第 8 章　对象：数据的另一个名称

在本章我们将会尝试做一些不同的事情。我们发现 PowerShell 中对于对象的使用是最让人困惑的内容之一，但同时也是 Shell 中最关键的内容，影响在 Shell 中的所有操作。这些年我们尝试通过不同的方式对该概念进行阐述，最终我们找到了能够让完全不同背景的受众都能接受的阐述方式。如果你之前曾有过编程经验并因此很容易能够接受对象的概念，请跳过 8.2 节。如果你没有编程背景且没有在脚本语言或编程语言中使用过对象，请从 8.1 节开始阅读本章。

8.1　什么是对象

花一点时间运行 PowerShell 中的 Get-Process。可以看到一个包含多列的表格，但这些信息仅仅是关于进程的冰山一角。进程对象还包括机器名、主窗口句柄、最大工作集大小、退出代码和时间、处理器掩码信息以及其他大量信息。实际上，你可以找出超过 60 个与进程有关的信息。为什么 PowerShell 仅仅展示少量的信息呢？

原因非常简单，PowerShell 当然可以提供屏幕上所无法容纳的更多的信息。当运行任意命令，比如 Get-Process、Get-Service、Get-EventLog 或其他命令时，PowerShell 会完全在内存中构造用于容纳关于项的所有信息的表格。例如 Get-Process，该表格由 67 列组成，每行对应运行在计算机中的一个进程。每一列包含一部分信息，比如说虚拟内存、CPU 利用率、处理器名称、进程 ID 等。然后，PowerShell 会检查你是否指定所需查看的列。如果你未指定（目前我们还没展示如何指定）想查看的列，Shell 会查看由微软提供的配置文件并只显示微软认为你希望查看的列。

一种查看所有列的方式是使用 ConvertTo-HTML 命令：

```
Get-Process | ConvertTo-HTML | Out-File processes.html
```

该命令不会过滤列，而是生成包含所有列的 HTML 文件。这是查看整个表的一种方式。

除去包含这些信息的列之外，表中每一行都有一些与之对应的方法。这些方法包括操作系统可以对进程进行的操作。比如说，操作系统可以关闭进程、杀死进程、刷新信息，或者等待进程退出等。

每当运行一个可以产生结果的命令时，输出结果在内存中以表的形式存放。当将输出结果以管道的方式由一个命令传送给另一个命令时，比如说：

```
Get-Process | ConvertTo-HTML
```

整个表通过管道进行传输。该表在传输过程中并不会过滤到只有一小部分列，而是直到所有的命令都运行后才会进行过滤。

下面是一些术语的变化。PowerShell 并不会将这些内存中的表命名为“表”，而是使用下述 4 个术语：

- 对象——这也就是所谓的“表行”。它代表单个事物，比如说单个进程或是单个服务。
- 属性——这也就是所谓的“表列”。它代表关于对象的一部分信息，比如说进程名称、进程 ID 或服务状态。
- 方法——这也就是所谓的“行为”。方法与某个对象关联并使得对象完成某些任务，比如说杀死进程或启动服务。
- 集合——这是整个对象的集合，我们曾称之为“表”。

如果你发现下面对于对象的讨论让你感到困惑，请随时回头参考上面包含 4 个要点的列表。请总是将对象的集合想象成一个在内存中巨大的信息表，表中一行即为对象而列为属性。

8.2 为什么 PowerShell 使用对象

PowerShell 使用对象来代表数据的一个原因是，当然你总需要某种方式代表数据，对吧？PowerShell 可以将数据以类似 XML 的格式存储，或使用纯文本表来代表。但微软是有一些具体的理由不这么做。

第一个原因是 Windows 本身就是一个面向对象的操作系统——或者至少，大部分在 Windows 上运行的软件是面向对象的。选择将数据构建成对象集合的方式将非常容易，因为大部分操作系统喜欢这种结构。

另一个使用对象的原因是这样会使事情简单，并给你提供更加强大的功能和更好的灵活性。现在，让我们假装 PowerShell 并不会生成对象作为命令的输出结果，而是生成一个简单的文本表。这也是你一开始认为的方式。当你运行类似 Get-Process 的命令时，

