DnsClientNrptRule               Function     DnsClient
Remove-DnsClientNrptRule            Function     DnsClient
Set-DnsClientNrptGlobal             Function     DnsClient
Set-DnsClientNrptRule               Function     DnsClient
```

是的！正如你看到的，这就是我们计算机上所有的 DnsClient 模块。前面的列表中显示了 Clear-DnsClientCache 命令，但是我们好奇哪个命令可用。为了找出该命令，我们手动加载该模块并列出所有命令。

动手实验：继续跟随我们运行这些命令。如果在你计算机上没有 DnsClient 这个模块，那是因为你使用了一个较旧的 Windows 版本。请考虑获取一个新的版本，甚至是在你的虚拟机里运行一个实验版本，直到可以运行下述命令：

```
PS C:\> import-module -Name DnsClient
PS C:\> get-command -Module DnsClient

Capability      Name
----------      ----
CIM             Add-DnsClientNrptRule
CIM             Clear-DnsClientCache
CIM             Get-DnsClient
CIM             Get-DnsClientCache
CIM             Get-DnsClientGlobalSetting
CIM             Get-DnsClientNrptGlobal
CIM             Get-DnsClientNrptPolicy
CIM             Get-DnsClientNrptRule
```

```
CIM             Get-DnsClientServerAddress
CIM             Register-DnsClient
CIM             Remove-DnsClientNrptRule
CIM             Set-DnsClient
CIM             Set-DnsClientGlobalSetting
CIM             Set-DnsClientNrptGlobal
CIM             Set-DnsClientNrptRule
CIM             Set-DnsClientServerAddress
Cmdlet          Resolve-DnsName
```

注意：可以查看关于 Clear-DnsClientCache 的帮助，或者甚至直接运行命令。PowerShell 会在后台为我们加载 DnsClient 模块。因为我们正处于探索阶段，这种方法可以查看到该模块的完整命令列表。

该命令列表看起来跟我们之前的列表或多或少有些相似。好的，让我们来看看 Clear-DnsClientCache 命令：

```
PS C:\> help Clear-DnsClientCache

名称
    Clear-DnsClientCache
语法
    Clear-DnsClientCache [-CimSession <CimSession[]>] [-ThrottleLimit
    <int>] [-AsJob] [-WhatIf] [-Confirm] [<CommonParameters>]
```

看起来已经很明确了，我们没有发现任何强制参数。让我们尝试运行命令：

```
PS C:\> Clear-DnsClientCache
```

好的，通常来说，没有消息就是最好的消息。尽管如此，我们更愿意看到该命令到底做了什么事情。尝试使用下面的命令：

```
PS C:\> Clear-DnsClientCache -verbose
详细信息: The specified name resolution records cached on this machine will be removed.
Subsequent name resolutions may return up-to-date information.
```

这个-verbose 开关虽然不会输出所有命令，但是对所有的命令都有效。在该示例中，我们得到一个指示发生了什么事情的信息，这让我们知道这个命令已经成功运行了。

7.7 配置脚本：在启动 Shell 时预加载扩展

假设你已经打开了 PowerShell，并且你已经加载了几个你最为喜欢的管理单元和模块。如果你接受这种方式，你需要为每个管理单元或模块运行一个个的命令。如果你有几个需要装载的话，这会花费几分钟来输入命令。当你不想使用 Shell 而关

闭了它的窗口时，下次重新打开 Shell 窗口，之前加载的管理单元和模块都不复存在了，而你需要运行命令重新加载它们。这是件多么可怕的事情。肯定有一个更好的方式可以解决该问题。

我们给你介绍 3 种更好的方式。第一个涉及创建一个控制台文件。这只能记录已经加载的 PSSnapins，对已经加载的模块是不起作用的。首先加载所有你想要的管理单元，接着运行下面的命令：

```
Export-Console c:\myShell.psc
```

运行该命令，可以把你在 Shell 中加载的管理单元列表保存到一个很小的 XML 文件。

接下来，你希望在某些地方创建一个新的 PowerShell 快捷方式，快捷方式的目标应该是：

