第 15 章　多任务后台作业

每个人都会跟你说“多任务”，对吧？为什么 PowerShell 不能同时处理多个任务实现“多任务”呢？事实证明，PowerShell 完全可以实现该功能，特别是涉及多台目标计算机的长时间运行的任务时。请确保在学习本章之前，你已经阅读过第 13 章和第 14 章，因为在本章中会更加深入地使用这些远程处理和 WMI 的概念。

15.1　利用 PowerShell 实现多任务同时处理

在你的印象中，你应该会将 PowerShell 视作一种单线程的应用程序，也就意味着 PowerShell 一次只能处理单个任务。你键入一条命令，然后按回车键，之后 PowerShell 就会等待该命令执行结束。除非第一条命令执行结束，否则你无法运行第二条命令。

但是借助于 PowerShell 的后台作业功能，它可以将一个命令移至另一个独立的后台线程（一个独立的，PowerShell 后台进程）。该功能使得命令以后台模式运行，这样你就可以使用 PowerShell 处理其他任务。但是你必须在执行该命令之前就决定是否这样处理，因为在按回车键之后，无法将一个正在运行的命令移至后台进程。

当命令处于后台模式时，PowerShell 会使用一些机制来查看这些进程的状态，获取产生的结果等。

注意： PowerShell for Linux 早期版本对 PowerShell 后台作业支持并不成熟。我们期望 PowerShell on Linux 最终会支持 PowerShell 后台作业但我们并不能保证它能支持到 Windows 版本那样。你已经了解如何阅读 PowerShell 帮助文档，本章会阐述这些概念。

15.2　同步 VS 异步

首先介绍一些术语。正常情况下，PowerShell 会使用同步模式执行命令，也就意味着，

在按回车键之后，你需要等待命令执行完毕。将一个命令置于后台模式将会使得该命名异步运行，也就是说，异步执行的命令在执行过程中，可以使用 PowerShell 处理其他任务。

下面是在两种模式中运行命令时的重要区别。

- 当在同步模式下运行命令时，可以响应输入请求；当使用后台模式运行命令时，根本就没有机会看到输入请求——实际上，当遇到输入请求时，会停止执行该命令。
- 在同步模式中，如果遇到错误，命令会立即返回错误信息；后台执行的命令也会产生错误信息，但是你无法立即查看这些信息。如果需要，你必须通过一些机制来获取这些信息（第 22 章会讲解如何实现）。
- 在同步模式中，如果忽略了某个命令的必要参数，PowerShell 会提示对应的缺失信息；如果是后台执行的命令，无法提示缺失信息，所以命令会执行失败。
- 在同步模式中，当命令开始产生执行结果时，就会立即返回；但是当命令处于后台模式时，你必须等待命令执行结束，才能获取缓存的执行结果。

通常情况下，我们会用同步模式执行命令，以便对这些命令进行测试，并使得可以正常工作。仅当它们被全面调试并能按照预期执行后，我们才会使用后台模式。我们只有遵循这些规则来保证命令的成功执行，这才是将命令置于后台模式的最好的时机。

PowerShell 将后台执行的命令称为作业（Jobs）。你可以通过多种方法创建作业，可以使用多种命令管理它们。

补充说明

严格意义上，本章中讨论的作业只是你将来会使用到的其中一种作业而已。本质上讲，作业只是 PowerShell 的一个扩展点，也就是说，对他人（不管是微软还是第三方）而言，都有可能创建其他功能（也命名为作业）。但是这些作业与本章描述的作业看起来并不一样，并且工作方式也不一样。实际上，本章末尾将讲到的调度作业（Scheduled Jobs）与本章前面提到的常规作业并不一致。当你为实现不同目的而扩展该 Shell 时，也会遇到很多其他一些作业。我们只是想让你知道这些小细节，并且理解到本章中所学的知识仅适用于 PowerShell 原生的常规作业。

15.3 创建本地作业

首先讲到的第一个作业类型应该是最简单的：本地作业。这是指一个命令几乎完全在你的本地计算机上运行（在后面会讲到对应的例外），并且该命令以后台模式运行。

为了创建这种类型的作业，你需要使用 `Start-Job` 命令。参数`-ScriptBlock` 使得你可以指定需要执行的命令（一个或多个）。PowerShell 会自动使用默认的作业名称（`Job1`、`Job2` 等）。当然，你也可以使用`-Name` 参数指定特定的作业名称。如果你需要作业运行在其他凭据下，那么可以使用`-Credential` 参数接受一个域名\用户名

