第 20 章　轻松实现远程控制

在第 13 章中，我们介绍了 PowerShell 的远程控制功能。其中，你使用了两个用于远程控制的 Cmdlet——Invoke-Command 和 Enter-PSSession——分别用于实现一对一以及一对多的远程控制。这两个命令的工作原理是创建一个远程连接，完成你指定的工作，然后关闭连接。

上面的方式并无不妥，但每次不断指定计算机名称、凭据、备用端口号等是一件非常麻烦的事情。在本章中，我们将发现实现远程控制更加简单、更可重用的方式。你还可以学到使用远程控制的第三种方式。

20.1　使得 PowerShell 远程控制更加容易

每次使用 Invoke-Command 或 Enter-PSSession 命令连接远程计算机时，你至少需要指定计算机名称（如果你需要在多台计算机上调用命令，则为多个名称）。根据具体环境的不同，你可能还需要指定备用凭据，这意味着需要提示你输入密码。你或许还需要指定备用端口或身份验证机制，这取决于你的组织如何配置远程控制。

上面的选项并不难，但不断重复输入却让人感到乏味。幸运的是，我们知道一种更好的方法：可重用会话。

20.2　创建并使用可重用会话

会话是一个在你的 PowerShell 副本与远程 PowerShell 副本之间的持久化连接。当一个会话处于活动状态时，你的计算机与远程计算机都会划分出一小部分用于维护连接的内存和处理器时间，还有非常少一部分与连接相关的网络流量。PowerShell 维护一个所有已打开的会话列表，你可以使用这些会话调用命令或进入远程 Shell。

你可以通过 New-PSSesion 这个 Cmdlet 创建一个新的会话，指定一个或多个计算机名称。如果需要，还可以指定备用用户名称、端口以及身份验证机制等。结果是一个存在 PowerShell 内存中的会话对象。

```
PS C:\> new-pssession -computername server-r2,server17,dc5
```

通过 Get-PSSession 获取创建好的会话。

```
PS C:\> get-pssession
```

虽然上面的方法可以奏效，但我们更倾向于创建 Session 后立刻将其存入变量。例如，Don 有 3 个基于 IIS 的 Web 服务器，它需要定期通过 Invode-Command 命令配置这些服务器。为了让过程变得简单，它将这些会话存入特定变量。

```
PS C:\> $iis_servers = new-pssession -comp web1,web2,web3
➥ -credential WebAdmin
```

请永远不要忘记这些会话会消耗资源。如果关闭 Shell，那么这些会话也会随之关闭，但如果你不是频繁使用这些会话，那么即使你希望使用同一个 Shell 完成其他任务，手动关闭这些会话也是不错的主意。

使用 Remove-PSSession 这个 Cmdlet 关闭会话。比如说，只关闭连接到 IIS 的会话，可以使用下面的命令。

```
PS C:\> $iis_servers | remove-pssession
```

或者，如果希望关闭所有处于开启状态的会话，使用下面的命令。

```
PS C:\> get-pssession | remove-pssession
```

就是这么简单。

一旦成功建立会话后，你该如何使用这些会话？在接下来几小节中，我们假设你已经创建了一个名称为$sessions 的变量，并至少包含两个会话。我们使用 localhost 和 Server-R2（你应该指定为符合你具体环境的计算机名称）。使用 Localhost 并不是一个语法糖：PowerShell 会开启一个真正指向本机 PowerShell 副本的远程会话。请记住，只有在所有连接到的计算机上都启用了远程控制时，远程连接才会生效。如果还未启用远程控制，请返回第 13 章。

动手实验：跟随上面的步骤并运行这些命令，确保使用有效的计算机名称。如果你只有一台计算机，请使用计算机名称和 localhost。

补充说明

有一个语法允许你使用一个命令创建多个会话，并将每个会话赋值给对应的变量（而不是像之前的示例，将其全部塞入一个变量）。

```
$s_server1,$s_server2 = new-pssession -computer server-r2,dc01
```

该语法将连接到 Server-R2 服务器的会话存入变量$s_server1，将连接到 DC01 服务器的会话存入$s_server2，这使得独立使用不同的会话变得简单。

但是请小心使用：我们曾见过会话的顺序和计算机名称的顺序不完全一致，导致$s_server1 最终包含连接到 DC01 而不是 Server-R2 的会话。你可以将变量内容显示出来，从而查看会话连接到哪一台计算机。

下面的代码用于建立好会话并使其运行。

```
PS C:\> $sessions = New-PSSession -comp SERVER-R2,localhost
```