你将会得到格式化好的文本作为输出结果：

```
PS C:\> get-process
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  ProcessName
-------  ------    -----      ----- -----   ------     --  -----------
     39       5     1876       4340    52    11.33   1920  conhost
     31       4      792       2260    22     0.00   2460  conhost
     29       4      828       2284    41     0.25   3192  conhost
    574      12     1864       3896    43     1.30    316  csrss
    181      13     5892       6348    59     9.14    356  csrss
    306      29    13936      18312   139     4.36   1300  dfsrs
    125      15     2528       6048    37     0.17   1756  dfssvc
   5159    7329    85052      86436   118     1.80   1356  dns
```

如果我们希望针对上述信息进行一些操作时会怎样？或许你希望针对所有运行 Conhost 的进程进行操作。为了完成该项操作，你必须对进程列表进行过滤。在 Unix 或 Linux Shell 中，你需要使用类似 Grep 的命令，并告诉该命令“请帮我检查这个文本列表，仅保留第 58～64 列包含‘conhost’字符的行，并删除其他行”。结果列表将会仅包含你所指定的进程：

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  ProcessName
-------  ------    -----      ----- -----   ------     --  -----------
     39       5     1876       4340    52    11.33   1920  conhost
     31       4      792       2260    22     0.00   2460  conhost
     29       4      828       2284    41     0.25   3192  conhost
```

接下来将上述文本通过管道传递给另一个命令，比如说从列表中获取进程 ID。“从第 52～56 列中获取字符，但丢弃前两列。”结果可能为：

```
1920
2460
3192
```

最终，你将上述文本通过管道传递给另一个命令，使用该命令杀死这些 ID 所代表的进程（或任何你希望做的操作）。

这实际上也是 Unix 和 Linux 管理员的工作。他们花费大量的时间学习如何更好地解析文本，使用类似 Grep、Awk 和 Sed 等工具，并必须熟练使用正则表达式。这一系列过程使得他们更容易定义他们希望计算机查找的文本模式。Unix 和 Linux 从业人员喜欢类似 Perl 的语言，因为该语言包含丰富的文本解析和文本操作方法。

但这种基于文本的方式存在一些问题：

- 你需要花费更多的时间在文本中打转，而不是完成真正的工作。
- 如果命令的输出结果改变——比如说，将 ProcessName 列移到表的第一列——你需要重写所有的命令，这是因为这些命令需要依赖列位置之类的东西。

- 你需要善于使用解析文本的语言或工具。不仅由于你的工作需要解析文本，解析文本还是实现目的的手段。

PowerShell 使用对象消除所有的文本操作开销。由于对象的工作机制类似内存中的表，因此你无须告知 PowerShell 信息所在的文本位置，而是仅仅需要输入列名。无论在屏幕或文件中如何组织输出结果，PowerShell 都知道去哪里获取数据，内存表总是同一个，因此你永远都不需要由于列移动而重写命令。这样的好处是你更多专注于如何实现功能，而不是这类不必要的开销。

当然，你必须学习一些使得你可以构建 PowerShell 属性的语法，但所需学习的内容将会比那些纯粹基于文本的 Shell 要少很多。

8.3 探索对象：Get-Member

如果说对象就像内存中一个巨大的表，而 PowerShell 仅仅在屏幕上展示表的一部分，那么如何看到其他你需要使用的属性呢？此时如果你想到使用 Help 命令，我们会很欣慰，因为毕竟我们在之前章节不遗余力地推崇使用帮助。但遗憾的是，这并不对。

帮助系统仅记录背景概念（以“关于”帮助主题的形式）和命令语法。如果需要了解更多关于对象的内容，使用另一个命令：Get-Member。你应该习惯于使用该命令。实际上，你更应该了解输入该命令的快捷方式。我们现在就提供给你：别名 Gm。

可以在任何产生某些输出的命令之后使用 Gm。例如，你已经知道运行 Get-Process 会在屏幕上产生一些输出，你可以将这些输出通过管道传送给 Gm：

```
Get-Process | Gm
```

当一个 Cmdlet 产生一个对象的集合时，就像 Get-Process 命令那样，整个集合直到管道末尾之前都可以被访问。直到最后一个命令运行完之前，PowerShell 都不会将对象的 89 个标签或属性过滤掉。直到最后一个命令运行完，才会创建你所见到的文本输出结果。因此在之前的例子中，Gm 可以完整访问进程对象的属性和方法，这是由于该命令还未被过滤用于显示。Gm 会查看每一个对象并构建一个包含对象属性和方法的列表，该列表内容如下：

