第 11 章 过滤和比较

到目前为止，我们使用 Shell 向你展示了不同类型的输出：所有进程、所有服务、所有事件日志条数、所有补丁。但是这些类型的输出并不总是你想要的结果。通常你会想要将结果范围缩小到你感兴趣的几个项。你将在本章学会这部分知识。

11.1 只获取必要的内容

Shell 提供了两种方式缩小结果集，它们都被归结为过滤。第一种方式：尝试指定 Cmdlet 命令只检索指定的内容。第二种方式：采用迭代的方法，通过第一个 Cmdlet 获得所有结果，并使用第二个 Cmdlet 过滤掉不想要的东西。

按道理，应该使用第一种方式：我们称之为尽可能提前过滤。这就像告诉 Shell 你要的是什么一样简单。例如，使用 Get-Service，你可以告诉它你想要的服务名称。

```
Get-Service -name e*, *s*
```

如果你想让 Get-Service 只返回正在运行的服务，而不考虑它们的服务名称，该 Cmdlet 就无法做到这一点，因为它没有提供用于设定该部分信息的相关的参数。

同理，如果你使用微软的活动目录模块，所有以 Get-开始的 Cmdlets 都提供了 -filter 参数。通过-filter *，你可以获取所有对象。我们不建议这样使用，因为加载这些对象将增加域控制器压力。你可以指定下面的过滤条件，就能很好表示出你所希望的是什么。

```
Get-ADComputer -filter "Name -like '*DC'"
```

再者，上述技能的优势在于该 Cmdlet 只获取匹配的对象。我们称之为左过滤技术。

11.2 左过滤

“左过滤”意味着尽可能把过滤条件放置在左侧或靠近命令行的开始部分。越早过滤不需要的对象，就越能减轻其他 Cmdlets 命令的工作，并且能减少不必要的信息通过网络传输到你的电脑。

左过滤技术的缺点是每个Cmdlet都可以通过自己的方式指定过滤，并且每个Cmdlet都会有不同的过滤方式。例如 Get-Service，你只能通过 Name 属性过滤服务。但是使用 Get-ADComputer，你可以根据 computer 对象可能存在的任何活动目录属性进行过滤。在有效使用左过滤技术之前，你需要学习不同 Cmdlet 的各种操作。这可能意味着学习的道路有些崎岖，但可以得到更好的性能。

当无法通过一个 Cmdlet 就可以完成你所需的所有过滤时，你可以使用一个叫作 Where-Object（它的别名为 Where）的核心 PowerShell 命令。这是一个通用的语法。当需要检索的时候，使用它过滤任何类型的对象，并把它传入管道。

为了使用 Where-Object，需要学会告诉 Shell 如何过滤出你想要的信息，这还包括使用 Shell 的比较操作符。有趣的是，一些左过滤技术中使用了相同的比较操作符，如活动目录模块下以 Get-开头的命令的-filter 参数，这就是一箭双雕。但是有些 Cmdlet 命令（如 Get-WmiObject，我们将在后面的章节中讨论）使用了完全不同的过滤和比较方式，当我们讨论这些 Cmdlet 命令时再做介绍。

11.3 使用比较操作符

在计算机中，比较总是涉及两个对象或者两个值来并测试它们彼此之间的关系。可能是测试它们是否相等或者是否其中一个比另外一个大，或者它们是否匹配某个文本表达式。这就需要使用比较操作符完成对关系的测试。测试的结果总是返回一个布尔值：true 或 false。换句话说，测试结果可能满足你指定的条件，可能不满足。

PowerShell 使用如下比较操作符。请注意，当比较文本字符串时会忽略大小写。大写字母与小写字母等价。

- -eq——相等，例如 5 -eq 5（返回 true）或者"hello" -eq "help"（返回 false）。
- -ne——不等于，例如 10 -ne 5（返回 true）或者"help" -ne "help"（返回 false，因为它们实际上相等的，这里测试它们是否不相等）。
- -ge 和-le——大于等于，小于等于，例如 10 -ge 5（返回 true）或者 Get-Date -le '2012-12-02'（这取决于你运行该命令的时间，这意味着可以比较日期）。