请记住，我们已经在这些计算机上启用了远程控制，并且这些计算机处于同一个域。如果你希望回忆起如何启用远程控制，请再次查看第 13 章。

20.3 利用 Enter-PSSession 命令使用会话

回忆第 13 章，Enter-PSSession 命令是用于进入远程计算机一对一的交互式 Shell 所用的命令。该命令的参数可以是一个会话对象，而不是具体的计算机名称。由于$session 变量中包含两个会话对象，我们必须通过索引指定使用其中哪一个会话对象（你在第 18 章中学到过）。

```
PS C:\> enter-pssession -session $sessions [0]
[server-r2]: PS C:\Users\Administrator\Documents>
```

可以看到命令提示符已经改变，表示我们已经在控制远程计算机。Exit-PSSession 命令用于帮助我们返回到本地提示符，但远程会话并不会中断，以便于后续使用。

```
[server-r2]: PS C:\Users\Administrator\Documents>exit-pssession
PS C:\>
```

或许你很难记起具体哪一个索引号对应哪一个计算机名称。如果是这种情况，你可以利用会话对象的属性进行区分。例如，当我们将$sessions 对象通过管道传递给 Gm 命令时，我们可以得到如下输出结果。

```
PS C:\> $sessions | gm

   TypeName: System.Management.Automation.Runspaces.PSSession
Name                      MemberType     Definition
------                    ----------     ----------
Equals                    Method         bool Equals(System.Object obj)
GetHashCode               Method         int GetHashCode()
GetType                   Method         type GetType()
ToString                  Method         string ToString()
ApplicationPrivateData    Property       System.Management.Automation.PSPr...
Availability              Property       System.Management.Automation.Runs...
ComputerName              Property       System.String ComputerName {get;}
```

```
ConfigurationName      Property       System.String ConfigurationName {...
Id                     Property       System.Int32 Id {get;}
InstanceId             Property       System.Guid InstanceId {get;}
Name                   Property       System.String Name {get;set;}
Runspace               Property       System.Management.Automation.Runs...
State                  ScriptProperty System.Object State {get=$this.Ru...
```

在上面的输出结果中，你可以看到会话对象包含一个名为 ComputerName 的属性。这意味着你可以筛选出该会话。

```
PS C:\> enter-pssession -session ($sessions |
➥where { $_.computername -eq 'server-r2' })
[server-r2]: PS C:\Users\Administrator\Documents>
```

这个语法的处境比较尴尬，因为如果你需要使用变量中的一个会话，但记不住其中的会话索引号，或许你会更容易忘记使用变量。

即使你将会话对象存于变量中，这些会话依然被存于 PowerShell 的一个打开会话的主列表中。这意味着你可以通过 Get-PSSession 访问这些会话。

```
PS C:\> enter-pssession -session (get-pssession -computer server-r2)
```

Get-PSSession 将会获取名称为 Server-R2 的计算机，并将其传递给 Enter-PSSession 命令的-Session 参数。

当我们第一次发现这个技巧时，我们非常震惊，但这也让我们更进一步。我们找出 Enter-PSSession 的完整帮助并仔细阅读-Session 参数。下面是我们所看到的。

```
-Session <PSSession>
    Specifies a Windows PowerShell session (PSSession) to use for the
    interactive session. This parameter takes a session object. You can
    also use the Name, InstanceID, or ID parameters to specify a
    PSSession.
    Enter a variable that contains a session object or a command that
    creates or gets a session object, such as a New-PSSession or Get-
    PSSession command. You can also pipe a session object to Enter-
    PSSession. You can submit only one PSSession with this parameter. If
    you enter a variable that contains more than one PSSession, the
    command fails.
    When you use Exit-PSSession or the EXIT keyword, the interactive
    session ends, but the PSSession that you created remains open and a
    vailable for use.
    Required?                    false
    Position?                    1
    Default value
    Accept pipeline input?       true (ByValue, ByPropertyName)
    Accept wildcard characters?  True
```

如果你回想一下第 9 章的内容，你将会发现在帮助末尾的管道输入信息非常有趣。

该信息告诉我们，-Session 参数可以从管道接受一个 PSSession 对象。我们知道 Get-PSSession 命令会生成 PSSession 对象，所以下述语法也可以生效。

```
PS C:\> Get-PSSession -ComputerName SERVER-R2 | Enter-PSSession
[server-r2]: PS C:\Users\Administrator\Documents>
```

该命令的确可以生效。我们认为，就算你已经将所有会话存入一个对象中，使用该方式也是一种更加优雅的获取单个对象的方式。

提示：为了方便，将会话存入一个变量是可以的。但请记住，PowerShell 已经保存了所有已打开会话的列表；将这些会话存入变量，只有在你需要一次性引用多个会话时才有用，正如你将在下一小节所见的。