```
PS C:\> get-process | gm

   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
```

```
NPM                 AliasProperty   NPM = NonpagedSystemMemo...
PM                  AliasProperty   PM = PagedMemorySize
VM                  AliasProperty   VM = VirtualMemorySize
WS                  AliasProperty   WS = WorkingSet
Disposed            Event           System.EventHandler Disp...
ErrorDataReceived   Event           System.Diagnostics.DataR...
Exited              Event           System.EventHandler Exit...
OutputDataReceived  Event           System.Diagnostics.DataR...
BeginErrorReadLine  Method          System.Void BeginErrorRe...
BeginOutputReadLine Method          System.Void BeginOutputR...
CancelErrorRead     Method          System.Void CancelErrorR...
CancelOutputRead    Method          System.Void CancelOutput...
```

由于列表过长，我们对上述列表进行了裁剪。但愿你能够理解其中的意思。

动手实验：不要只相信我们所说的。现在你可以趁热打铁运行一些我们提供的命令，以便查看完整的输出结果。

顺便说一下，还有一个可能会让你感兴趣的知识点，就是一个对象的属性、方法以及其他附加到对象的东西都被称为成员。就好像对象本身是一个乡村俱乐部，所有属性和方法都是俱乐部的成员。这也是 Get-Member 名称的由来：该命令获取对象成员的列表。但请记住，PowerShell 中的惯例是使用单数名词，所以 Cmdlet 的名称为 Get-Member，而不是“Get-Members”。

重要：请注意 Get-Member 输出结果的第一行，这一行很容易被忽视。这一行是 TypeName，是分配给特定类型对象的唯一名称。它现在看起来好像并不重要——毕竟，谁会关心它的名称呢？但该名称将会在下一章节成为关键内容。

8.4 对象标签，也就是所谓的“属性”

当你查看 Gm 的输出结果时，你会注意到一些不同种类的属性：

- 脚本属性；
- 属性；
- NoteProperty；
- 别名属性。

补充说明

通常来说，.Net Framework 中的对象——也就是所有 PowerShell 对象的来源——只包含“属性”。PowerShell 会动态添加其他内容：ScriptProperty、NoteProperty、AliasProperty 等。如果你正好在微软的 MSDN 文档中查看某个对象类型（你可以将对象的类型名称输入 MSDN 的搜索框），你无法找到这些额外的属性。

PowerShell 有一个扩展类型系统（ETS）负责添加这些后来的属性。为什么它会这么做？

拿某些案例来说，它使得对象具有更好的一致性，比如为原生只具有类似 ProcessName 属性的对象添加 Name 属性（这也是别名属性的作用）。还有一些情况是暴露对象中隐藏的一些信息（进程对象包含一些脚本属性完成这项工作）。

当你在 PowerShell 的世界中，这些属性的行为都会变得一致。但当这些属性并没有在官方文档页面中出现时，也请不要惊讶：Shell 会自动添加这些额外的属性，通常会使得你的工作更加轻松。

对实现你的目标来说，这些属性都一样，唯一的区别是属性原本是如何被创建出来的。但你不必担心这些。对你来说，这些都是“属性”，使用的方法并无不同。

属性总是包含一个值。例如，进程对象的 ID 属性可能是 1234，对象的名称属性的值可能是 NotePad。属性用于描述关于对象的某些方面：它的状态、它的 ID、它的名称等。在 PowerShell 中，属性通常是只读的，意味着你无法通过给 Name 属性赋一个新值来改变服务的名称。但你可以通过读取 Name 属性来获取服务的名称。我们估计你在 PowerShell 中 90%的工作都需要与属性打交道。

8.5 对象行为，也就是所谓的“方法”

很多对象都支持一个或多个方法，正如我们之前提到过的，是你可以指导对象的行为。进程对象包含一个 Kill 方法，它会终止进程。某些方法需要一个或多个输入参数来为某个行为提供额外的细节信息，但在早期的 PowerShell 学习中，你不会遇到这些需要参数的方法。实际上，你可能使用多个月甚至多年 PowerShell 而从来不需要执行一个有参数的方法，这是由于这些方法可以和 Cmdlets 互相替代。

例如，如果你需要终止进出那个，可以通过三个办法实现。其中一个办法是获取对象并执行 Kill 方法，另一个办法是使用一系列 Cmdlets：

