

10

第 10 章 格式化及如何正确使用

现在快速回顾一下：你已经知道 PowerShell Cmdlets 可以用于生成对象，并且这些对象通常含有比 PowerShell 默认所显示的属性多。你也已经知道如何使用“Gm”命令获取一个对象的所有属性，以及如何使用“Select-Object”自定义你想看到的属性。到目前为止，你看到的基本上都是通过 PowerShell 的默认配置和规则把结果输出到显示器上（或者文件形式和硬拷贝格式）。本章将会介绍如何覆盖这些默认值并为命令创建自定义输出格式。

10.1 格式化：让输出更加美观

我们并不想让你感觉 PowerShell 是成熟的报表管理工具，因为 PowerShell 的确不是。但 PowerShell 的确在收集计算机的信息方面是一把好手，如果有恰当的输出结果，你当然可以使用这些信息生成报表。目的是获得正确的输出结果，这也是格式化的意义所在。

表面上看，PowerShell 的格式化系统貌似很容易（大部分情况下也的确如此）。但有时候一些需要技巧的方式会让你掉入陷阱中，所以希望你能明白它的工作机制与工作原理。本章不打算展示新命令，而是解释整个系统的工作机制，交互方式，以及限制。

10.2 默认格式

现在运行一下我们熟悉的命令“Get-Process”，然后注意结果的列头部分。可以看到，报表列头部分的名称与属性名称并不完全相符。每个列头都有固定的宽度、别名等。你是否意识到这些结果来自于某些配置文件？你可以在安装 PowerShell 的路径下找到其中一个名为“.format.pslxml”的文件。其中进程对象的格式化目录在“DotNetTypes.format.pslxml”中。

动手实验：接下来你需要一直打开 PowerShell，以便跟随我们的脚步前进，并且从中理解格式化系统的底层结构。

下面我们先修改 PowerShell 的安装目录，并且打开“DotNetType.format.pslxml”文件。注意，不要保存对该文件的任何变更。该文件带有数字签名，即使一个简单的回车或者空格，都会影响签名并阻止 PowerShell 从中获取信息。

```
PS C:\>cd $pshome
```

```
PS C:\>notepad dotnettypes.format.pslxml
```

然后从中找出准确的类型并返回给“Get-Process”。

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

接下来完成下述步骤。

(1) 复制完整的类型名称：System.Diagnostics.Process，并粘贴。可以用键盘光标高亮选中类型名，然后按回车键复制到粘贴板。

(2) 切换到记事本，然后按 Ctrl+F 组合键打开查找窗口。

(3) 在窗口中粘贴类型名，然后单击“查找下一个”。

(4) 你能找到的第一个对象一般是“ProcessModule”，这不是进程对象。所以继续查找下一个对象，直到找到“System.Diagnostics.Process”为止，如图 10.1 所示。

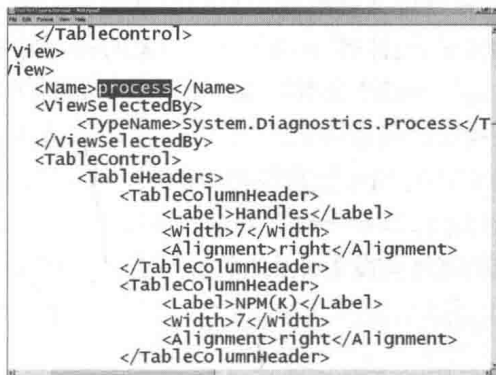


图 10.1 在 Windows 记事本中定位进程视图

你现在看到的是在记事本中以默认形式显示一个进程的管理目录。稍微向下滚动一点，可以看到表视图的定义。这可能是你期望的结果，因为你已经知道进程显示在一个多列的表中。从中可以看到熟悉的列名称，如果再往下一点点，还能发现用于定义列与属性对应关系的信息，还能看到列宽度及别名的定义等。浏览过后，关闭记事本，切记不要把信息保存到该文件中，然后返回 PowerShell。

当运行“Get-Process”，在 Shell 中会发生下面的事情。

(1) Cmdlet 把类型为 “System.Diagnostics.Process” 的对象放入管道。

(2) 在管道的末端是一个名称为 “Out-Default” 的隐藏 Cmdlet。该 Cmdlet 的作用是把需要运行的命令全部放入管道中，注意此 Cmdlet 总会存在。