（DOMAIN\UserName）的凭据，同时该参数也会使得提示输入密码。如果没有指定一个脚本块，你也可以使用-FilePath 参数来使得作业执行包含多个命令的完整脚本文件。

下面是一个简单的示例。

```
PS C:\> Start-Job -ScriptBlock {Dir}

Id      Name      PSJobTypeName    State      HasMoreData      Location
---     -----     -------------    ------     ------------     --------------
1       Job1      BackgroundJob    Running    True             localhost
```

该命令创建了一个作业对象，并且正如示例所示，该作业会立即开始运行。同时，该作业会按照顺序被赋予一个作业 ID 号，正如上面表格所示。

我们认为，这些作业完全运行于本地计算机上，的确如此。如果你执行一个可支持-ComputerName 参数的命令，在这种情形下，作业中的命令会被允许访问远程计算机。比如下面的示例。

```
PS C:\> Start-Job -ScriptBlock {
➥Get-EventLog Security -Computer Server-R2
➥}
Id       Name      PSJobTypeName       State     HasMoreData       Location
---      -----     --------------      ------     ------------     ------------
3        Job3      BackgroundJob       Running   True             localhost
```

动手实验： 我们期望你能持续跟随并执行所有的命令。如果你仅有一台计算机可以使用，请使用真实的本地计算机名称，同时用 localhost 作为第二台计算机，这样 PowerShell 会采用类似处理两台计算机的方式来运行命令。

作业的进程会在你本地计算机上运行，它会与指定的远程计算机进行连接（比如本示例中的 Server-R2）。所以从某种程度上说，这个作业就是一个“远程作业”。但是由于该命令实际上是在本地运行，所以我们仍然将它视为本地作业。

细心的读者可能已经注意到，创建的第一个作业被命名为 Job1，同时 ID 为 1，但是创建的第二个 Job 名为 Job3，同时 ID 为 3。原因是，每个作业至少都有包含一个子作业，第一个子作业（job1 的子作业）会被命名为 job2，其 ID 为 2。在本章后面章节会讲到子作业相关的知识。

另外，需要记住几点：尽管本地作业是在本地运行，但是它们也会需要使用 PowerShell 远程处理系统的架构，也就是在第 13 章中所讲的知识。如果你还没启用远程处理，那么将无法创建本地作业。

15.4 WMI 作业

创建作业的另一种方法是使用 Get-WMIObject 命令。正如我们在上一章所讲，Get-WMIObject 命令会与一台或多台远程计算机进行连接，但是通过串行方式实现。

这意味着如果给出一长串计算机名称，将需要花费很长的时间执行某个命令，那么将该命令移至后台作业就成为了必然选择。为了将该命令置为后台运行模式，像往常一样执行 Get-WMIObject 命令，但是需要加上-AsJob 参数。此时，你不能指定一个自定义的作业名称，只能使用 PowerShell 指定的默认作业名称。

动手实验： 如果你在测试环境中运行相同的命令，那么需要在 C:根目录下新建一个名为 allservers.txt 的文本文件（因为在这些示例中，均在该路径下执行命令），同时按照每行一个名称的格式在该文件中写入多个计算机名称。你可以将本地计算机名称，以及多个 localhost 放在该文件中，正如我们展示的这样。

```
PS C:\> Get-WMIObject Win32_OperatingSystem -ComputerName (
➥Get-Content allservers.txt) -AsJob

Id      Name      PSJobTypeName    State      HasMoreData     Location
--      ------    -------------    -------    ------------    ------------
5       Job5      WmiJob           Running    True            Server-R2,lo...
```

在该示例中，PowerShell 会创建一个上层的父作业（Job5，如上面返回结果中所示），同时会针对指定的每个计算机创建一个子作业。你可以看到，上面的输出表格的 Location 列中包含多个计算机名称，也就表明该作业也会在这些计算机上运行。

