第 26 章　使用他人的脚本

尽管我们希望你能从头开始编写一些自己的 PowerShell 命令脚本，但是我们也意识到，在编写过程中你会严重依赖于互联网上的一些示例。不管你是直接利用别人博客中的示例还是修改在在线脚本代码库——比如 PowerShell 代码库（http://PoshCode.org）中发现的脚本，其实能利用借鉴别人的 PowerShell 脚本也算作一项重要的核心技能。在本章中，我们会带领你学会通过该过程理解别人的脚本，并最终将脚本修改以适合我们的需要。

特别感谢：感谢提供本章脚本的 Christoph Tohermes 和 Kaia Taylor。我们特意让他们提供一些带有瑕疵的脚本，这些脚本与我们通常见到最佳实践中那些完美的脚本不一样。在某些情况下，我们甚至会故意将他们提供的脚本进行破坏，使得本章中的场景更真实。我们非常感激他们对该学习活动所做的贡献。

请注意，我们选择这些脚本主要是因为在这些脚本中，他们使用了一些在本书中并未涉及的高阶 PowerShell 功能。再次，我们需要说明，这就是真实的世界：你总是会碰到陌生的东西。本练习的一个目的是尽快知道某个脚本的功能，即便你并未学习过该脚本用到的所有技术。

26.1　脚本

代码清单 26.1 展示了名为 New-WebProject.ps1 的完整脚本。该脚本主要用于调用微软 IIS Cmdlet——该 Cmdlet 存在于已安装 Web 服务角色的 Windows Server 2008 R2 以及之后版本的操作系统上。

代码清单 26.1　New-WebObject.ps1

```
param(
```

```
  [parameter(Mandatory = $true)]
  [string] $Path,
  [parameter(Mandatory = $true)]
  [string] $Name
  )
$System = [Environment]::GetFolderPath("System")
$script:hostsPath = ([System.IO.Path]::Combine($System, "drivers\etc\"))
➥+"hosts"

function New-localWebsite([string] $sitePath, [string] $siteName)
{
 try
 {
  Import-Module WebAdministration
 }
 catch
 {
 Write-Host "IIS PowerShell module is not installed. Please install it
➥first, by adding the feature"
}
Write-Host "AppPool is created with name: " $siteName
New-WebAppPool -Name $siteName
Set-ItemProperty IIS:\AppPools\$Name managedRuntimeVersion v4.0
Write-Host
if(-not (Test-Path $sitePath))
{
  New-Item -ItemType Directory $sitePath
  }
  $header = "www."+$siteName+".local"
  $value = "127.0.0.1 " + $header
  New-Website -ApplicationPool $siteName -Name $siteName -Port 80
➥-PhysicalPath $sitePath -HostHeader ($header)
 Start-Website -Name $siteName
 if(-not (HostsFileContainsEntry($header)))
 {
  AddEntryToHosts -hostEntry $value
 }
}

function AddEntryToHosts([string] $hostEntry)
{
 try
 {
  $writer = New-Object System.IO.StreamWriter($hostsPath, $true)
  $writer.Write([Environment]::NewLine)
  $writer.Write($hostEntry)
 $writer.Dispose()
```

```
 }
 catch [System.Exception]
 {
  Write-Error "An Error occured while writing the hosts file"
 }
}
function HostsFileContainsEntry([string] $entry)
{
 try
 {
  $reader = New-Object System.IO.StreamReader($hostsPath + "hosts")
  while(-not($reader.EndOfStream))
  {
   $line = $reader.Readline()
   if($line.Contains($entry))
   {
    return $true
   }
  }
  return $false
 }
 catch [System.Exception]
 {
  Write-Error "An Error occured while reading the host file"
 }
}
```

第一部分是一个参数块，你已经在第 21 章中进行了对应的学习。

```
param(
  [parameter(Mandatory = $true)]
  [string] $Path,
  [parameter(Mandatory = $true)]
  [string] $Name
  )
```

该参数块看起来有点不同，它定义了一个-Path 和一个-Name 参数，并且这两个参数均为强制性参数。公平的是，当你运行该命令时，你需要这两个信息。

下一组的命令行看起来更加神秘。

```
$System = [Environment]::GetFolderPath("System")
$script:hostsPath = ([System.IO.Path]::Combine($System, "drivers\etc\"))
➥+"hosts"
```