(3) “Out-default” 将对象传输到 “Out-Host”，原因是 PowerShell 控制台默认把输出结果显示到机器所在的显示屏上（称为 host）。理论上，可以在 Shell 中编写把文件或打印机作为默认输出设备的脚本，但是目前为止还没听说过有人这样做。

(4) 大部分以 Out-开头的 Cmdlets 不适合用在标准对象中，而主要用于特定格式化指令上。所以当 “Out-Host” 发现标准对象时，会把它们传递给格式化系统。

(5) 格式化系统以其内部格式化的规则检查对象的类型（我们将在下面介绍）。然后用这些规则产生格式化指令，最终传回 “Out-Host”。

(6) 一旦 “Out-Host” 发现已经生成了格式化指令，就会根据该指令生成显示到屏幕上的结果。

上面提到的内容也会在你手动指定 “Out-Cmdlet” 时候发生。比如在运行 “Get-Process | Out-File procs.txt” 和 “Out-File” 时，PowerShell 会看到你发送了一些普通对象。它会把这些对象发给格式化系统，然后创建格式化指令后回传给 “Out-File”。“Out-File” 基于这些指令创建格式化后的文本文件。所以在需要把对象转换为用户可读的文本输出格式时，格式化系统就会起到作用。

在上面 5 个步骤中，PowerShell 依赖于什么格式化规则？其中第一个规则是系统会检查对象类型是否已经被预定义视图处理过。也就是我们在 “DotNetType.format.ps1xml” 中所见到的：一个针对-process 对象的预定义视图。PowerShell 中还预装了其他的 “.format.ps1xml” 文件，这些文件在 Shell 启动时会被自动加载。你也可以创建自己的预定义视图，但是这部分内容超出了本书范围。

格式化系统对特定的对象类型查找相应的预定义视图，在本例中也就是查找处理 “System.Diagnostics.Process” 对象的视图。

如果没找到对应的视图会发生什么？比如运行：

```
Get-WmiObject Win32_OperatingSystem | Gm
```

选中对象的类型名称（或至少选择 “Win32_OperatingSystem” 部分），然后尝试在其中一个 “.format.ps1xml” 文件中查找该名称。为了节省时间，我们直接告诉你找不到该类型名称。

这是格式化系统下一步要做的事情，我们也可以称之为第二个格式化规则：格式化系统会查找是否有人为该对象类型预定义默认显示属性集。这些可以在另外一个配置文件 “Types.ps1xml” 中找到。现在继续用记事本打开（记住别保存任何修改），然后使用查找功能定位关键字 “Win32_OperatingSystem”。一旦找到它之后，就可以看到 “DefaultDisplayPropertySet”，如图 10.2 所示，注意下面列出的 6 个属性。

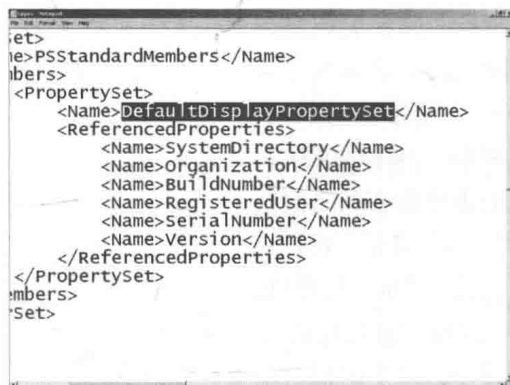


图 10.2 在记事本中定位 DefaultDisplayPropertySet

现在返回 PowerShell，然后运行：

```
Get-WmiObject Win32_OperatingSystem
```

结果是不是看起来很熟悉？仅仅显示这些属性是由于它们来自于默认的“Types.ps1xml”文件。如果格式化系统找到一个默认显示属性集，那么格式化系统会使用这些属性为下一步做准备，如果没有找到，那么下一步将考虑对象全部的属性值。

接下的分支，即格式化第三个规则——用于确定输出样式。如果格式化系统显示 4 个或以下的属性，输出结果会以表格形式展现。如果有 5 个或以上的属性，输出结果会使用列表形式。这就是“Win32_OperatingSystem”对象的结果不以表格显示的原因。它的结果有 6 列，所以以列表形式展示。其中原理就是当属性超过 4 个时，不截断输出结果的情况下，很难将输出结果正确展示在表格中。

现在你已经了解格式化的工作机制，并且明白了大部分以 Out 开头的 Cmdlets 都会自动触发格式化系统，以便找到所需的格式化指令。下面看看我们如何控制格式化系统，并且覆盖默认值。

