

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 可以用于产生对象，并且这些对象通常含有比 PowerShell 默认显示更多的属性。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”去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创建你自己的命令的输出格式。

10.1 格式化：让输出更加美观

因为 PowerShell 并不是完全成熟的用于生成管理报表的工具，所以读者不要因为前面的例子而产生误解。但是 PowerShell 的确能很好地收集计算机的信息，并以定制的格式输出结果。用于输出定制格式的方式称为格式化。

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）。但是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们是如何工作的，并且为什么要这样。本章不打算演示什么新命令，而是解释整个系统是如何工作的，并且你如何与它交互，另外还有需要注意的限制。

10.2 默认格式

现在执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，它们并不是非常符合常规的属性名。取而代之的是一个固定的宽度、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.ps1xml”文件。注意，别在这个文件中保存任何变更信息。这个文件带有数字签名，即使一个简单的回车或者空格号，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.ps1xml
```

然后从中找出准确的类型并返回给“Get-Process”：

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

接下来完成下面的步骤：

(1) 复制完整的类型名：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。这可能是你希望见到的结果，因为你已经知道如何把进程显示在一个多列的表中。从中可以看到熟悉的列名，如果再往下一点点，还能发现特定表中每列的展示属性，还有列宽及别名的定义等。浏览过后，关闭记事本，切记不要把信息保存到这个文件中，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为“Out-Default”的隐藏的 Cmdlet。这个 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”把对象传输到“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，可以写一个 Shell 把文件或打印机作为默认输出设备，但是目前为止还没听说过有人这样做。
- (4) 大部分“Out-Cmdlets”不适合用在普通对象中，而主要用于特定格式化指令上。所以当“Out-Host”看到那些普通对象时，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，最终传输回“Out-Host”。
- (6) 一旦“Out-Host”发现已经生成了格式化指令，就会根据这个指令产生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”的时候发生。比如运行“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成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 12 步中，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。也就是我们见到的“DotNetType.format.ps1xml”：针对进程的对象。PowerShell 中预装了其他的“.format.ps1xml”文件，在 Shell 启动时自动加载。你也可以创建自己的预定义视图，但是这部分超出了本书范围。

格式化系统对特定对象的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
```

选中对象类型的名字（最少选择“Win32_OperatingSystem”部分），然后尝试在其中一个“.format.ps1xml”文件中查找它。为了节省时间，我们直接告诉你找不到的。

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：格式化系统寻找是否有针对这个对象类型的“default display property set”。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“Win32_OperatingSystem”。一旦找到它之后，向下滚动一点点，就可以看到“DefaultDisplayPropertySet”，如图 10.2 所示，注意下面列出的 6 个属性。