理解 Get-WMIObject 命令仅会在本地计算机上运行非常重要；该命令会使用正常的 WMI 通信机制与指定的远程计算机进行连接。它仍然一次只在一台计算机上执行，并且遵循直接跳过不可访问的计算机的默认规则等。实际上，该实现过程等同于同步执行 Get-WMIObject 命令，唯一不同点是该命令在后台运行。

动手实验： 你也会发现存在除 Get-WMIObject 外的其他命令来启动一个作业。尝试执行 Help * -Parameter AsJob，看看你是否可以找到所有的这种命令。

请注意，在第 14 章中学到的新的 Get-CimInstance 命令，并没有包含-AsJob 参数。如果你想在作业中使用该命令，请运行 Start-Job 或者 Invoke-Command（你即将学到的命令），并且将 Get-CIMInstance（或者说，任何新的 CIM 命令）放在脚本块中。

15.5 远程处理作业

下面介绍最后一种可以用来创建新作业的技术：PowerShell 的远程处理功能，也就是你在第 13 章中学习的功能。当使用 Get-WMIObject 时，你会使用-AsJob 参数实现该功能，但是这里我们会通过将该参数添加到 Invoke-Command Cmdlet 中实现。

这里有一个重要的不同点：在-ScriptBlock 参数（或者是该参数的别名，-Command）中指定的任意命令都会并行发送到指定的每台计算机。可以同时访问多达

32 台计算机（除非你修改了-ThrottleLimit 参数允许同时访问更多或者更少的计算机），所以当你指定了超过 32 个计算机名称，仅有前 32 台计算机会开始执行该命令。当在前 32 台计算机即将结束时，剩余的计算机才可以开始执行这些命令。另外，当在所有计算机上都执行结束后，上层的父作业会返回一个完整的状态。

与另外两种新建作业的方式不同，该技术要求你在每台目标计算机上安装第二版或者之后版本的 PowerShell，同时要求在每台目标计算机上 PowerShell 中均启用远程处理。因为命令会真正在每台远程计算机上执行，所以可以通过分布式计算工作负载提升复杂的或者长时间运行命令的性能。执行结果会返回到你的本地计算机。在你准备查看它们之前，结果都会与作业一起被存储。

在下面的示例中，你可以看到通过-JobName 参数指定一个特有的作业名称，这样就不需要无意义的默认名称。

```
PS C:\> Invoke-Command -Command {Get-Process}
➥-ComputerName (Get-Content .\allservers.txt )
➥-AsJob -JobName MyRemoteJob

Id      Name            PSJobTypeName   State      HasMoreData     Location
---     ----            -------------   -----      ------------    -----------
9       MyRemoteJob     RemoteJob       Running    True            Server-R2,loca...
```

15.6 获取作业执行结果

当开启一个作业之后，你最想做的第一件事应该就是确认作业是否执行结束。Get-Job 这个 Cmdlet 可以获取在系统中定义的所有作业，并且返回其状态。

```
PS C:\> Get-Job

Id       Name            PSJobTypeName   State      HasMoreData     Location
---      ----            -------------   -----      ------------    --------
1        Job1            BackgroundJob   Completed  True            localhost
3        Job3            BackgroundJob   Completed  True            localhost
5        Job5            WmiJob          Completed  True            Server-R2,loca...
9        MyRemoteJob     RemoteJob       Completed  True            Server-R2,loca...
```

你也可以通过作业 ID 或者名称去查询特定的作业信息。我们建议你可以尝试该命令并且将返回结果通过管道传递给 Format-List *，因为你已经收集了很多有用的信息。

```
PS C:\> get-job -id 1 | format-list *

State         : Completed
```

```
HasMoreData    : True
StatusMessage  :
Location       : localhost
Command        : dir
JobStateInfo   : Completed
Finished       : System.Threading.ManualResetEvent
InstanceId     : e1ddde9e-81e7-4b18-93c4-4c1d2a5c372c
Id             : 1
Name           : Job1
ChildJobs      : {Job2}
Output         : {}
Error          : {}
Progress       : {}
Verbose        : {}
Debug          : {}
Warning        : {}
```

动手实验：如果你一直跟着执行上面的命令，请记住，你的作业 ID 以及名称与上面返回的结果不一样。请通过 Get-Job Cmdlet 的结果获取你的环境中的作业 ID 与名称，然后使用它们来替换上面示例中对应的部分。同时请记住微软已经在过去几年 PowerShell 的几个版本不断扩展作业对象，输出结果的属性可能会有不同。