顺便提一下，格式化系统也是导致为什么 PowerShell 有时显得会“撒谎”。例如，运行 Get-Process 并查看列头，看到名称为 PM(K) 的列了吗？这就是一个“谎言”，这是由于并没有名称为 PM(K) 的属性，但有一个名称为 PM 的属性。这里所需知道的是格式化列头仅仅是格式化列头。列头无须与底层属性名称一致。查看属性名称唯一安全的方式是使用 Get-Member。

10.3 格式化表格

在 PowerShell 中，有 4 个用于格式化的 Cmdlets。我们将介绍日常使用最多的 3 种（第 4 种会在本节结尾的“补充说明”中简要介绍）。首先是“Format-Table”，其别名为“Ft”。

如果你查看“Format-Table”的帮助文档，可以发现这个命令有很多参数。我们将演示其中最常用的几个。

- **-autoSize**——通常情况下，PowerShell 会根据窗口宽度生成表格（除非存在一个预定义视图，如针对进程的预定义视图，在该视图中定义了列宽度）。一个包含较少列的表格会在列之间留下大量空白，这样的表格不会很美观。通过添加“-autoSize”参数，可以强制结果集仅保存足够的列空间，使得表格更加紧凑，但是会使得 Shell 花费额外时间生成输出结果，这是由于 Shell 会查看所有对象，并找到每列最长行的长度。尝试为下面命令添加“-autoSize”参数，并比对不加该参数产生的输出结果。

```
Get-WmiObject Win32_BIOS | Format-Table -autoSize
```

- **-property**——该参数接收一个逗号分隔符列表，该列表包含期望显示的属性值。这些属性不区分大小写，但是 Shell 会使用你提供的参数作为列头。因此对于大小写格式化不美观的属性值，你可以使其更加美观（如使用“CPU”替代“cpu”）。另外，该参数也接受通配符，可以使用“*”代表的所有属性，或者使用“c*”标识所有以 c 开头的属性名称。但是需要注意的是，Shell 仍然只会显示可以被表格容纳的属性，而不是输出所有你指定的列。该参数是位置参数，所以可以不提供参数名称，只需要在第一个参数位置提供属性列表即可。尝试运行下面的语句（结果见图 10.3）。

```
Get-Process | Format-Table -property *
```

```
Get-Process | Format-Table -property ID,Name,Responding -autoSize
```

```
Get-Process | Format-Table * -autoSize
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
38	6	1984	4552	57	43.88	2196	conhost
31	4	796	2272	22	0.14	2508	conhost
29	4	832	2300	41	2.27	2888	conhost
32	5	976	2904	46	0.11	3236	conhost
506	13	1900	4064	48	3.78	320	csrss
213	12	7928	5892	53	54.17	372	csrss
296	30	14576	19516	143	20.83	1300	dfsrvs
122	15	2584	6188	41	0.63	1760	dfssvc
5157	7329	85720	87088	122	4.48	1356	dns
65	7	1824	4684	53	0.25	324	dwm
669	40	27176	41668	174	8.28	2100	explorer
129	9	3032	5056	38	8.95	2500	fdhost
48	6	1020	3244	25	0.02	2432	fdlauncher
0	0	0	24	0	0	0	Idle
134	14	5760	11684	68	0.08	1420	inetinfo
100	14	2988	4876	39	0.13	1464	ismserv
1332	111	34368	30308	164	67.03	484	lsass
194	11	2820	5676	30	7.55	492	lsm
308	42	52244	52348	559	11.03	1236	Microsoft.ActiveDirectory...
146	18	3228	7180	60	0.09	2576	msdtc
1793	44	473460	492088	1010	53.44	3028	powershell
545	54	128788	133564	766	224.25	3760	powershell_ise

图 10.3 创建关于进程的自动大小表格

- `-groupBy`——该参数会导致每当指定的属性值发生变化时，生成一个新的列头集合。该参数只有第一次对某个对象的特定属性排序时才能生效。示例如下。

```
Get-Service | Sort-Object Status | Format-Table -groupBy Status
```

- `-wrap`——如果 Shell 需要把列的信息截断，会在列尾带上省略号 (...) 以便标识信息被截断。该参数使得 Shell 可以封装信息，这使你的表变长，但会保留你期望显示的所有信息。下面是示例。