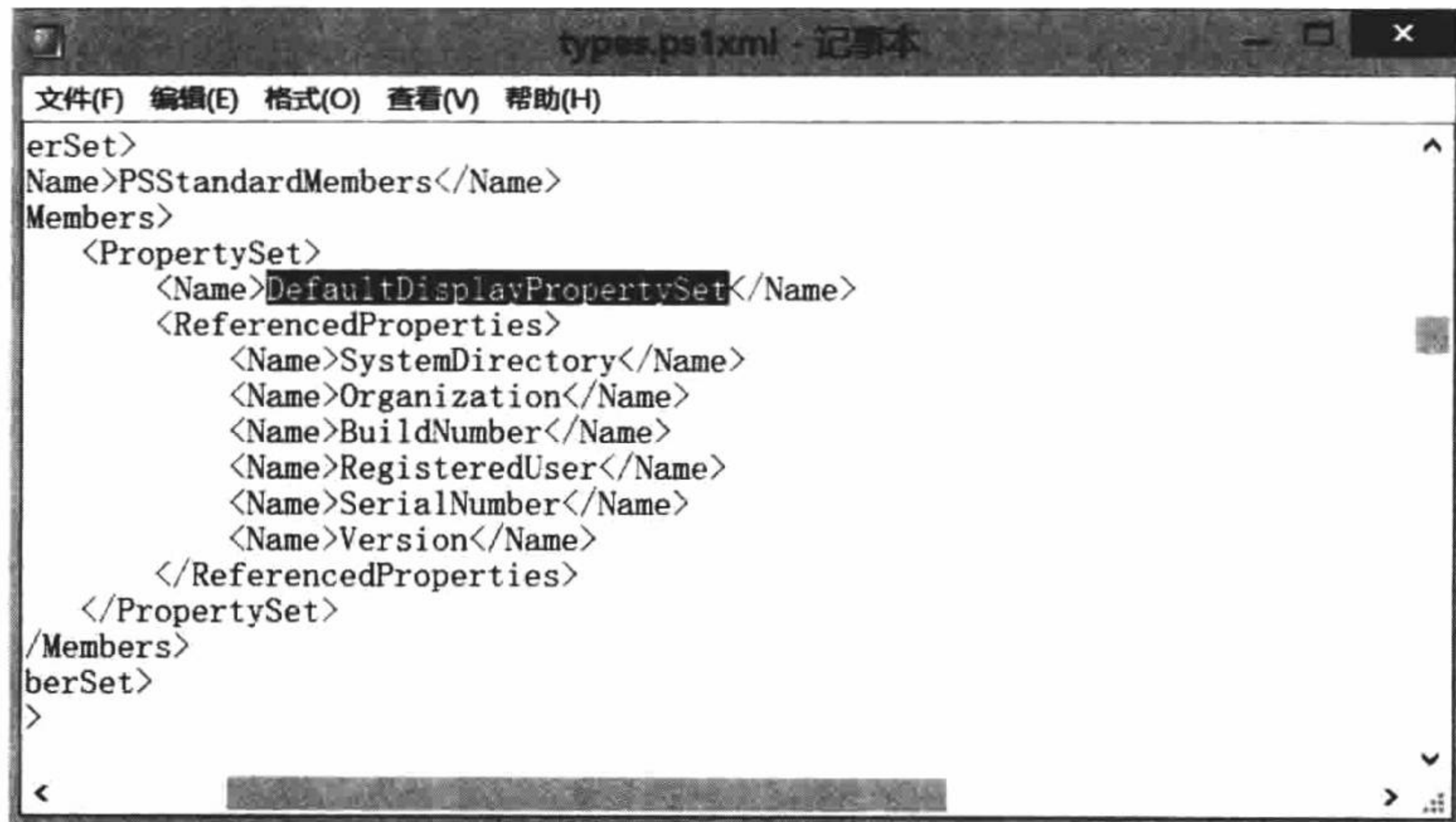


图 10.2 在记事本中定位 DefaultDisplayPropertySet

现在返回 PowerShell，然后运行：

```
Get-WmiObject Win32_OperatingSystem
```

结果是不是看起来很熟悉？这些属性单独看起来的确如此，因为它们来自于默认的“Types.ps1xml”文件。如果格式化系统找到一个“default display property set”，会把这个属性集用于下一步的决策。如果没有找到，那么下一步的决策将考虑所有对象的属性值。

接下来是决策，即格式化第三个规则——用于决定输出的样式。如果格式化系统将显示 4 个或以下的属性，将决定以表格形式展现。如果有 5 个或以上的属性，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，可能不能很好地展示到一个未经处理的实时表格中。

现在你已经了解格式化是如何工作的，并且明白了大部分“Out-Cmdlets”会自动触发格式化系统，以便能按需找到格式化指令。下面看看我们如何控制格式化系统，并且覆盖默认值。

10.3 格式化表格

在 PowerShell 中，有 4 种格式化的 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 会根据你的屏幕生成和填充表格（除非存在一个预定义视图，如针对进程的）。这意味着结果集会只有少量的列，并且列与列之间的空隙较大，并不总是引人注目。通过使用“-autoSize”，可以强制结果集仅保存足够的列空间，使得表格更加紧凑，但是会耗费少量的时间让 Shell 产生输出，因为对于每个对象都需要在格式化的过程中搜寻每个列的最大值。尝试在下面的语句中对比是否有“-autoSize”参数的结果：

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

- **-property**——该参数接收一个以逗号分割的希望包含在结果表格中的属性列表。这些属性是不区分大小写的，但是 Shell 会使用你的输出作为列头。如果你希望输出格式以期望的形式展现，需要规定好名字的格式（如使用“CPU”替代“cpu”）。另外，这个参数也接受通配符，可以使用“*”替代的所有属性，或者使用如“c*”标识所有以 c 开头的属性名。但是需要注意的是，这个 Shell 依旧使用仅能填充到一个表格的属性作为展示，并不是你指定的都会输出。这个参数是依赖位置的，所以可以不输入参数名，只需要在第一个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）：