其中 ChildJobs 属性是返回信息中最重要的部分之一，在后面会讲到该部分。

为了获取一个作业的执行结果，请使用 Receive-Job 命令。但是在运行该 Cmdlet 之前，请先了解下面的一些知识点。

- 你必须指定希望获取返回结果的对应作业。可以通过作业 ID、作业名称，或者通过 Get-Job 命令获取作业列表，之后将它们通过管道传递给 Receive-Job 命令。
- 如果你获取了父作业的返回结果，那么该结果会包含所有子作业的输出结果。当然，你也可以获取一个或多个子作业的执行结果。
- 正常情况下，当获取了一个作业的返回结果之后，会自动在作业的输出缓存中清除对应的数据，这样你不能再次获取它们。可以通过-Keep 命令在内存中保留输出结果的一份拷贝。或者如果你希望保存一份拷贝以作它用，也可以将结果输出到 CliXML 中。
- 作业的返回结果可能是反序列化的对象，也就是你在第 13 章中所学的知识。也就意味着返回的结果是它们产生时的一个快照，它们可能不会包含可以执行的任何方法。但是如果需要的话，你直接将作业的返回结果通过管道传递给一些 Cmdlet，比如 Sort-Object、Format-List、Export-CSV、ConvertTo-HTML、Out-File 等。

下面是一个示例。

```
PS C:\>Receive-Job -ID 1

    Directory: C:\Users\Administrator\Documents
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ---------
d----          11/21/2009 11:53 AM     Integration Services Script Component

d----          11/21/2009 11:53 AM     Integration Services Script Task

d----           4/23/2010  7:54 AM     SQL Server Management Studio
d----           4/23/2010  7:55 AM     Visual Studio 2005
d----          11/21/2009 11:50 AM     Visual Studio 2008
```

前面的输出展示了一个比较有趣的结果。这里重新回忆一下最开始创建该作业的命令。

```
PS C:\> Start-Job -ScriptBlock { Dir }
```

尽管当运行该命令时，PowerShell 是在 C:\路径下，但是在结果中的路径却是 C:\Users\Administrator\Documents。正如你所见，本地作业运行时会在不同的上下文中，这可能会导致路径改变。当使用后台作业时，请永远不要猜测这些文件路径。因此需要使用绝对路径从而确保你可以关联作业命令可能需要的任何文件。如果我们希望后台作业获取 C:\下的目录信息，那么我们应该这样执行命令。

```
PS C:\> Start-Job -ScriptBlock { Dir C:\ }
```

当我们获取 Job1 的结果时，我们并没有指定-Keep 参数。如果我们再次获取这部分结果，不会得到任何信息，因为这部分结果已经没有与作业一同被缓存了。

```
PS C:\> Receive-Job -ID 1
PS C:\>
```

下面的命令展示了如何强制结果驻留在内存缓存中。

```
PS C:\>Receive-Job -ID 3 -Keep

  Index Time          EntryType    Source               InstanceID Message
  ----------          ----------   ---------            ---------- --------
  6542 Oct 04 11:55   SuccessA...  Microsoft-Windows...       4634 An...
  6541 Oct 04 11:55   SuccessA...  Microsoft-Windows...       4624 An...
  6540 Oct 04 11:55   SuccessA...  Microsoft-Windows...       4672 Sp...
  6539 Oct 04 11:54   SuccessA...  Microsoft-Windows...       4634 An...
```

你最终会希望释放用于缓存作业结果的内存，后面会进行对应的说明。但是首先，我们快速看一下如何将作业结果通过管道直接传递给其他 Cmdlet。

```
PS C:\>Receive-Job -Name MyRemoteJob | Sort-Object PSComputerName |
➥Format-Table -GroupBy PSComputerName

   PSComputerName: localhost

Handles  NPM(K)    PM(K)      WS(K)     VM(M)   CPU(s)     Id ProcessName PSComputerName
-------  ------    -----      -----     -----   ------     -- ----------- --------------
    195      10     2780       5692        30     0.70    484 lsm         loca...
    237      38    40704      36920       547     3.17   1244 Micro...    loca...
    146      17     3260       7192        60     0.20   3492 msdtc       loca...
   1318     100    42004      28896       154    15.31    476 sass        loca...
```