```
Get-Service | Format-Table Name,Status,DisplayName -autoSize -wrap
```

动手实验：你应该在 Shell 中运行上面提到的所有示例，然后尝试混合使用这些技术，实现看哪些功能可以生效，以及能够生成的输出结果。

10.4 格式化列表

有时候你所需展示的信息过多无法适应表格宽度，此时使用列表就很恰当。是时候用上“`Format-List`”了，注意你可以使用它的别名：`F1`。

该 Cmdlet 与“`Format-Table`”有一些相同的参数，包括“`-property`”。实际上，`F1` 是另一个用于展示对象属性的方式，和 `Gm` 不同。`F1` 也同样显示这些属性的值，以便你可以看到每个属性包含的信息。

```
Get-Service | F1 *
```

图 10.4 展示了输出结果的示例。我们经常使用 `F1` 作为查看对象属性的候选方案。

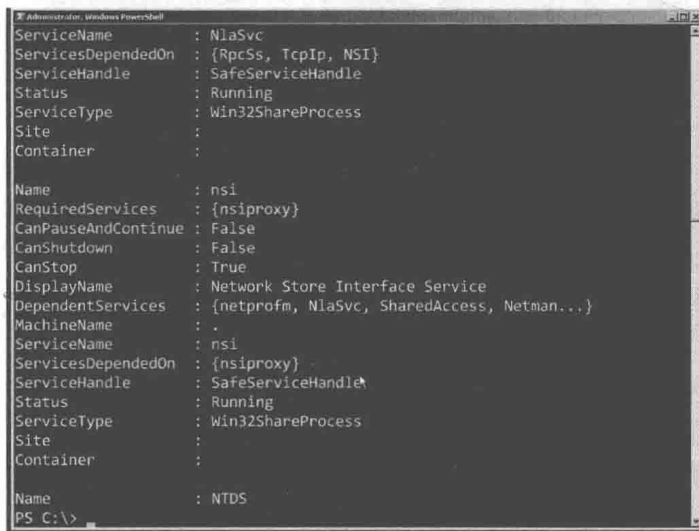


图 10.4 检查显示在列表中的服务

动手实验：查阅“`Format-List`”的帮助文档，尝试使用该命令的参数。

10.5 格式化宽列表

最后一个 Cmdlet 是 “Format-Wide” (或者别名 Fw), 用于展示一个宽列表。它仅展示一个属性的值, 所以它的 “-property” 参数仅接受一个属性名称, 而不是接受列表, 并且不接受通配符。

默认情况下, “Format-Wide” 会查找对象的 “Name” 属性, 因为 “Name” 是广泛使用的属性并且通常包含有用信息。该命令默认输出结果只有两列, 但是 “-columns” 参数可以用于指定输出更多的列。

```
Get-Process | Format-Wide name -col 4
```

图 10.5 展示了该命令结果的示例。

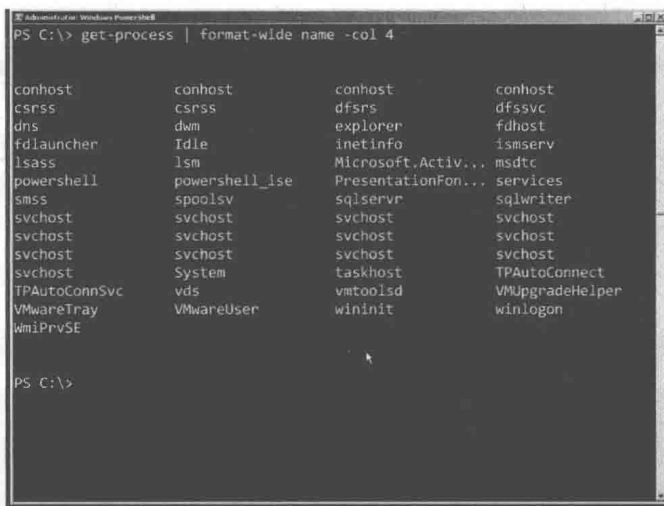


图 10.5 在一个宽列表中显示进程名称

动手实验：仔细阅读 “Format-Wide” 的帮助文档, 并尝试使用其参数。

10.6 创建自定义列与列表条目

返回前一章, 重新阅读 9.5 小节: “数据不对齐时: 自定义属性”。在该节中, 我们展示了如何使用哈希表结构添加对象的自定义属性。 “Format-Table” 与 “Format-List” 都能使用同样的结构创建自定义列或自定义表条目。

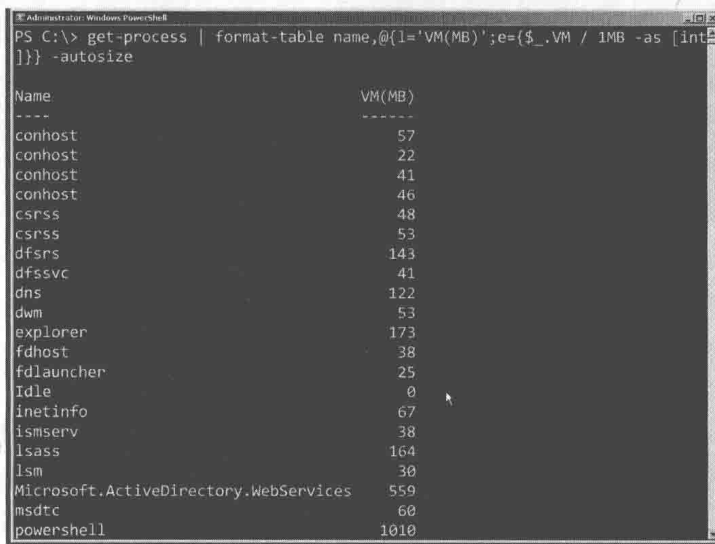
可以通过提供与属性名称不同的列头创建自定义列。