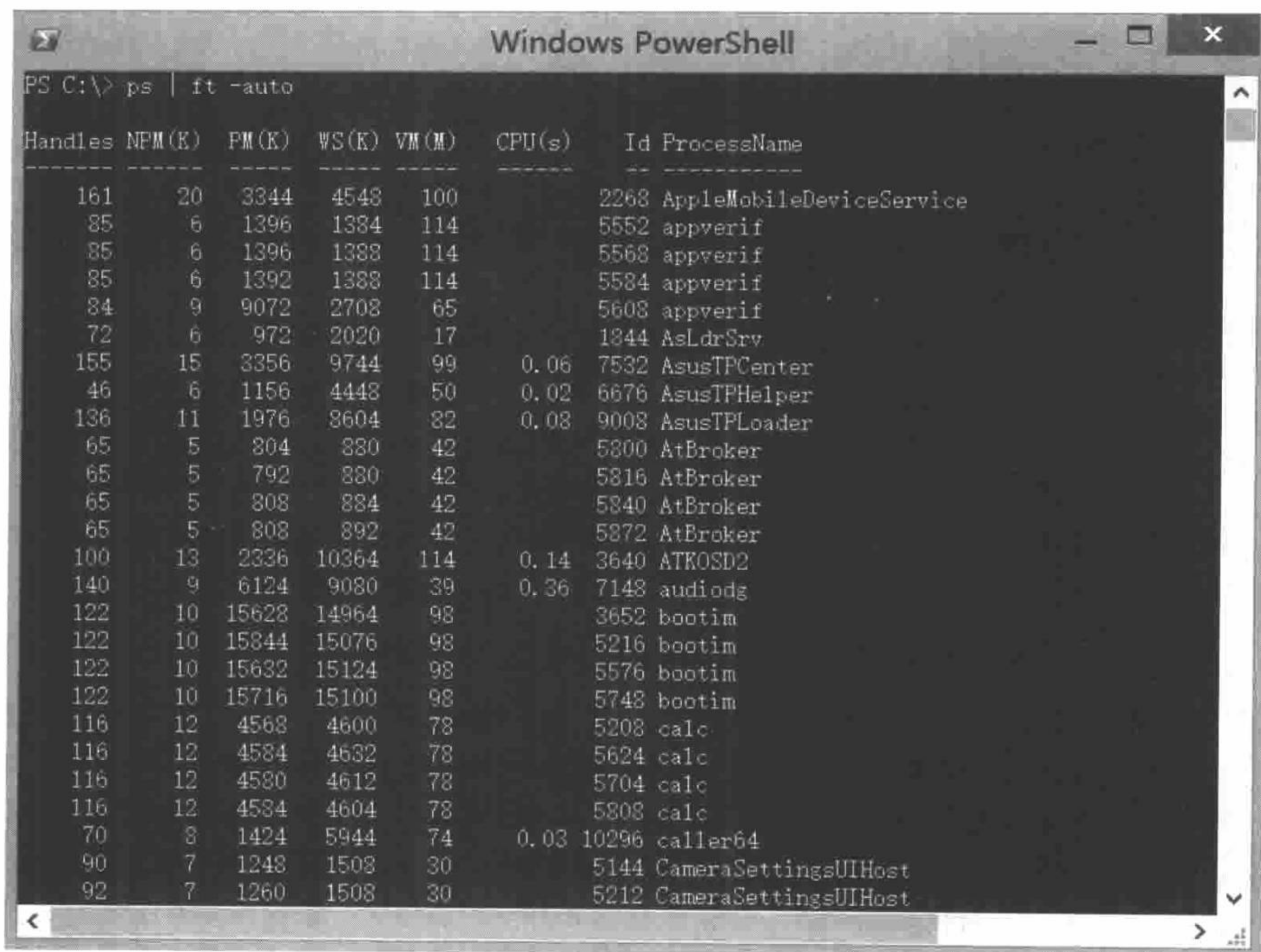
```
Get-Process | Format-Table -property *
Get-Process | Format-Table -property ID,Name,Responding -autoSize
Get-Process | Format-Table * -autoSize
```