```
Get-Process -Name Notepad | Stop-Process
```

你还可以使用单个 Cmdlet 完成这项任务：

```
Stop-Process -name Notepad
```

在整本书中，我们更专注于使用 PowerShell Cmdlet 完成任务。Cmdlet 提供了最简单、最管理员导向、最聚焦任务的方式完成工作。而使用方法就开始进入.NET Framework 编程的领域，这会更加复杂且需要更多的背景知识。鉴于此，你将会很少——或是从不看到我们在本书中执行对象的方法。实际上，我们在这一点上的哲学是：“如果无法通过 Cmdlet 完成，那就回头使用 GUI 完成”。相信我们，在你的职业生涯中都不会感受到这种哲学。但现在来说，保持使用“PowerShell 的方式”做事是一个不错的办法。

补充说明

在学习 PowerShell 的本阶段，你无须懂得关于对象方法的知识。但除了属性和方法之外，对象还有一个事件。事件是以对象的方式通知你某些事情发生了。一个进程对象，举例来说，可以在进程结束时触发 Exited 事件。你可以将你自己的命令附加到这些事件上，比如说，当进程结束时发送一封邮件。以这种方式和事件交互是高级主题，并且超出了本书的范畴。

8.6 排序对象

大部分 PowerShell Cmdlets 以确定性的方式产生对象，这意味着每次运行命令时都会以相同的顺序产生对象。例如，服务和进程都按照字母表顺序对名称进行排序。事件日志倾向于按照事件排序。那么假如我们希望改变排序方式，该如何做？

例如，假设我们希望显示一个进程列表，按照对虚拟内存（Vitrual Memory，VM）的消耗由高到低进行排列。我们将需要基于 VM 属性对列表进行重新排序。PowerShell 提供了一个简单的 Cmdlet、Sort-Object，就像其名称那样，可以对对象进行排序：

```
Get-Process | Sort-Object -property VM
```

动手实验：我们希望你运行一些命令。我们不会将输出结果写入书中，因为输出结果表有点长。但如果你跟着教程运行，你会在你的屏幕上得到同样的结果。

该命令并不是我们最终想要的结果。它虽然以 VM 进行排序，但是以升序形式，最大值在列表底部。通过阅读 Sort-Object，可以发现-descending 参数可以反转排序。我们还注意到，-property 参数是定位参数，因此无须输入参数名称。我们还告诉过你 Sort-Object 有一个别名，也就是 Sort，所以你可以在下一个动手实验中少输入一些内容：

```
Get-Process | Sort VM -desc
```

我们还将-descending 简化为-desc，仍然可以得到想要的结果。-property 参数接受多个值（如果你查看过帮助文件，我们确定你可以发现这一点）。

为了以防两个进程使用的虚拟内存相同，我们还希望按照进程 ID 进行排序。下述命令可以实现这一点：

```
Get-Process | Sort VM, ID -desc
```

和之前一样，通过以逗号分隔列表的方式将多个值传递给任意支持多个值的参数。

8.7 选择所需的属性

另一个有用的 Cmdlet 是 Select-Object。该 Cmdlet 从管道接受对象，你可以指定希望显示的属性。这使得你可以访问任意属性，减少返回列表，只返回你感兴趣的列，而

默认情况下由 PowerShell 配置规则控制。这对于将对象输出到 HTML 的 ConvertTo-HTML 命令来说非常有用，因为该 Cmdlet 通常会创建包含所有属性的表。

比较下面两个命令的结果：

```
Get-Process | ConvertTo-HTML | Out-File test1.html
Get-Process | Select-Object -property Name, ID, VM, PM |
➥ConvertTo-HTML | Out-File test2.html
```

动手实验：请尝试分别运行上述命令，然后在 IE 浏览器中查看输出的 HTML 结果，以比较区别。

请花一些时间查看 Select-Object（或者可以使用该命令别名：Select）。-property 参数看上去是定位参数，这意味着我们可以将上面运行的命令缩短：