该作业是我们通过 Invoke-Command 命令所创建的。和以前一样，该 Cmdlet 会添加 PSComputerName 属性，这样我们就能追踪哪个对象来自于哪台计算机。因为我们从上层父作业中获取了结果，它包含了我们指定的所有计算机上的作业，这将允许命令可以对结果按照计算机名称进行排序，然后针对每台计算机创建独立的表组。

Get-Job 命令也会告知你还有哪些作业还留有剩余的结果。

```
PS C:\>Get-Job

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -------    -------------   --------
1               Job1            Completed  False           localhost
3               Job3            Completed  True            localhost
5               Job5            Completed  True            server-r2,lo...
8               MyRemoteJob     Completed  False           server-r2,lo...
```

当某个作业的输出结果没有被缓存时，对应的 HasMoreData 列为 False。在 Job1 和 MyRemoteJob 这两个场景中，我们已经获取了这部分结果，并且获取时并未指定-Keep 参数。

15.7 使用子作业

在前面我们提及，所有的作业都由一个上层父作业以及至少一个子作业组成。我们再次查看该作业。

```
PS C:\>Get-Job -ID 1 | Format-List *

State         : Completed
HasMoreData   : True
```

```
StatusMessage :
Location      :localhost
Command       :dir
JobStateInfo  : Completed
Finished      :System.Threading.ManualResetEvent
InstanceId    : e1ddde9e-81e7-4b18-93c4-4c1d2a5c372c
Id            : 1
Name          : Job1
ChildJobs     : {Job2}
Output        : {}
Error         : {}
Progress      : {}
Verbose       : {}
Debug         : {}
Warning       : {}
```

动手实验：不要照搬该部分的脚本，因为你如果自始至终都照搬的话，那么你之前已经获取ID为1的作业结果（也就是说，此时无法再次获取该结果）。如果你希望执行该脚本，那么请执行Start-Job -Script{Get-Service}新建一个作业，然后使用该作业ID替换我们示例中的ID。

你可以看到，Job1包含了一个子作业Job2。既然你知道了它的名字，那么你就可以直接获取该作业的信息。

```
PS C:\>Get-Job -Name Job2 | Format-List *

State         : Completed
StatusMessage :
HasMoreData   : True
Location      :localhost
Runspace      :System.Management.Automation.RemoteRunspace
Command       :dir
JobStateInfo  : Completed
Finished      :System.Threading.ManualResetEvent
InstanceId    : a21a91e7-549b-4be6-979d-2a896683313c
Id            : 2
Name          : Job2
ChildJobs     : {}
Output        : {Integration Services Script Component, Integration Services
                Script Task, SQL Server Management Studio, Visual Studio
                2005...}
Error         : {}
Progress      : {}
Verbose       : {}
Debug         : {}
Warning       : {}
```

有些时候，某个作业会包含多个子作业，它们都会以这种格式罗列出来。此时你可能希望采用不同的方式来罗列它们，比如下面这样。

```
PS C:\>Get-Job -ID 1 | Select-Object -Expand ChildJobs

WARNING: column "Command" does not fit into the display and was removed.
ID              Name            State      HasMoreData     Location
---             ----            -----      -----------     ------------
2               Job2            Completed  True            localhost
```

该技术会针对 ID 为 1 的作业创建一个表格用于存放子作业。该表格可以采用任意的长度，只要能将它们罗列出来。

你也可以使用 Receive-Job 命令指定作业名称或 ID 获取来自任意独立子作业的结果。

15.8 管理作业的命令

针对作业，也可以使用另外 3 个命令。对这 3 个命令中任意一个，你都可以指定作业 ID、作业名称，或者先获取作业信息，然后通过管道传递给这 3 个命令之一。

- Remove-Job——该命令会移除一个作业，包括从内存中移除该作业缓存的所有输出结果。
- Stop-Job——如果某个作业看起来卡住了，你可以通过执行该命令停止它。但是仍然可以获取截止到该时刻产生的结果。
- Wait-Job——该命令在下面场景中比较有用：当使用一段脚本开启一个作业，同时希望该脚本在作业运行完毕之后继续执行。该命令会使得 PowerShell 停止并等待作业执行，在作业执行结束后，允许 PowerShell 继续执行。