它们看起来并不像在做任何危险的事——类似 GetFolderPath 语句并不会导致任何报警。要想知道它们到底实现了什么功能，那么就需要将它们放到 Shell 中去执行。

```
PS C:\> $system = [Environment]::GetFolderPath('System')
PS C:\> $system
C:\Windows\system32
PS C:\> $script:hostsPath = ([System.IO.Path]::Combine
($system,"drivers\etc\"))+"hosts"
PS C:\> $hostsPath
C:\Windows\system32\drivers\etc\hosts
PS C:\>
```

$script:hostsPath 代码创建了一个新的变量。这样除了**$system** 变量之外，又有了一个新的变量。这几行命令定义了一个文件夹路径以及文件路径。请记下这几个变量的值，这样在学习该脚本过程中可以随时参照。

该脚本的后面包含了 3 个函数：New-LocalWebsite，AddEntryToHosts 和 HostsFile ContainsEntry。一个函数类似于包含在一个脚本中的某部分脚本：每个函数都代表着可以被单独调用的已打包的脚本块。你可以看到，每个函数都会定义一个或多个输入参数，尽管在上面的 Param()块中并未看到。相反，它们采用了一种仅在函数中才合法的参数定义方法：在函数名称后面的括号中将参数罗列出来（和 Parameter()块一样）。其实，这也可算作一种快捷方式。

如果查看该脚本，你不会看到这些函数被脚本本身调用，因此如果照搬这些脚本，那么脚本根本无法运行。但是在函数 New-LocateWebSite 中，你可以看到用了函数 HostsFileContainsEntry。

```
if(-not (HostsFileContainsEntry($header)))
 {
  AddEntryToHosts -hostEntry $value
 }
```

同时，你也可以看到，函数 AddEntryToHoses 被该代码调用。该函数被嵌套在 IF 语句中。你可以在 PowerShell 中执行 Help *IF*来获取更多的帮助信息。

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

Name                                Category  Module

----                                --------  ------
diff                                Alias
New-ModuleManifest                  Cmdlet    Microsoft.PowerShell.Core
Test-ModuleManifest                 Cmdlet    Microsoft.PowerShell.Core
Get-AppxPackageManifest             Function  Appx
Get-PfxCertificate                  Cmdlet    Microsoft.PowerShell.S...
Export-Certificate                  Cmdlet    PKI
Export-PfxCertificate               Cmdlet    PKI
Get-Certificate                     Cmdlet    PKI
Get-CertificateNotificationTask     Cmdlet    PKI
Import-Certificate                  Cmdlet    PKI
```

```
Import-PfxCertificate                Cmdlet     PKI
New-CertificateNotificationTask      Cmdlet     PKI
New-SelfSignedCertificate            Cmdlet     PKI
Remove-CertificateNotification...    Cmdlet     PKI
Switch-Certificate                   Cmdlet     PKI
Test-Certificate                     Cmdlet     PKI
about_If                             HelpFile
```

HelpFile 通常罗列在最后，比如这里的 About-If。通过阅读该命令对应的结果集，你就可以看到 IF 语句的工作原理。在上面示例的上下文中，该语句会检查函数 HostsFileContainsEntry 返回的值是 True 还是 False；如果返回 False，就会调用函数 AddEntryToHosts。该语句暗示 New-LocalWebSite 函数才是脚本中“最主要”的函数，或者称之为期望被运行并触发某些变更的函数。HostsFileContainsEntry 和 AddEntryToHosts 函数看起来就像是函数 New-LocalWebSite 的功能函数——在需要时才会被调用。所以，此时我们需要关注 New-LocalWebSite 函数。