- **-groupBy**——每当指定的属性值变更时，创建一个具有新列头的结果集。这个参数只在你第一次对具有相同属性的对象进行排序时工作良好。可以通过例子去理解参数是如何工作的：

```
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 (...) 以便标识信息被截断。这个参数能使 Shell 把信息收起，会使你的表变长，但是当你需要查看的时候却很有用。下面是例子：

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



Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
161	20	3344	4548	100		2268	AppleMobileDeviceService
85	6	1396	1384	114		5552	appverif
85	6	1396	1388	114		5568	appverif
85	6	1392	1388	114		5584	appverif
84	9	9072	2708	65		5608	appverif
72	6	972	2020	17		1844	AsLdrSrv
155	15	3356	9744	99	0.06	7532	AsusTPCenter
46	6	1156	4448	50	0.02	6676	AsusTPHelper
136	11	1976	8604	82	0.08	9008	AsusTPLoader
65	5	804	880	42		5800	AtBroker
65	5	792	880	42		5816	AtBroker
65	5	808	884	42		5840	AtBroker
65	5	808	892	42		5872	AtBroker
100	13	2336	10364	114	0.14	3640	ATKOSD2
140	9	6124	9080	39	0.36	7148	audiodg
122	10	15628	14964	98		3652	bootim
122	10	15844	15076	98		5216	bootim
122	10	15632	15124	98		5576	bootim
122	10	15716	15100	98		5748	bootim
116	12	4568	4600	78		5208	calc
116	12	4584	4632	78		5624	calc
116	12	4580	4612	78		5704	calc
116	12	4584	4604	78		5808	calc
70	8	1424	5944	74	0.03	10296	caller64
90	7	1248	1508	30		5144	CameraSettingsUIHost
92	7	1260	1508	30		5212	CameraSettingsUIHost

图 10.3 创建关于进程的自动大小表格

动手实验：你应该运行上面提到的所有 Shell，然后尝试通过混合这些技术，体会它们是如何运作和如何排序的。

10.4 格式化列表

有时候你需要水平地把数据展现到一个表中，此时使用列表就很有用。“Format-List”是时候用上了，注意你可以使用其别名：Fl。

这个 Cmdlet 也支持一些类似的参数，如“Format-Table”，包括“-property”。实际上，Fl 是另外一个展示对象属性的方法，和 Gm 不一样。Fl 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 | Fl *
```