```
Get-Service |  
➤ Format-Table @{name='ServiceName';expression=  
➤ {$_.Name}},Status,DisplayName
```

甚至使用更复杂的数学表达式。

```
Get-Process |
  Format-Table Name,
  @{name='VM(MB)';expression={$_.VM / 1MB -as [int]}} -autosize
```

图 10.6 展示了前面命令的结果。其实我们做了一点小动作，用了一些之前没提到过的技术。下面我们稍微说明一下。



Name	VM(MB)
conhost	57
conhost	22
conhost	41
conhost	46
csrss	48
csrss	53
dfsrs	143
dfssvc	41
dns	122
dwm	53
explorer	173
fdhost	38
fdlauncher	25
Idle	0
inetinfo	67
ismserv	38
lsass	164
lsn	30
Microsoft.ActiveDirectory.WebServices	559
msdtc	60
powershell	1010

图 10.6 创建一个定制的结果，统计列表 MB 值

- 我们从“Get-Process”开始，相信你已经很熟悉该命令了。如果你运行“Get-Process | Fl *”，你会看到“VM”的属性是以字节为单位，虽然默认的表格视图显示并不如此。
- 我们从进程的“Name”属性开始讨论“Format-Table”。
- 接着，我们使用一个特殊的哈希表创建一个显示为“VM(MB)”的自定义列。这是以分号作为分隔符的第一部分，第二部分定义了值或表达式，对于示例来说，就是将 VM 属性的值除以 1 MB。在 PowerShell 中的斜线是除法操作。另外，PowerShell 能够识别“KB”“MB”“GB”“TB”和“PB”这些缩写，分别代表 kilobyte、megabyte、gigabyte、terabyte 和 petabyte。
- 除法运算的结果会带有小数点，这也是我们不希望看到的。“-as”操作符可以帮助我们数据结果从浮点型转换成整型（如指定[int]）。该 Shell 会根据情况向上或者向下取整，以便显示合适的结果。其最终结果是没有小数的数值。

我们这里展示了除法与取整的技巧，因为这部分内容对美观的输出结果非常有用。我们不会在本书中花更多的篇幅介绍这些操作符（虽然我们会告诉你*用于乘法，以及+和-分别用于加法与减法）。

补充说明

建议你重复运行下面的例子。

```
Get-Process |  
Format-Table Name,  
@{name='VM(MB)';expression={$_.VM / 1MB -as [int]}} -autosize
```

不过这次不要在一行中全部输入，而是按照上面的格式输入。你会发现，当你输入完第一行之后（也就是以管道符结尾），PowerShell 会出现一个提示符。因为你以管道符结尾，所以 Shell 知道你准备输入更多的命令，直到你以花括弧、引号和括号结尾为止。

如果你不想以这种“扩展输入模式”输入，按 Ctrl+C 组合键停止，并再次运行上面的示例。在这种情况下，输入第二行文本并按回车键，然后继续输入第三行，再按回车键。在这种模式下，你必须多按一次回车，以便告知 Shell 你已经完成输入。然后 Shell 会逐行按顺序运行你的输入。

与“Select-Object”不同，它的哈希表仅接受一个 Name 和 Expression 作为哈希键（虽然对于 Name 属性，可以用 N、L 和 Label；对于 Expression 属性，可以用 E）。“Format-”命令可以处理用于控制显示的额外的键字。这些关键字对于“Format-Table”尤其有效。

- **FormatString**: 指定一个格式化代码，使得数据按照指定格式显示，该参数主要用于数值型和日期型数据。可以到 MSDN 的“Formatting Types”([http://msdn.microsoft.com/en-us/library/fbxft59x\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/fbxft59x(v=vs.95).aspx)) 页中查看用于标准数值与日期格式的格式化代码，以及用于自定义数值与日期格式的格式化代码。

- **Width**: 指定列宽。

- **Alignment**: 指定列的对齐格式，可以为左对齐或者右对齐。

使用额外的键修改上面的代码，实现同样的输出结果，并更加美观。