例如，为了移除已经获取了结果的作业，我们可以使用下面的命令。

```
PS C:\>Get-Job | Where { -Not $_.HasMoreData } | Remove-Job
PS C:\>Get-Job

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
---             -----           ------     ------------    ------------
3               Job3            Completed  True            localhost
5               Job5            Completed  True            server-r2,lo...
```

在 PowerShell 中，作业也可能执行失败，也就意味着在执行过程中发生了某些错误。考虑下面的示例。

```
PS C:\>Invoke-Command -Command { Nothing } -Computer NotOnline -AsJob -Job
Name ThisWillFail

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
---             ----            -----      -----------     ------------
11              ThisWillFail    Failed     False           NotOnline
```

在这里，我们向根本不存在的计算机发送一条不存在的命令来开启一个作业。当然，该作业立即就会失败，正如返回的 State 列。此时，我们根本就不需要使用 Stop-Job，因为该作业并未运行。但是我们仍然可以获取对应的子作业列表。

```
PS C:\>Get-Job -ID 11 | Format-List *

State         : Failed
HasMoreData   : False
StatusMessage :
Location      :notonline
Command       : nothing
JobStateInfo  : Failed
Finished      :System.Threading.ManualResetEvent
InstanceId    : d5f47bf7-53db-458d-8a08-07969305820e
ID            : 11
Name          :ThisWilLFail
ChildJobs     : {Job12}
Output        : {}
Error         : {}
Progress      : {}
Verbose       : {}
Debug         : {}
Warning       : {}
```

此时，我们就可以获取其子作业的信息了。

```
PS C:\>Get-Job -Name Job12

WARNING: column "Command" does not fit into the display and was removed.
Id              Name            State      HasMoreData     Location
--              -----           -----      ------------    ------------
12              Job12           Failed     False           NotOnline
```

正如你所见，该作业并没有产生任何输出，因此你并不能获取对应的结果。但是该作业的错误信息仍然保留在结果中，你可以使用 Receive-Job 命令获取这部分信息。

```
PS C:\>Receive-Job -Name Job12

Receive-Job: [NotOnline]Connecting to remote server failed with the following
error message:WinRM cannot process the request. The following error occured
while using Kerberos authentication:The network psth was not found.
```

完整的错误信息很长，在这里我们做了一些截断从而节省一些空间。你可以看到，错误信息中包含产生错误的计算机名称：[NotOnline]。当仅有某台计算机无法连接时会发生什么呢？我们看下面的示例：

```
PS C:\>Invoke-Command -Command { Nothing }
➥-Computer NotOnline,Server-R2 -AsJob -JobName ThisWillFail

警告：列“Command”无法显示，已被删除。
ID              Name            State      HasMoreData     Location
--              ----            ------     ------------    -----------
13              ThisWillFail    Running    True            NotOnline,Se...
```

稍待片刻，再执行下面的命令。

```
PS C:\>Get-Job

警告：列“Command”无法显示，已被删除。
ID              Name            State      HasMoreData     Location
--              -----           ------     ------------    ------------
13              ThisWillFail    Failed     False           NotOnline,Se...
```

可以看到该作业仍然失败，但是让我们检查一下独立的子作业状态。

```
PS C:\>Get-Job -id 13 | Select -expand ChildJobs

警告：列“Command”无法显示，已被删除。
ID              Name            State      HasMoreData     Location
--              -----           ------     -----------     -----------
14              Job14           Failed     False           NotOnline
15              Job15           Failed     False           Server-R2
```

好吧，它们都失败了。我们都能预感到 Job14 会失败，并且也知道失败的原因，但是 Job15 怎么了？

```
PS C:\>Receive-Job -Name Job15
Receive-Job : The term 'nothing' is not recognized as the name of a Cmdlet, function,
script file, or operable program. Check the spelling of the name, or if a path was
included, verify that the path is correct and try again.
```