图 10.4 展示了命令的结果。我们经常使用 Fl 作为发现对象属性的候选方案。

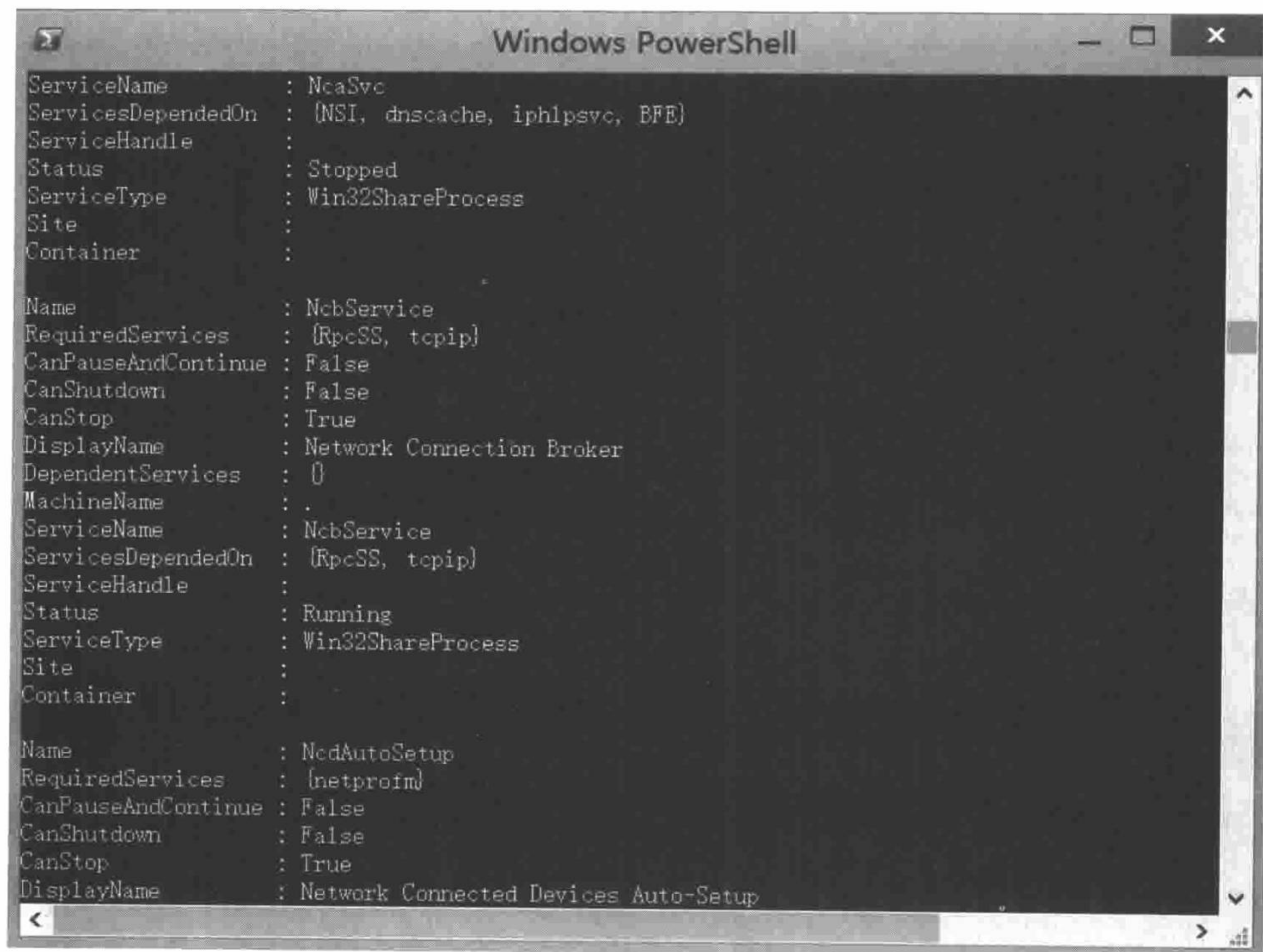


图 10.4 检查显示在列表中的服务

动手实验：查阅“Format-List”的帮助文档，并尝试体会它们参数的异同。

10.5 宽度的格式化

我们将展示的最后一个是“Format-Wide”（或者别名 Fw），用于展示一个宽列表。它仅展示一个单独属性的值，所以它的“-property”参数仅接受一个属性名，而不是列表，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。默认显示只有两列，但是“-columns”参数可以用于指定更多的列：

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

图 10.5 展示了其结果。



图 10.5 在一个宽列表中显示进程名

动手实验：仔细阅读“Format-Wide”的帮助文档，并且动手尝试它的每个参数并检查结果。

10.6 定制列和列表记录

返回前一章，重新阅读 9.5 节，“参数不对齐时，请自定义属性”。在该节中，我们展示了如何使用哈希表结构来添加对象的自定义属性。“Format-Table”和“Format-List”能使用这些相同的结构创建自定义表列或自定义表记录。

可以通过提供不同于属性名的列头来自定义显示结果：

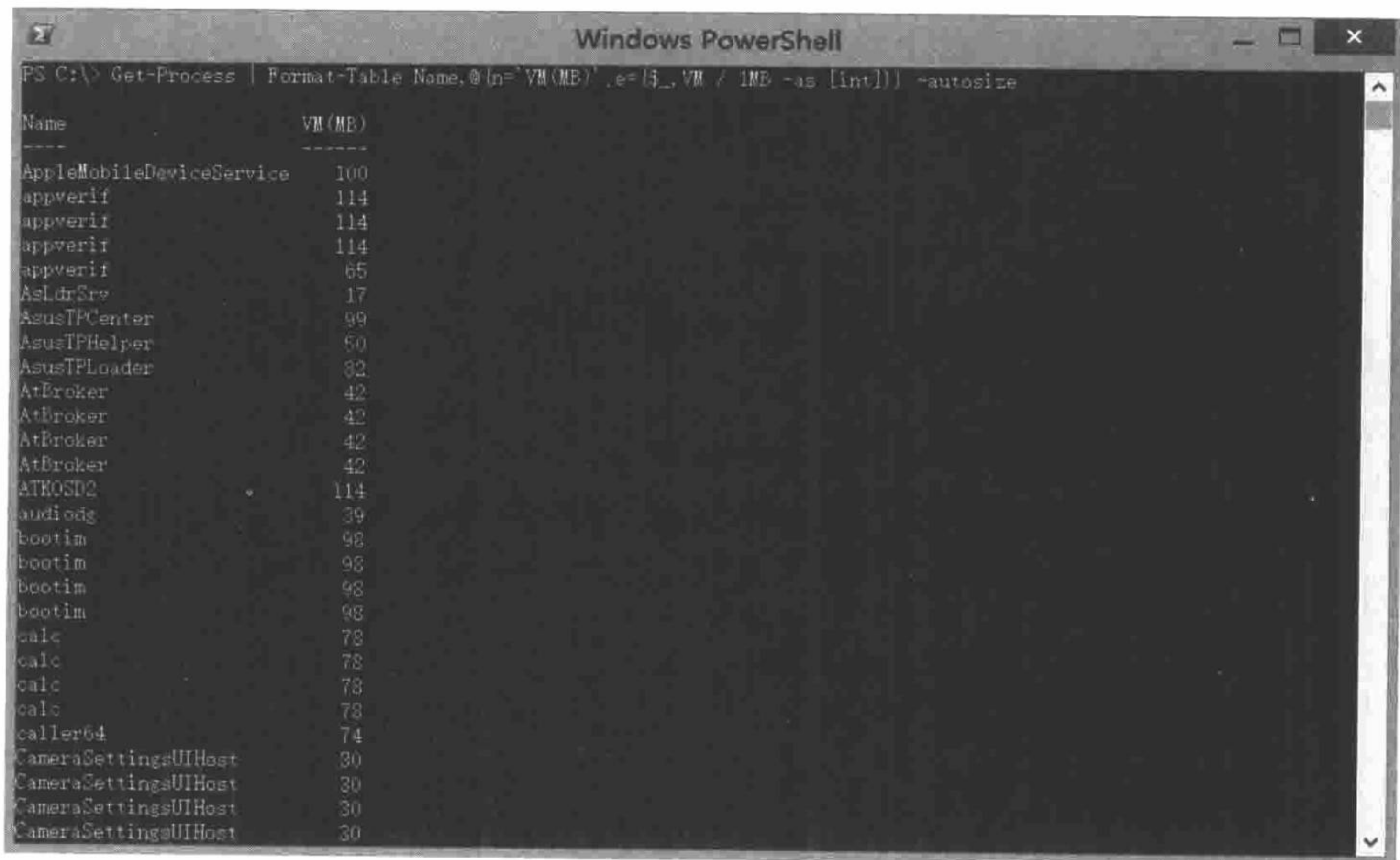
```
Get-Service |
Format-Table @{n='ServiceName';e={$_.Name}},Status,DisplayName
```