```
Get-Process |  
Format-Table Name,  
@{name='VM(MB)';expression={$_.VM};formatstring='F2';align='right'}  
-autosize
```

现在我们并不需要使用除法，因为 PowerShell 会以两位小数并右对齐的形式格式化输出结果。

10.7 输出到文件、打印机或者主机上

一旦格式化完成某些对象，你就必须决定它的去向。

如果命令最后是 Format-开头的 Cmdlet，由 Format-开头的 Cmdlet 创建的格式化指令将会传递给“Out-Default”，然后该命令会将结果传递给“Out-Host”，最后显示结果到显示屏。

Get-Service | Format-Wide

你可以手动在上面命令中输入“Out-Host”，并以管道符连接。实际上运行了下面的语句。

Get-Service | Format-Wide | Out-Host

另外一种方式是用管道把格式化指令传递给“Out-File”或“Out-Printer”，从而将结果输出到文件或者打印机中，正如你在 10.9 小节中看到的，只有这三个以“Out-”开头的 Cmdlet 才可以跟在以“Format-”开头的 Cmdlet 后面。

请记住，“Out-Printer”和“Out-File”都有默认的输出宽度，意味着输出结果在文件或打印的结果看上去可能和显示器上看到的的一致。这些 Cmdlets 允许你使用“-width”参数控制宽度，输出你想要的表格。

10.8 输出到 GridView 中

“Out-GridView”提供了另一种形式的输出结果。但是注意，从技术角度来讲，这并不真正意义上的格式化。实际上，“Out-GridView”完全绕过了格式化子系统。它不需要调用以“Format-”开头的 Cmdlets，不生成格式化指令，不会在控制台窗口输出文本结果。“Out-GridView”不接收“Format-”Cmdlet 的输出，仅接收其他 Cmdlets 输出的对象。Out-GridView 可能无法在非 Windows 系统中生效。

图 10.7 显示了网格的样子。

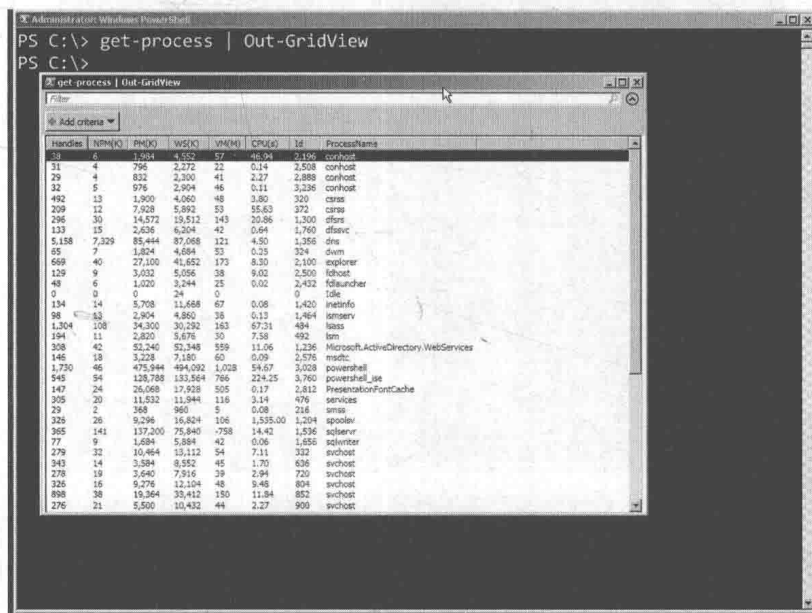


图 10.7 “Out-GridView” Cmdlets 的结果

10.9 常见误区

正如本章开头所说, PowerShell 的格式化系统对新手来说存在不少陷阱。根据过往经验,我们整理出两个主要的注意事项,希望能帮助读者更好地避开这些陷阱。

10.9.1 总是在最右边格式化

切记: format right。以“Format-”开头的 Cmdlet 应该是命令行最后一个 cmdlet。而 Out-File 与 Out-Printer 却是例外。其原因是以“Format-”开头的 Cmdlets 生成格式化指令,仅有“Out-”Cmdlet 能合理地处理这些指令。如果一个以“Format-”开头的 Cmdlet 作为命令行的结尾,指令将使用“Out-Default”(作为管道的结尾)该 cmdlet 会指向“Out-Host”,这会导致非预期的格式化。

为了演示,运行以下命令。