20.4 利用 Invoke-Command 命令使用会话

Invoke-Command 命令展示了 Session 对象的价值，你习惯于用该命令将一个命令（或一个完整的脚本）并行在多个远程计算机上执行。我们已经将所有的会话存储在 $Session 变量中，我们可以通过下面的命令轻松将多个计算机作为目标。

```
PS C:\> invoke-command -command { get-wmiobject -class win32_process }
➥ -session $sessions
```

注意，我们将一个 Get-WmiObject 命令发送到远程计算机。我们本可以选择使用 Get-WmiObject 命令自带的-computername 参数，但是由于下面 4 个原因，我们没有这么做。

- 远程控制通过一个预定义的端口进行传输，WMI 却不是。远程控制因此针对在防火墙后的计算机更加容易使用，这是由于更容易开启必要的防火墙例外。微软 Windows 防火墙包含必要的状态检测，为 WMI 随机端口选择提供特定的例外，从而使得 WMI 随机端口选择（也就是端点匹配）可以正常工作，但对于其他第三方防火墙产品来说却难以管理。通过远程控制就容易很多，因为只需要将一个端口设为例外。
- 将所有的进程传输到本地费时费力。使用 Invoke-Command 这个 Cmdlet，可以让每一台计算机完成各自的工作，并将结果返回。
- 远程控制并行执行，默认可以连接最多 32 台计算机。WMI 顺序执行，一次只能在一台计算机上执行。
- 我们无法通过 Get-WmiObject 使用我们预定义的会话对象，但可以通过 Invoke-Command 使用。

注意：在 PowerShell v3 中，新的 CIM Cmdlet（比如说 Get-CimInstance）并不像 Get-WmiObject 那样有一个-computerName 参数。新的 Cmdlet 被设计的本意就是，如果希望在远程计算机上执行，请通过 Invoke-Command 将其发送过去。

Invoke-Command 的-Session 参数也可以通过括号命令提供，正如我们在之前章节对计算机名称所做的那样。举例来说，下面的语句会将命令发送给计算机名称以“loc”开头的已连接会话。

```
PS C:\> invoke-command -command { get-wmiobject -class win32_process }
➥ -session (get-pssession -comp loc*)
```

你或许会期望 Invoke-Command 可以从管道中接收会话对象，就像 Enter-PSSession 命令那样。但通过查看 Invoke-Command 的完整帮助，会发现它并不支持这种使用管道的技巧。很不幸，但之前使用括号表达式的示例可以无需过于复杂的语法提供了同样的功能。

20.5 隐式远程控制：导入一个会话

隐式远程控制是对我们来说最酷、最有用的功能之一——可能是在任何操作系统的命令行界面中迄今为止最酷、最有用的功能。但不幸的是，该功能并未记入 PowerShell 文档。当然，那些必要的命令都有良好的文档，但这些必要命令共同汇集在一起形成的这个强大功能却没有在文档中被提及。所幸，我们在本文中对该功能进行了阐述。

让我们重新回顾一下场景：你已经知道微软针对 Windows 和其他产品发行越来越多的模块和插件，但由于各种各样的原因，你无法将这些模块安装在本地计算机上。在 Windows Server 2008 R2 上第一次发行的活动目录（ActiveDirectory）模块就是一个很好的示例：该模块只存在于 Windows Server 2008 R2 以及安装远程服务器管理工具（Remote Server Administration Tools，RSAT）的 Windows 7 上。如果计算机的操作系统是 Windows XP 或 Windows Vista 呢？是否就无法安装了？当然不是，你可以使用隐式远程控制。

让我们通过一个示例来查看完整的过程。

```
PS C:\> $session = new-pssession -comp server-r2        <-- 1 Establishes connection
PS C:\> invoke-command -command
➥{ import-module activedirectory }                     <-- 2 载入远程控制模块
➥-session $session
PS C:\> import-pssession -session $session
➥-module activedirectory                               <-- 3 Imports remote commands
➥-prefix rem
ModuleType Name                     ExportedCommands    <-- 4 查看临时本地模块
---------- ----                     ----------------
Script     tmp_2b9451dc-b973-495d... {Set-ADOrganizationalUnit, Get-ADD...
```

下面是本示例的解释。

① 首先，通过与一台装有活动目录模块的远程计算机建立一个会话。我们需要该计算机装有 PowerShell v2 或更新版本（在 Windows Server 2008 R2 以及更新版本的操作系统上），我们必须启用该计算机的远程控制。