甚至使用更复杂的数学表达式来替代：


```
Get-Process |
Format-Table Name,
@{n='VM(MB)';e={$_.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。

- 我们从“Get-Process”开始，相信你已经很熟悉这个命令。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”。



```
PS C:\> Get-Process | Format-Table Name,@{n='VM(MB)';e={$_.VM / 1MB -as [int]}} -autosize
```

Name	VM(MB)
AppleMobileDeviceService	100
appverif	114
appverif	114
appverif	114
appverif	114
appverif	65
AsLdrSrv	17
AsusTPCenter	99
AsusTPHelper	60
AsusTPLoader	82
AtBroker	42
AtBroker	42
AtBroker	42
AtBroker	42
ATKOSD2	114
audioads	39
bootim	98
bootim	98
bootim	98
bootim	98
calc	78
calc	78
calc	78
calc	78
caller64	74
CameraSettingsUIHost	30
CameraSettingsUIHost	30
CameraSettingsUIHost	30
CameraSettingsUIHost	30

图 10.6 创建一个定制的结果，统计表列 MB 值

- 接着，我们创建一个以“VM(MB)”为标签的列，其值或者表达式是针对对象的常规 VM 属性并且除以 1 MB。在 PowerShell 中的斜线是除法操作。另外，“KB”“MB”“GB”“TB”和“PB”缩写分别代表 kilobyte、megabyte、gigabyte、terabyte 和 petabyte。
- 除法运算的结果会自带小数点组件，“-as”操作符可以帮助我们帮数据结果从浮点型转换成整型（如指定[int]）。这个 Shell 会适当地向上或者向下取整，以便显示适当的结果。其最终结果是没有小数的数值。

补充说明

建议你重复执行下面的例子：

```
Get-Process |  
Format-Table Name,  
@{n='VM(MB)';e={$_.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 组合键来忽略。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。你必须在这种模式下在一个空行中按最少一次回车键，以便告知 Shell 你已经完成输入。然后 Shell 会逐行顺序执行你的输入。

在这里提到除法运算及其数据类型修改的技巧，是因为在输出美观的结果时非常有用。我们不打算在本书中花费过多时间在这些操作符中（仅仅介绍“*”代表乘法，“+/-”分别代表加减运算）。

和“Select-Object”不一样，它的哈希表仅接受一个名字和表达式键（对于名字，可以为 N、L 和 Label；对于表达式，可以接受 E）。为了可视化展示，“Format-”命令可以接受额外的关键字。这些关键字对于“Format-Table”十分有效。

- **FormatString**：指定一个格式化代码，让结果根据这个代码格式化，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/26etazsy.aspx>)页中查看标准数值型和日期型格式的可用代码。另外，这里还包含了自定义的格式。
- **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。

使用这些关键字修改上面的代码，能使结果更加易读和美观。

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

现在我们并不需要使用除法，因为 PowerShell 会以两个小数位并右对齐的形式格式化结果。

10.7 输出到文件、打印机或者主机上

一旦对象被格式化，你必须决定结果的去向。

如果命令以“Format-Cmdlet”结束，格式化指令将按“Format-Cmdlet”的“Out-Default”创建，也就是以“Out-Host”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。实际上运行了下面语句：

```
Get-Service | Format-Wide | Out-Host
```