```
Get-Service | Format-Table | Gm
```

如图 10.8 所示,你会看到“Gm”并没有显示关于服务对象的信息,因为“Format-Table”并不输出服务对象。它消费你通过管道传输过来的服务对象,并且输出格式化指令——这正是“Gm”所看到并输出的。

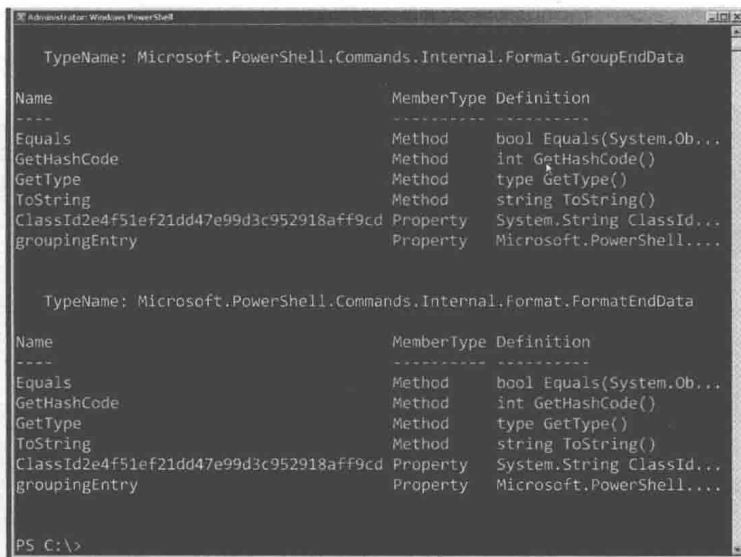


图 10.8 格式化 Cmdlets 产生的特定格式化指令,可见其易读性不高

动手实验:

```
Get-Service | Select Name,DisplayName,Status | Format-Table |  
ConvertTo-HTML | Out-File services.html
```

接着用 IE 打开 `Services.html` 文件，你可以看到让你抓狂的结果。你并没有把服务对象用管道传输到“ConvertTo-HTML”中，你只是传输了格式化指令，然后转换成 HTML。这里演示了为什么一旦使用了某个以“Format-”开头的 Cmdlet，要么把它作为命令行的最后一个 Cmdlet，要么必须出现在“Out-File”或者“Out-Printer”的前面。

同样，我们知道“Out-GridView”也不寻常（最起码针对“Out-”Cmdlet 来说），因为它不接受格式化指令，仅接受常规对象。可以用下面命令查看差异。

```
PS C:\>Get-Process | Out-GridView
PS C:\>Get-Process | Format-Table | Out-GridView
```

这就是为什么我们额外提醒“Out-File”和“Out-Printer”是仅有的需要跟在以“Format-”开头的 Cmdlet 后面的命令（技术上，“Out-Host”也可以跟在以“Format-”开头的 Cmdlet 后面，但是没有必要，因为以“Format-”开头的 Cmdlet 结尾的命令无论如何都会输出到“Out-Host”上）。

10.9.2 一次一个对象

另外一件需要避免的事就是把多种对象放入管道。格式化系统先在管道中查找第一个对象，然后使用定义格式处理该对象。如果管道包含两个或以上的对象，那么结果可能与你期望的会有不同。

比如运行：