对，这就是原因，我们让它执行了一个根本不存在的命令。正如你所见，每一个子作业都会由于不同的原因执行失败，PowerShell 能分别进行追踪。

15.9 调度作业

在 v3 版本的 PowerShell 中引入了针对调度作业的支持——可以在 Windows 的计划任务程序中使用 PowerShell 友好的方式创建任务。这里的作业与之前讲的那些作业相比，会采用不同的工作方式。正如前面写到的，作业是 PowerShell 中的一个扩展点，也就意味着允许存在多种通过不同方式实现的作业。调度作业正好是这些不同种类的作业中的一种。这种作业与标准计划任务作业有一点差别，这些作业的输出结果会存到磁盘中以供 PowerShell 后续进行使用。实际上，术语调度作业（scheduled jobs）与调度任务（scheduled tasks）并不同——前一种是与 PowerShell 相关的，后一种是你经常使用的传统作业。

你通过创建一个触发器（New-JobTrigger）开启一个调度作业，该触发器主要用于定义任务的运行时间。同时，你也可以使用 New-ScheduledTaskOption 命令设置该作业的选项。之后你使用 Register-ScheduledJob 命令将该作业注册到计划任务程序中。该命令采用计划任务程序中的 XML 格式来创建作业的定义，之后在磁盘上新建一个层级结构的文件夹存放每次作业运行的结果。

现在看下面的示例。

```
PS C:\> Register-ScheduledJob -Name DailyProcList -ScriptBlock { Get-Process }
➥-Trigger (New-JobTrigger -Daily -At 2am) -ScheduledJobOption
➥ (New-ScheduledJobOption -WakeToRun -RunElevated)

警告: 列“Enabled”无法显示，已被删除。
ID              Name            JobTriggers     Command
--              ----            -----------     --------------
1               DailyProcList   {1}             Get-Process
```

该命令会新建一个作业，该作业在每天凌晨两点执行 Get-Process 命令。如果有必要，会唤醒计算机，同时要求该作业运行在高级特权下。当作业执行完毕后，你可以回到 PowerShell 中，执行 Get-Job 查看每次该调度作业执行结束时的一个标准作业列表。

```
PS C:\>Get-Job

警告: 列“Command”无法显示，已被删除。
ID      Name            State       HasMoreData     Location
--      --------        -------     ------------    ------------
6       DailyProcList   Completed   True            localhost
9       DailyProcList   Completed   True            localhost
```

不像常规的作业，从调度作业中获取结果并不会导致结果被删除，因为它们是被缓存在磁盘上，而非内存中。之后可以继续多次获取该结果。当你移除这些作业时，对应的结果也会从磁盘上被移除。如图 15.1 所示，输出的结果会存放于磁盘上特定的文件夹中，Receive-Job 命令可以阅读这些结果。

你可以通过 Register-ScheduledJob 命令的-MaxResultCount 参数控制存放的结果数量。

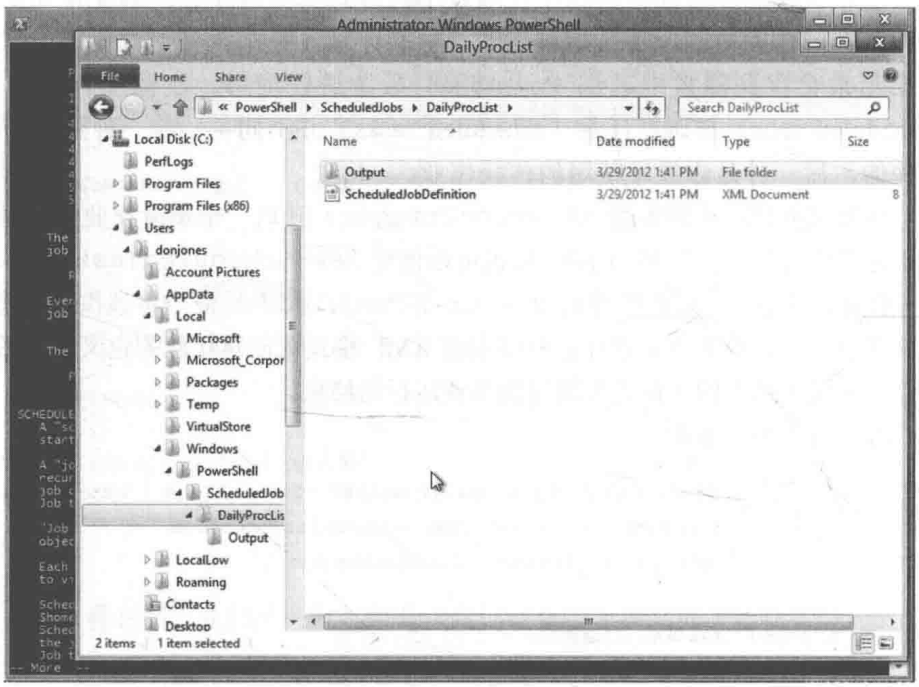

图 15.1 调度作业的输出结果存放于磁盘

15.10 常见困惑点

一般情况下，作业都是比较简单的，但是我们曾经见到其他人做导致混淆的事。请不要这样做。

```
PS C:\>Invoke-Command -Command { Start-Job -ScriptBlock { Dir } }
➥-ComputerName Server-R2
```