② 我们告诉远程计算机导入其本地的活动目录模块。这只是一个示例。我们当然可以选择载入任意模块，甚至是在需要时添加一个 PSSnapin。由于会话处于打开状态，该模块将一直在远程计算机上处于被载入状态。

③ 我们接下来告诉我们的计算机从远程会话中导入命令。我们只需要在活动目录模块中的命令，并在每个命令的名词部分加入“rem”前缀。这使得我们可以更容易跟踪远程命令。这还意味着从远程会话导入的命令不会与已经在本地 Shell 中导入的命令冲突。

④ PowerShell 在本地计算机创建一个临时模块，用于代表远程命令。这些命令并不是被复制过来的；PowerShell 为其创建了指向远程计算机的快捷方式。

现在我们就可以运行活动目录模块的命令了，甚至是使用帮助命令。我们使用 `New-remADUser` 来代替 `New-ADUser`，这是由于我们在命令的名词部分添加了前缀“rem”。该命令在我们关闭 Shell 或关闭与远程连接的会话之前一直存在。当我们打开一个新的 Shell 时，我们必须重复上述过程来重新获得访问远程命令的权限。

当我们运行这些命令时，它们并不是在我们本地计算机上执行，而是隐式地在远程计算机上执行。在远程计算机上执行完成后，将结果发送给本地计算机。

我们可以想象出这样一个世界：我们永远不需要在本地计算机安装管理工具，这将避免多少麻烦。今天，你需要在本地操作系统上安装可运行的工具，并与你尝试管理的远程计算机进行通信——这使得几乎不可能匹配所有远程与本地的功能。而在未来，你无须再这么做。你将只需要使用隐式远程控制。服务器将通过 PowerShell 将其管理功能作为一个服务开放出来。

接下来到了坏消息时间：通过隐式远程连接获取到本地计算机的结果是反序列化的结果，这意味着对象的属性将会复制到一个 XML 文件中，以便通过网络进行传输。用这种方式收到的对象不会包含任何方法。在大多数情况下，这并不是一个问题。但你希望以编程的方式使用模块或插件时，这些模块或插件对隐式远程控制的支持就不会那么好了。我们希望该限制不会影响到你，这是由于对方法的依赖违反了一些 PowerShell 的设计实践。如果你用到了这些对象，则无法通过隐式远程控制的方式使用它们。

20.6 使用断开会话

PowerShell v3 对远程控制引入了两项提升。

首先，会话不再那么脆弱，意思是在网络闪断或其他传输中断的情况下，会话不会断开。即使在没有显式使用会话对象时，你也可以用到这项提升。即使你在使用类似 `Enter-PSSession` 和它的`-ComputerName` 参数时，从技术角度，你也是在底层使用了会话。因此，你获得了更稳定的连接。

在第 3 版中，你必须显式使用的一项功能：断开会话。比如你正在以用户 Admin1（是 Domain Admins 组成员）的身份连接到名称为 Computer1 的计算机上，并创建一个连接到名称为 COMPUTER2 的连接。

```
PS C:\> New-PSSession -ComputerName COMPUTER2

Id Name            ComputerName  State
-- ----------------- ------------- -----
 4 Session4         COMPUTER2     Opened
```

然后你就可以关闭连接。该操作仍然是在 Computer1 上进行的。当你完成该操作后，它会将两台计算机之间的连接断开，但会在 Computer2 上保留一份 PowerShell 的副本。注意，你可以通过指定 Session 的 ID 号完成该操作，该 ID 号会在你第一次创建 Session 时显示。

```
PS C:\> Disconnect-PSSession -Id 4

Id Name               ComputerName  State
-- ------------------ ------------- -----
 4 Session4           COMPUTER2     Disconnected
```

上面的内容值得你深入考虑——你在 COMPUTER2 上保留一份 PowerShell 的副本处于运行状态。因此为其分配一个适用的超时时间就变得很重要。在 PowerShell 早期的版本中，断开连接的 Session 将会被丢弃，所以无须清理工作。在第 3 版中，未被回收的会话可能会导致一些问题，这意味着你必须负责起回收工作。

但最酷的地方在于，我们可以登录到另一台计算机，也就是 COMPUTER3 上，用同样的域账号 Admin1，并获取运行在 COMPUTER2 上的会话列表。

```
PS C:\> Get-PSSession -computerName COMPUTER2

Id Name            ComputerName    State
-- ----------------- --------------- -----
 4 Session4         COMPUTER2       Disconnected
```