另外一种方式是用管道把格式化指令的“Out-File”或“Out-Printer”定位到文件或者打印机中，比如从“常见的困惑”中看到，只有这三个“Out-” Cmdlet 才可以跟在“Format-” Cmdlet 后面。

请记住，“Out-Printer”和“Out-File”都有默认的输出宽度，意味着它们的结果宽度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 另外一个输出：网格

“Out-GridView”提供了另一种有用的输出功能。但是注意，这并不是技术上的格式化。实际上，“Out-GridView”完全绕过了格式化子系统。它不需要调用“Format-”Cmdlets，不产生格式化指令，没有文本结果输出到控制台窗口。“Out-GridView”不接收“Format-” Cmdlet 的输出，仅接收其他 Cmdlets 输出的对象。

图 10.7 显示了网格的样子。

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
161	20	3344	4548	100		2,268	AppleMobileDeviceService
85	6	1396	1384	114		5,552	appverif
85	6	1396	1388	114		5,568	appverif
85	6	1392	1388	114		5,584	appverif
84	9	9072	2708	65		5,608	appverif
72	6	972	2020	17		1,844	AsLdrSrv
155	15	3356	9744	99	0.06	7,532	AsusTPCenter
46	6	1156	4448	50	0.02	6,676	AsusTPHelper
136	11	1976	8604	82	0.08	9,008	AsusTPLoader
65	5	804	880	42		5,800	AtBroker
65	5	792	880	42		5,816	AtBroker
65	5	808	884	42		5,840	AtBroker
65	5	808	892	42		5,872	AtBroker
100	13	2336	10364	114	0.14	3,640	ATKOSD2
140	9	6128	9080	39	0.44	7,148	audiodg
122	10	15628	14964	98		3,652	bootim
122	10	15844	15076	98		5,216	bootim

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 总是以右对齐来格式化

切记：format right。你的“Format-” Cmdlet 应该是“Out-File”或者“Out-Printer”作为仅有表达式时的命令行的最后一个命令。其原因是“Format-” Cmdlets 产生格式化指令，仅有“Out-” Cmdlet 能合理地处理这些指令。如果一个“Format-” Cmdlet 作为命令行的结尾，指令将使用“Out-Default”（总为管道的结尾）即指向“Out-Host”，这会导致非预期的格式化。

为了演示，执行以下命令：

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

你会看到如图 10.8 所示，“Gm”没有显示你希望的服务对象的信息，因为“Format-Table” Cmdlet 并不输出服务对象。它处理掉你通过管道传输的服务对象，并且输出格式化指令——这正如你看到的“Gm”显示的结果。



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“`Format-`” 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`”也是不寻常的（最起码针对“`Out-`” Cmdlet 来说），因为它不接受格式化指令且仅接受常规对象。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“`Format-`” Cmdlet 后面的命令（技术上，“`Out-Host`”也可以跟在“`Format-`” Cmdlet 后面，但是没有必要，因为以“`Format-`” 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，然后使用定义的格式处理这个对象。如果管道包含两个或以上的对象，那么结果可能不是你想要的。

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以管道形式传入第二个命令。这意味着两个命令是单独运行的，但是会把它们的输出传到相同的管道中。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 的格式化系统并不用于把多个对象和结果按你期望的形式合并。