执行该命令之后，会对 Server-R2 计算机开启一个临时的连接，并且在该计算机上开启一个本地作业。遗憾的是，该连接会立即中断，这样就导致你无法重新连接并且获取该作业的信息。一般而言，不要混淆和随意匹配开启作业的 3 种方式。

比如下面的命令也不是一个好主意。

```
PS C:\>Start-Job -ScriptBlock { Invoke-Command -Command { Dir }
➥-ComputerName SERVER-R2 }
```

该命令过于冗长了；完全可以通过保留 Invoke-Command 部分，之后使用-AsJob 参数来使得该作业在后台运行。

更少的混淆，但同样有趣的是教室里学生经常问到的关于作业的一些问题。其中最重要的一个问题可能是“我们是否可以看到由其他人开启的作业呢”，这里的答案是“不能”，但是调度作业例外。常规的作业完全包含在 PowerShell 进程中。尽管你可以看到其他用户在运行 PowerShell，但是你还是没有办法看到该进程内部的一些信息。这和其他应用程序一样。例如，你可以看到他人有运行微软的 Word 软件，但是你无法看到他们正在编辑的文档，因为这些文档完全隐藏于 Word 的进程中。

仅当 PowerShell 进程开启，作业才会维持。当你关闭进程后，在进程中定义的任何作业就会消失。无法在 PowerShell 外部的任意地方定义作业，所以它们依赖于继续运行的进程，保证可以自行维护。

针对前面的论述，调度作业是一个例外：具有权限的任何人都可以看到它们，修改它们，删除它们，以及获取它们的结果。这是因为它们存放于物理磁盘上。请注意，它们存放于你的用户配置文件下，因此它通常要求管理员从配置文件中获取文件（和结果）。

15.11 动手实验

动手实验：对于本章的动手实验环节，你需要操作系统为 Windows 8（或之后）或者 Windows Server 2012（或之后）运行 PowerShell v3 或更新版本的计算机。

下面的实验应该能帮助你理解如何在 PowerShell 中使用各种类型的作业以及任务。在进行这些实验时，请不要要求自己仅通过单行代码实现。某些时候，可能将它们拆成独立的步骤会更容易。

1．创建一次性的后台作业用于寻找 C:驱动器中所有的 PowerShell 脚本。任何需要很长时间运行完成的任务都比较合适。

2．你意识到该后台作业在一些服务器上识别所有 PowerShell 脚本非常有效。你如何在一组远程计算机上运行任务 1 中相同的命令呢？

3．创建一个后台作业获取计算机上系统事件日志中最近的 25 条错误记录，之后将记录导出为 CliXML。你期望在每周一到周五的早上 6 点运行，这样当你上班时就可以查看这些作业。

4．你会使用哪个 Cmdlet 获取一个作业的结果，然后在作业队列中如何存放这些结果？

15.12 动手实验答案

1. `Start-Job {dir c:\ -recurse -filter '*.ps1'}`
2. `Invoke-Command -scriptblock {dir c:\ -recurse -filter *.ps1} -computername (get-content computers.txt) -asjob`
3.
```
$Trigger=New-JobTrigger -At "6:00AM" -DaysOfWeek "Monday", "Tuesday","Wednesday","Thursday","Friday" -Weekly
$command={ Get-EventLog -LogName System -Newest 25 -EntryType Error | Export-Clixml c:\work\25SysErr.xml}
Register-ScheduledJob -Name "Get 25 System Errors" -ScriptBlock $Command -Trigger $Trigger
#检查被创建的作业
Get-ScheduledJob | Select *
```
4. `Receive-Job -id 1 -keep`
   当然，你可以使用任意适用的作业 ID 或作业名称。