```
function New-localWebsite([string] $sitePath, [string] $siteName)
{
 try
 {
  Import-Module WebAdministration
 }
 catch
 {
  Write-Host "IIS PowerShell module is not installed. Please install it
➥first, by adding the feature"
 }
 Write-Host "AppPool is created with name: " $siteName
 New-WebAppPool -Name $siteName
 Set-ItemProperty IIS:\AppPools\$Name managedRuntimeVersion v4.0
 Write-Host
 if(-not (Test-Path $sitePath))
 {
  New-Item -ItemType Directory $sitePath
 }
 $header = "www."+$siteName+".local"
 $value = "127.0.0.1 " + $header
 New-Website -ApplicationPool $siteName -Name $siteName -Port 80
➥-PhysicalPath $sitePath -HostHeader ($header)
 Start-Website -Name $siteName
 if(-not (HostsFileContainsEntry($header)))
 {
  AddEntryToHosts -hostEntry $value
 }

}
```

你可能不太理解 Try 块。快速查找对应的帮助文档（Help *Try*）会显示 About_Try_Cacth_Finally 帮助文档，其中阐述到：Try 部分中的任何命令都有可能产生一个错误信息。如果确实产生了错误信息，那么就会执行 Catch 部分的命令。所以上面的命令可以解释为：该函数会尝试载入 WebAdministration 模块，如果载入失败，那么会显示一个错误信息。坦白讲，我们认为在发生错误时，应该完全退出该函数，但是在这里并非如此。所以当 WebAdministration 模块未成功载入时，你可以想象，这里会看到更多的错误信息。所以在执行该脚本之前，你必须保证 WebAdministration 模块可用！

Write-Host 块主要用作帮助追踪脚本运行进度。下一个命令是 New-WebAppPool。查看帮助文档，发现该命令包含在 WebAdministration 模块中，该命令的帮助文档阐述了其作用。接下来，Set-ItemProperty 命令看起来像是对刚建立的 AppPool 对象设置某些选项。

看起来这里简单的 Write-Host 命令，仅是为了在屏幕上放置一个空行。确实如此。如果你查看 Test-Path，你会发现它会检查一个给定的路径是否存在，在这个脚本中是指一个文件夹。如果不存在，那么脚本就会使用 New-Item 命令创建该文件夹。

变量$Header 在创建后被用作将$SiteName 转化为一个类似“www.sitename.local”的网址，同时$Value 变量用作添加一个 IP 地址。之后 New-WebSite 命令会在使用多个参数后被执行——你可以通过阅读该命令对应的帮助文档来查看各个参数的作用。

最后执行 Start-WebSite 命令。在帮助文档中有说明，该命令会启动对应的网站使其运行。此时就会调用 HostsFileContainsEntry 和 AddEntryToHosts 命令。它们会确保$Value 变量中的值对应的站点信息会以（IP 地址-名称）格式被添加到本地 Hosts 文件中。

26.2 逐行检查

在前面的小节中，我们采用的是逐行分析该脚本，这也是我建议你们采用的方式。当你逐行查阅每一行时：

- 识别其中的变量，并找出其对应的值，之后将它们写在一张纸上。因为大部分情况下，变量都会被传递给某些命令，所以记下每个变量可能的值会帮助你预测每个命令的作用。
- 当你遇到一些新的命令时，请阅读对应的帮助文档，这样可以理解这些命令的功能。针对 Get-类型的命令，尝试运行它们——将脚本中变量的值传递给命令的参数——来查看这些命令的输出结果。
- 当你遇到不熟悉的部分时，比如[Environment]，请考虑在虚拟机中执行简短的代码片段来查看该片段的功能（使用虚拟机有助于保护你的生产环境）。可以通过在帮助文档中搜寻（使用通配符）这些关键字来查阅更多的信息。

最重要的是，请不要跳过脚本中的任意一行。请不要抱有这种想法：“好吧，我不知道这一行命令的功能是什么，那么我就可以跳过它，继续看后面的命令。”请一定先

停下来，找出每一行命令的作用或者你认为它们可以实现的功能。这样才能保证你知道需要修改哪些部分的脚本来满足特定的需求。

26.3 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。

代码清单 26.2 呈现了一个完整的脚本。看看你是否能明白该脚本所实现的功能，以及实现的原理。你是否能找到该脚本中可能会出现的错误？需要如何修改该脚本才能使得可以在你的环境中运行？

请注意，你应该照搬该脚本，但是如果在你的系统中无法执行，你是否能够跟踪到问题所在？请记住，你应该见过该脚本里面的大部分命令，如果遇到没见过的命令，请查看 PowerShell 的帮助文档。帮助文档中的示例部分包含本脚本中用到的所有技术。

代码清单 26.2 Get-LastOn.ps1