```
Get-Process | Select Name, ID, VM, PM | ConvertTo-HTML | Out-File test3.html
```

请花一些时间体验 Select-Object。实际上，可以修改下述命令进行其他尝试，该命令将结果展现在屏幕上。

```
Get-Process | Select Name, ID, VM, PM
```

请尝试从列表中添加或删除不同的进程对象属性并查看结果。在最多可以指定多少属性的情况下保持输出结果以表的形式展现？在选择多少属性的情况下就会强制 PowerShell 在输出结果中使用别名而不是表？

补充说明

Select-Object 还拥有-First 和-Last 参数，这两个参数可以保留管道中对象的子集。例如，Get-Process | Select First 10 将会保留前 10 个对象。但不能加过滤条件，比如选择特定的进程，只能选择前（或最后）10 个。

警告：人们经常会将 Select-Object 和 Where-Object 这两个 PowerShell 命令搞混，虽然目前你还没有见过 Where-Object。Select-Object 用于选择所需的属性（或列），还可以选择输出行的任意子集（使用-First 和-Last）。Where-object 基于筛选条件从管道中移除或过滤对象。

8.8 在命令结束之前总是对象的形式

PowerShell 管道在最后一个命令执行之前总是传递对象。在最后一个命令执行时，PowerShell 将会查看管道中所包含的对象，并根据不同的配置文件决定哪一个属性被用于构建展示在屏幕上的最终结果。它还会基于一些内部规则和配置文件确定展示是表还是列表（我们将会在接下来的章节阐述更多关于这些规则和配置，以及如何修改它们）。

一个重要的事实是，在一个命令行中，管道可以包含不同类型的对象。在接下来的例子中，我们将会选择一个命令行，并且每一个命令单独占一行，这样将更容易解释我们所谈论的内容。

下面是第一个示例。

```
Get-Process |
Sort-Object VM -descending |
Out-File c:\procs.txt
```

在本例中，首先运行 Get-Process，该命令将进程对象放入管道。下一个命令是 Sort-Object，该命令并不会改变管道中的内容，仅仅是改变对象的顺序，直到 Sort-Object 结束，管道仍然只包含进程。最后一个命令是 Out-File。在这里，PowerShell 生成输出结果，也就是管道中所包含的内容——进程对象，并根据 PowerShell 的内部规则将对象格式化，最终结果存入指定文件。

接下来是一个稍复杂的例子。

```
Get-Process |
Sort-Object VM -descending |
Select-Object Name, ID, VM
```

该命令以同样的方式运行。Get-Process 将进程对象放入管道。接下来运行 Sort-Object，该命令将同样的进程对象放入管道。但 Select-Object 就有所不同了。进程对象总是拥有相同的成员。Select-Object 并不能通过删除你不需要的属性减少属性列表。如果这样的话，结果就不再是进程对象，而是 Select-Object 创建一个名为 PSObject 的自定义对象，PowerShell 使用这个对象将属性从进程对象中复制出来，结果是自定义对象被放入管道。

动手实验：尝试在一个命令行中输入上述 3 个 Cmdlet。请记住，你需要在一行中输入所有的命令。请注意输出结果和正常运行 Get-Process 的输出结果有何不同。

当 PowerShell 发现光标已经到达命令行结尾时，它必须知道如何对文本输出结果进行排版。这是由于管道中包含的对象不再是进程对象，PowerShell 不会再将默认规则和配置应用于进程对象，而是通过查询 PSObject 的规则和配置，这也是当前管道中包含的配置类型。由于 PSObjects 用于自定义输出，微软并没有为 PSObjects 提供任何规则或配置。而是 PowerShell 将尽最大努力进行猜测并产生表。在理论上，产生的表可以容纳上述 3 列信息，但表并不像正常的 Get-Process 输出结果那样排版很好看，这是由于 Shell 缺少使得表更好看的额外的配置信息。

你可以使用 Gm 查看管道中不同的对象。请记住，你可以在任何产生输出结果的 Cmdlet 之后使用 Gm。