非常简单明了，不是吗？如果你以其他用户的身份登录，就无法看到这些会话。即使该身份为管理员，你也只能看到在 COMPUTER2 上创建的会话。既然已经看到了，那么你就可以重新连接。

```
PS C:\> Get-PSSession -computerName COMPUTER2 | Connect-PSSession

Id Name            ComputerName  State
-- ----------------- ------------- -------
 4 Session4         COMPUTER2     Open
```

我们花一些时间讨论和管理这些会话。在 PowerShell 的 WSMAN：Drive，你可以发现大量帮助你管控已断开会话的设置。你还可以通过组策略对大多数配置进行中心化管理。需要寻找的关键设置如下。

在 WSMan:\localhost\Shell 下：

- `-IdleTimeout` 指定当远程 Shell 中没有用户活动时，远程 Shell 将保持打开状态的最长时间。在指定的时间过后，远程 Shell 将被自动删除。默认值是 2 000 小时，或 84 天。
- `-MaxConcurrentUsers` 指定可以在同一计算机上通过远程 Shell 同时执行远程操作的最大用户数。
- `-MaxShellRunTime` 指定会话可以打开的最长时间。默认值为无限。请记住，IdleTimeout 参数可以覆盖该参数。
- `-MaxShellsPerUser` 指定任何用户可以在同一系统上远程打开的并发 Shell 的最大数目。将该值与 `MaxConcurrentUsers` 相乘，可以得到计算机上所有用户最大会话数量的值。

在 WSMan:\localhost\Service 下：

- `-MaxConnections` 设置连接到整个远程控制架构下的连接数上限。即使你设置了每个用户可运行的 Shell 数量或上限值的用户，`MaxConnections` 也会限制传入连接。

作为一个管理员，很明显你需要比普通用户有更高的责任心。你需要负责跟踪会话，尤其是你需要断开连接和重新连接。设置合理的超时时间，可以确保 Shell 的会话不会长时间闲置。

20.7 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。如果你只有一个客户端版本的计算机（运行 Windows 7 或 Windows 8），你就无法完成本实验中的第 6～9 步。

为了完成本次动手实验，你需要两台计算机：一台作为远程控制的控制端，另一台作为远程控制的接收端。如果你只有一台计算机，使用计算机名称对其进行远程控制。这种方式的体验和真正的远程连接非常类似。

提示： 在第 1 章中，我们提到了一个在 CloudShare.com 中的多计算机虚拟环境。你可以找到其他类似的基于云计算的虚拟主机。通过使用 CloudShare（www.cloudshare.com），我们无须部署 Windows 操作系统，这是由于该服务已经提供了供我们使用的模板。你当然需要为此服务付费，且该服务并不是对所有的国家可用。但如果你可以使用该服务，在本地没有环境时，这是获得一个实验环境的极佳方式。

1. 在 Shell 中关闭所有已打开的连接。

2. 建立一个连接到远程计算机的会话，并将会话存入一个命名为$session 的变量。

3. 利用$session 变量建立一个一对一到远程计算机的远程控制 Shell 会话。

4. 将 Invoke-Command 命令与$session 变量结合使用获取远程计算机上的服务列表。

5. 利用 Invoke-command 与 Get-PSSession 命令从远程计算机上获取最近 20 条远程安全事件日志条目。

6. 利用 Invoke-Command 与$session 变量在远程计算机上载入 ServerManager 模块。

7. 将 ServerManager 模块的命令由远程计算机导入到本地计算机，并使得“rem”成为命令名词部分的前缀。

8. 运行刚刚导入的 Get-WindowsFeature 命令。

9. 关闭储存在$session 变量中的会话。

注意：多亏了 PowerShell v3 中的新功能，你还可以利用 Import-Module 命令一步完成步骤 6 和步骤 7。请随意查看该命令的帮助文档，看看你是否能想出如何从远程计算机导入一个模块。

20.8 进一步学习

快速盘点一下你的环境：包含哪些启用 PowerShell 的产品？Exchange Server? SharePoint Server? VMware vSphere? System Center Virtual Machine Manager?上述产品或其他产品都包括 PowerShell 模块或插件，其中大多数插件或模块都可以通过 PowerShell 远程控制进行访问。

20.9 动手实验答案

```
1. get-pssession | Remove-PSSession
2. $session=new-pssession -computername localhost
3. enter-pssession $session
   Get-Process
    Exit
4. invoke-command -ScriptBlock { get-service } -Session $session
5. Invoke-Command -ScriptBlock {get-eventlog -LogName System-Newest
➥20} -Session (Get-PSSession)
```

6. Invoke-Command -ScriptBlock {Import-Module ServerManager}
➥-Session $session
7. Import-PSSession -Session $session -Prefix rem-Module ServerManager
8. Get-RemWindowsFeature
9. Remove-PSSession -Session $session