```
function get-LastOn {
<#
.DESCRIPTION
Tell me the most recent event log entries for logon or logoff.
.BUGS
Blank 'computer' column

.EXAMPLE
get-LastOn -computername server1 | Sort-Object time -Descending |
Sort-Object id -unique | format-table -AutoSize -Wrap
ID                  Domain              Computer    Time
--                  ------              --------    ----
LOCAL SERVICE       NT AUTHORITY                    4/3/2012 11:16:39 AM
NETWORK SERVICE     NT AUTHORITY                    4/3/2012 11:16:39 AM
SYSTEM              NT AUTHORITY                    4/3/2012 11:16:02 AM

Sorting -unique will ensure only one line per user ID, the most recent.
Needs more testing

.EXAMPLE
PS C:\Users\administrator> get-LastOn -computername server1 -newest 10000
-maxIDs 10000 | Sort-Object time -Descending |

Sort-Object id -unique | format-table -AutoSize -Wrap

ID              Domain      Computer  Time
--              ------      --------  ----
Administrator   USS                   4/11/2012 10:44:57 PM
```

```
ANONYMOUS LOGON  NT AUTHORITY          4/3/2012 8:19:07 AM
LOCAL SERVICE    NT AUTHORITY          10/19/2011 10:17:22 AM
NETWORK SERVICE  NT AUTHORITY          4/4/2012 8:24:09 AM
Student          WIN7                  4/11/2012 4:16:55 PM
SYSTEM           NT AUTHORITY          10/18/2011 7:53:56 PM
USSDC$           USS                   4/11/2012 9:38:05 AM
WIN7$            USS                   10/19/2011 3:25:30 AM

PS C:\Users\administrator>

.EXAMPLE
get-LastOn -newest 1000 -maxIDs 20
Only examines the last 1000 lines of the event log

.EXAMPLE
get-LastOn -computername server1| Sort-Object time -Descending |
Sort-Object id -unique | format-table -AutoSize -Wrap
#>

param (
        [string]$ComputerName = 'localhost',
        [int]$Newest = 5000,
        [int]$maxIDs = 5,
        [int]$logonEventNum = 4624,
        [int]$logoffEventNum = 4647
      )

      $eventsAndIDs = Get-EventLog -LogName security -Newest $Newest |
      Where-Object {$_.instanceid -eq $logonEventNum -or
➥$_.instanceid -eq $logoffEventNum} |
      Select-Object -Last $maxIDs
➥-Property TimeGenerated,Message,ComputerName

      foreach ($event in $eventsAndIDs) {
        $id = ($event |
        parseEventLogMessage |
        where-Object {$_.fieldName -eq "Account Name"} |
        Select-Object -last 1).fieldValue

        $domain = ($event |
        parseEventLogMessage |
        where-Object {$_.fieldName -eq "Account Domain"} |
        Select-Object -last 1).fieldValue

        $props = @{'Time'=$event.TimeGenerated;
            'Computer'=$ComputerName;
```

```
                'ID'=$id
                'Domain'=$domain}

            $output_obj = New-Object -TypeName PSObject -Property $props
            write-output $output_obj
        }
}

function parseEventLogMessage()
{
    [CmdletBinding()]
    param (
        [parameter(ValueFromPipeline=$True,Mandatory=$True)]
        [string]$Message
)

$eachLineArray = $Message -split "`n"

foreach ($oneLine in $eachLineArray) {
    write-verbose "line:_$oneLine_"
    $fieldName,$fieldValue = $oneLine -split ":", 2
        try {
            $fieldName = $fieldName.trim()
            $fieldValue = $fieldValue.trim()
        }
        catch {
            $fieldName = ""
        }

        if ($fieldName -ne "" -and $fieldValue -ne "" )
        {
            $props = @{'fieldName'="$fieldName";
                    'fieldValue'=$fieldValue}

            $output_obj = New-Object -TypeName PSObject -Property $props
            Write-Output $output_obj
            }
    }
}
Get-LastOn
```