```
%windir%\system32\WindowsPowerShell\v1.0\powerShell.exe
➥-noexit -psconsolefile c:\myShell.psc
```

当你使用该快捷方式打开一个新的 PowerShell 窗口，这将加载控制台，并且该 Shell 会自动加载控制台文件里面列表中的所有管理单元。再次提醒，不能包括模块。如果同时存在管理单元和模块或者你只想加载其中某些模块，这种情况下你应该怎么做呢？

提示： 请记住，PowerShell 会自动加载 PSModulePath 环境变量的其中一个路径中的模块。如果你想预加载模块，你只需要考虑该模块是否存在 PSModulePath 环境变量的其中一个路径中。

答案就是使用配置脚本。我们在前面提到，将在本书的第 25 章进行详细的讨论。现在按照下面的步骤来学习如何使用它们：

1. 在你的文档目录创建一个名为 WindowsPowerShell（在文件夹名中不要包含空格）的新文件夹。

2. 在上面创建的文件夹中使用记事本创建一个名为 profile.ps1 的新文件。当你使用记事本保存该文件时，需要确保文件名使用引号括起来（“profile.ps1”）。使用引号是为了防止记事本在文件名加上.txt 的文件扩展名。如果加上了.txt 扩展名，这种方法就行不通了。

3. 在刚刚创建的文本文件输入 Add-PSSnapin 和 Import-Module 命令，以一行一个命令的格式来加载管理单元和模块。

4. 回到 PowerShell 中，你需要启用脚本的执行功能，这在默认情况下是禁用的。我们将会在第 17 章讨论这样操作带来的安全隐患，但是现在我们假设你是在一个单独的虚拟机或者是单独的测试机上做该操作的，这样安全性就不再是个问题了。在该脚本中，运行 Set-ExecutionPolicy RemoteSigned 命令。需要注意的是，该命令只有在你以管理员身份运行 Shell 的时候才会执行。也可以使用组策略对象（GPO）来覆盖该设置。如果是这样做，你会得到一个警告消息。

5. 假设到目前为止你没有收到任何的错误或者警告。关闭并重启 Shell，这将会自动加载 profile.ps1 文件，执行里面的命令，为你加载喜欢的管理单元和模块。

动手实验： 如果你没有找到一个喜欢的管理单元或模块，创建上面这个简单的配置文件将是一个很好的练习。如果实在不知道输入什么，可以在配置脚本输入 cd \，这样你每次打开 Shell 的时候就会跳转到系统盘的根目录。但不要在你生产环境中的机器上执行上面的操作，因为我们还没有解决所有的安全隐患。

7.8 常见误区

使用 PowerShell 的新手，当他们开始操作模块和管理单元时经常会做一件错误的事情：他们不阅读帮助文档。特别地，他们在查看帮助的时候不使用-example 或者-full 开关。

坦白说，查看内建的示例是学习使用一个命令最好的方式。是的，滚动数以百计的命令列表可能是有点吓人（如 Exchange Server，新增的命令大大超过了 400 个），但是通过在命令 Help 和 Get-Command 基础上加通配符应该可以更容易缩小列表的范围。因此，阅读帮助文档吧！

7.9 动手实验

注意： 在本实验中，你需要一个 Windows 7、Windows Server 2008 R2 或者是更高版本的操作系统来运行 PowerShell v3 甚至是更高的版本。

通常，我们假设在你的计算机或者虚拟机上的操作系统为最新版本（客户端或者服务器版本）来运行测试。

在本实验，你只有一个任务：运行网络故障诊断包。当你成功做到了，你需要寻找“实例 ID”敲入回车键，运行 Web 连接测试，并且从一个指定的页面中寻求帮助。使用 http://videotraining.interfacett.com 作为你的测试地址。我们希望你获取的返回信息是“没有发现问题”，这意味着你运行该检查成功了。

为了完成该任务，你需要找到一个可以获取到故障诊断包的命令，并且需要一个可以执行故障诊断包的命令。你还需要找到这些包所处的位置和它们的名字。你需要知道的所有内容都在 PowerShell 里，帮助系统将为你找到它们。

这是你得到的所有帮助！