```
Get-Process; Get-Service
```

其中分号允许我们把两个命令合并在一个命令行中，而不是把第一个命令的输出并以管道形式传入第二个命令。这意味着两个命令单独运行，但是会把它们的输出结果传到相同的管道中。如果你动手运行或者查看图 10.9，会看到第一个命令的输出是合理的，但是当显示服务对象时，输出结果会变成另一个格式，而不是使用相同的表格，此时 PowerShell 会使用列表显示。PowerShell 的格式化系统并不是被设计用于接收不同类型的对象，合并输出结果同时使得结果美观。

如果你希望将来自不同地方的两个信息以同一种格式输出，那该怎么办？你当然可以这么做，格式化系统能够以非常优雅的方式实现这点。但这是高级主题，本书并不涉及。

补充说明

技术上而言，格式化系统可以处理多种类型的对象——只要你告知它处理方式。运行“`Dir | Gm`”，你可以发现管道包含了“DirectoryInfo”和“FileInfo”对象（Gm 可以与包含不同类型对象的管道结合使用，并显示所有对象的成员信息）。当仅运行 Dir 时，输出结果非常清晰。这是因为微软已经对“DirectoryInfo”和“FileInfo”对象提供了预定义的自定义格式化实体，该格式化由“Format-Custom”完成。

“Format-Custom”主要用于展示多种预定义视图。技术上,你可以自己创建预定义视图,但是所需的 XML 语法相对复杂,并且目前没有文档支持,所以当前仅能使用微软提供的定制视图。

微软的定制视图被广泛使用。例如,PowerShell 的帮助系统以对象形式储存,你所看到已格式化的帮助文件是将这些对象传递给自定义视图处理后的结果。

```

122      11      2532      5716      55      0.20      1412 taskhost
120      11      2528      7348      73 1,048.08 2904 TPAutoConnect
131      11      3140      6988      55      58.56      2292 TPAutoConnSvc
135      16      2600      7612      45      0.83      2344 vds
244      17      6300      11096      89      42.67      1676 vmtoolsd
87       9       2796      6608      41      0.17      1888 VMUpgradeHelper
73       10      3156      6256      76      0.61      2748 VMwareTray
221      17      8856      16848      118      6.95      2512 VMwareUser
79       10      1304      3984      47      0.14      380 wininit
92       7       1364      4444      30      0.09      412 winlogon
277      15      7364      12956      52      4.19      3436 WmiPrvSE

Status      : Running
Name        : ADWS
DisplayName  : Active Directory Web Services

Status      : Stopped
Name        : AeLookupSvc
DisplayName  : Application Experience

Status      : Stopped
Name        : ALG
DisplayName  : Application Layer Gateway Service
  
```

图 10.9 把两个不同类型的对象同时放入管道会引起 PowerShell 格式化系统混乱

10.10 动手实验

注意: 本实验需要 PowerShell v3 或以上版本。

尝试独立完成下面任务:

1. 显示一个仅包含进程名称、ID 的表格,无论这些进程是否对 Windows 有响应(“Responding”属性提供该信息)。尽可能使这些信息横向填满整个窗口,但不要截断任何信息。

2. 显示一个包含进程名称、ID 的表格。表中的列还要包含虚拟内存和物理内存的使用情况,以 MB 为标识单位。

3. 在 Windows 中使用“Get-EventLog”显示一个可用事件日志的列表(提示:你需要查看帮助文档,从而知道该使用哪个参数)。并把这些信息格式化成表,日志需要显示 Name 与保留期限,分别在表头以“LogName”和“RetDays”显示。

4. 显示一个服务列表,处于运行与停止状态的服务分别显示在不同表格中,处于运行状态的服务在第一个(提示:你可能需要使用“-groupBy”参数)。

5. 将 C 盘根目录下所有的目录以四列宽的列表形式显示。
6. 以列表形式显示 C:\Windows 目录中所有的.exe 文件的名称、版本信息以及文件大小。PowerShell 使用 length 属性, 但你的输出结果应该显示为 Size。

10.11 进一步学习

现在是实现格式化系统的好时机。尝试使用三个以“Format-”开头的主要 Cmdlets 创建不同格式的输出。在下一章将频繁要求你使用特定的格式化形式, 所以你需要记住在这一章频繁使用的参数。

10.12 动手实验答案

1. `get-process | format-table Name,ID,Responding -autosize -Wrap`
2. `get-process | format-table Name,ID,
@{l='VirtualMB';e={$_.vm/1mb}},
@{l='PhysicalMB';e={$_.workingset/1MB}} -autosize`
3. `Get-EventLog -List | Format-Table
@{l='LogName';e={$_.LogDisplayname}},
@{l='RetDays';e={$_.MinimumRetentionDays}} -autosize`
4. `Get-Service | sort Status -descending | format-table -GroupBy Status`
5. `Dir c:\ -directory | format-wide -column 4`
6. `dir c:\windows*.exe |
Format-list Name,VersionInfo,@{Name= "Size";Expression={$_.length}}`