那么如何把两个结果集以单一格式显示呢？此时可以使用本书没有提到的一些高级话题来处理这个需求，从而让格式化系统能合理地美化结果。

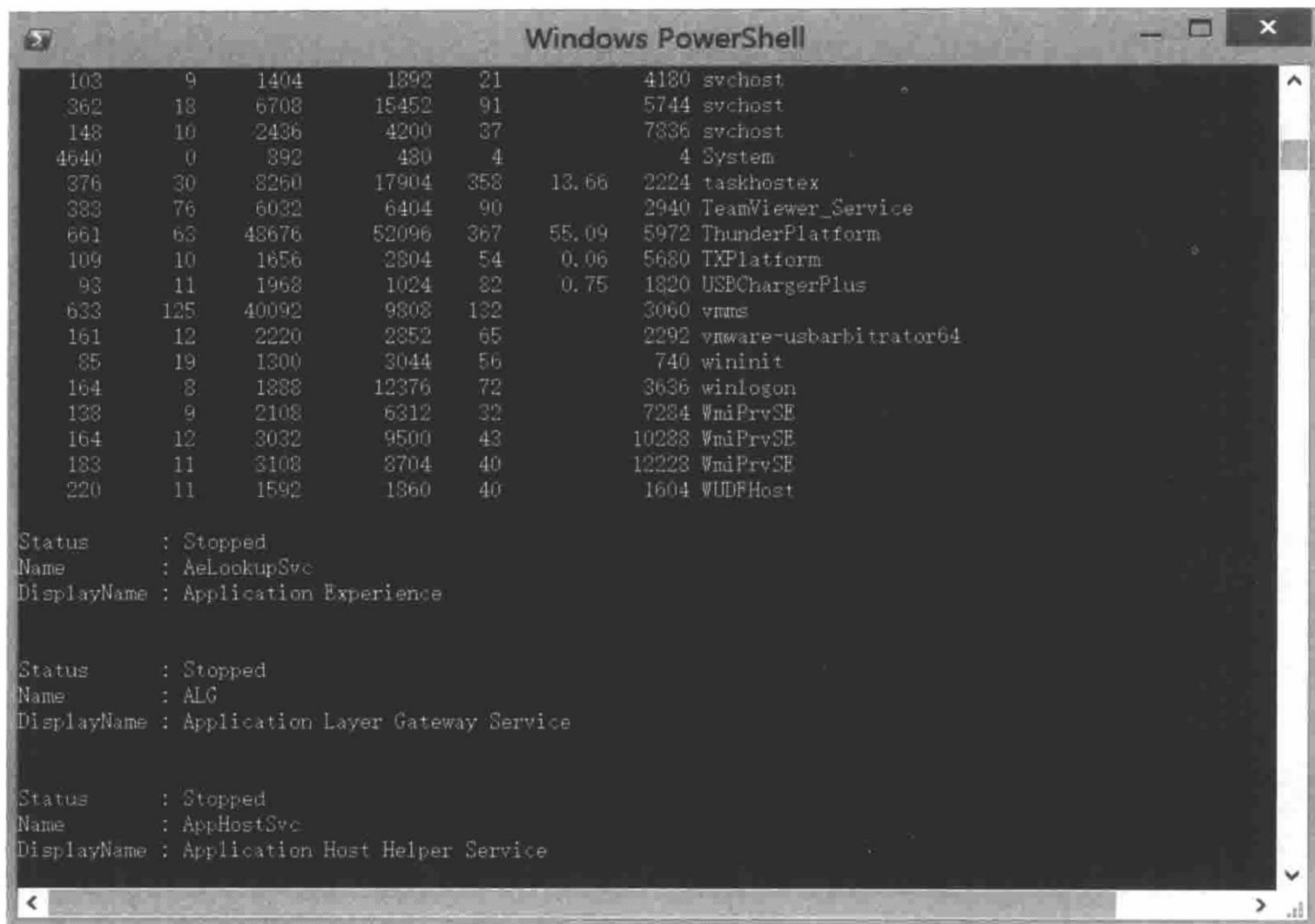


图 10.9 把两个不同类型的对象同时放入管道会引起 PowerShell 格式化系统混乱

补充说明

技术上而言，格式化系统可以处理多个对象的类型——如果你已经告知它如何处理。运行“Dir | Gm”，你可以发现管道包含了“DirectoryInfo”和“FileInfo”对象（Gm 可以在多个对象类型的管道中正常运行，并且以统一格式显示它们）。当你仅运行 Dir 时，输出结果非常清晰。这是因为微软已经对“DirectoryInfo”和“FileInfo”对象进行了格式化系统预定义，并且由“Format-Custom” Cmdlet 完成。

“Format-Custom”主要用于展示不同的预定义视图。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，并且目前未知，并没有公开，所以当前仅能使用微软提供的定制视图。

微软的定制视图的确很有用。从 PowerShell 的帮助信息里面可以找到这些对象，也可以从屏幕中看到对应的格式化帮助文件。

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. 显示一个表格，包含进程名、ID，不管这些进程是否对 Windows 响应（“Responding”属性中能找到这些信息）。尽可能使这些信息横向填满整个窗口，但不要使任何信息截断。
2. 显示一个表格，包含进程名、ID。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. 使用“Get-EventLog”显示所有可用事件日志的列表（提示：你需要查看帮助文档，以便找到能完成这个任务的信息）。并把这些信息格式化成一个表，日志需要显示名字和保留期限，分别以“LogName”和“RetDays”表示。
4. 显示一个关于服务的列表，针对服务的正在运行和结束分别显示。而正在运行的服务需要优先显示（提示：你可能需要使用“-groupBy”参数）。

10.11 进一步学习

这是针对格式化系统实验的好时机。尝试使用三个主要的“Format-” 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，所以你需要在这一章中锻炼相关技能，并且记好本章中的常用参数。