```
Get-Process | Sort VM -descending | gm
Get-Process | Sort VM -descending | Select Name, ID, VM | gm
```

动手实验：请分别运行上述两个命令，并查看输出结果的区别。

请注意，PowerShell 会展示出管道中对象的类型名称作为 Gm 输出结果的一部分。在第一个例子中，对象类型为 System.Diagnostics.Process，但是在第二个例子中，管道里包含另一种类型的对象。这个新的“经过筛选”的对象仅包含 3 个指定属性——Name、ID 和 VM，以及另外一些由系统生成的成员。

即便 Gm 产生对象并将对象放入管道，在运行 Gm 之后，管道也不再包含进程对象或是“经过筛选”的对象，它仅包含由 Gm 生成的对象类型：Microsoft.PowerShell.Commands.MemberDefinition。你可以通过在管道中对 Gm 的输出结果再次使用 Gm 命令证明：

```
Get-Process | Gm | Gm
```

动手实验：你一定很想尝试该命令，该命令让人感到有些费解。首先是 Get-Process 命令，将进程对象放入管道。然后运行 Gm，该命令分析进程对象并生成该对象的 MemberDefinition 对象。然后将结果再次利用管道传输给 Gm，该命令将分析并产生 MemberDefinition 成员列表作为输出结果。

掌握 PowerShell 的一个关键点是在任意时间点知道当前管道中的对象类型。Gm 可以帮助你实现这一点，但自己将整个命令从头到尾过一遍将会更好地帮助你理清头绪。

8.9 常见误区

参加我们课程的学生在开始学习 PowerShell 时通常会犯一些错误，虽然随着经验的积累，这些错误都会被修正，但我们还是希望他们所犯的错误会引起你的警觉。下面的列表可以帮助你在走错方向时及时改正。

- 请记住，PowerShell 帮助文件不包括有关对象属性的信息。你必须将对象利用管道传输给 Gm（Get-Member）来查看属性列表。
- 请记住，你可以在产生结果的任意管道末尾添加 Gm 命令。类似 Get-Process -name Notepad | Stop-Process 的命令行正常情况下不产生结果，所以将|Gm 置于管道末尾不会产生任何结果。
- 请注意输入的整洁性。请在管道操作符两边加入空格，这是由于命令行看起来更像 Get-Process | Gm，而不是 Get-Process|Gm。在这里添加空格是有原因的，请使用空格。
- 请记住，管道中在不同阶段可以包含不同类型的对象。请考虑当前在管道中的对象类型是什么，并把精力集中在下一个命令对当前类型的对象所做的操作。

8.10 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。

目前为止，本章或许比其他章节覆盖了更多、更难以及更新的知识点。希望我们的讲述方式能够帮你理解这些概念。下面的练习可以帮助你巩固所学到的知识。请尝试完成所有练习，并根据 MoreLunches.com 的配套视频和示例代码辅助你的学习。其中一部分任务需要你利用在之前章节所学的知识，这是为了帮你巩固之前的知识。

1．找出生成随机数字的 Cmdlet。

2．找出显示当前时间和日期的 Cmdlet。

3．任务#2 的 Cmdlet 产生的对象类型是什么？（由 Cmdlet 产生的对象类型名称是什么？）。

4．使用任务#2 中的 Cmdlet 和 Select-Object，仅显示是星期几，示例如下（警告：输出结果将会靠右对齐，请确定 PowerShell 窗口没有水平滚动条）：

```
DayOfWeek
---------
    Monday
```

5．找出可以显示已安装的补丁（hotfix）的 Cmdlet。

6．使用任务#5 的 Cmdlet 显示已安装的补丁列表，按照安装日期对列表进行排序，并仅仅显示如下几列：安装日期、补丁 ID、安装用户。请记住，在命令默认输出显示的列头并不一定是属性的实际名称——你需要查找实际的属性名称来确保这一点。

7．重复任务#6，但这次按照补丁描述对结果进行排序，并输出描述、补丁 ID、安装日期列，最终将结果保存到 HTML 文件。

8．从安全事件日志中显示最新的 50 条列表（如果安全事件列表为空，你也可以使用其他日志，比如系统或应用程序日志）。按照时间升序对日志进行排序，同时也按照索引排序。显示索引、时间以及每条记录的来源。将这些信息存入文本文件（不是 HTML 文件，而是纯文本文件）。你可以尝试使用 Select-Object 以及它们的-first 或-last 参数实现本任务；但请不要这么做，还会有更好的方法。同时目前请避免使用 Get-Winevent；可以使用一个更好的 Cmdlet 完成本任务。