- -gt 和-lt——大于和小于，例如 10 -lt 10（返回 false）或者 100 -gt 10（返回 true）。

对于字符串的比较，如果需要区分大小写，可以使用下面的集合：-ceq，-cne，-cgt，-clt，-cge，-cle。

如果想一次比较多个对象，可以使用布尔运算符-and 和-or。通常在每个子表达式两边加上圆括号，使得表达式更容易阅读。

- (5 -gt 10) -and (10 -gt 100)返回 false，因为一个或两个子表达式返回值为 false。
- (5 -gt 10) -or (10 -lt 100)返回 true，因为最后一个子表达式返回值为 true。

另外，布尔值-not 对 true 和 false 取反。在处理一个变量或者已经包含 true 或 false 的属性时，这可能会有用。而你想测试相反的条件。例如，需要测试一个进程是否没有响应，可以这样做（使用$_作为进程对象的容器）：

```
$_.Responding -eq $False
```

Windows PowerShell 定义了$False 和$True 表示 false 和 true 的布尔值。另外一种书写方式如下。

```
-not $_.Responding
```

因为 Responding 通常包含 true 和 false，-not 使得 false 取反变为 true。如果进程没有响应，意味着 Responding 返回 false。然而上面的比较却返回 true，这就暗示着该进程“没有响应”。我们更喜欢使用第二种方式，因为在英语的阅读习惯中，它更接近我们的测试内容：“我想看看这个进程是否没有响应”。有些时候，你可以看到-not 运算符简写为感叹号（!）。

当你需要比较文本字符串时，还有其他几个有用的比较运算符。

- -like 接受*作为通配符，所以可以比较："Hello" -like "*ll*"（返回 true）。它的反义运算符为-notlike。它们不区分大小写。区分大小写可以使用-clike 和-cnotlike。
- -match 用于文本字符串与正则表达式进行比较。-notmatch 是个逻辑上的反义词。并且正如你所想，-cmatch 和-cnotmatch 提供了区分大小写的语法。正则表达式超出了本书的讨论范围。

Shell 的好处是你可以在命令行运行上面几乎所有的测试（除了前面提到的$_占位符，它不能独立运行，但是你可以在下一节看到它是如何运行的）。

动手实验：继续尝试上述比较操作符示例的部分或全部，在一行中输入 5 -eq 5 并敲回车键，看看返回的内容。

在 about_comparison_operators 的帮助文件中可以找到其他可用的比较运算符，你将在本书的第 25 章中了解其他运算符。

补充说明

如果 Cmdlet 命令不使用 11.3 节中讨论的 PowerShell 风格的比较运算符，可以使用高中或大学（甚至是工作中）学过的更加传统的编程语言形式的比较运算符。

- = 等于
- <> 不等于
- <= 小于或等于
- >= 大于或等于
- > 大于
- < 小于

如果支持布尔运算符，通常关键字是 AND 和 OR。有些 Cmdlet 命令可能提供类似 LIKE 的运算符。例如，通过-filter 参数可以找到 Get-WmiObject 支持的所有运算符。当我们在第 14 章讨论该 Cmdlet 时，会重现该列表。

每个 Cmdlet 的设计者挑选如何（以及是否需要）处理过滤，通过查看该 Cmdlet 的完整的帮助可以获得设计者期望 Cmdlet 运行方式的示例，包括帮助文件末尾附近的使用方法示例。

11.4 过滤对象的管道

当已经写好一个比较表达式，可以在哪里使用它？使用我们之前提到的比较语言。可以与一些 Cmdlet 的-filter 参数共同使用，可以与活动目录中模块以 GET-开头的命令共同使用。你也可以与 Shell 的通用过滤命令 Where-Object 共同使用。

例如，你是否想过滤掉其他信息，只留下正在运行的服务？

```
Get-Service | Where-Object -filter { $_.Status -eq 'Running' }
```

-filter 参数是一个位置参数，这意味着你经常看到很多命令没有显式指定该参数，而它的别名为 Where。

```
Get-Service | Where { $_.Status -eq 'Running' }
```

如果你习惯大声阅读上面代码，这会听起来合情合理："where status equals running。"这就阐述了它的工作原理：当你传递多个对象到 Where-Object 时，它会使用它的过滤器检查每个对象。一次只放置一个对象到占位符$_，接着运行比较操作从而查看返回值是 true 还是 false。如果是 false，该对象就会被管道移除。如果返回 true，该对象就会从 Where-Object 传输到下一个 Cmdlet 的管道中。在上面的示例中，下一个 Cmdlet 命令是 Out-Default，这会是管道的末尾（在第 8 章已经讨论过），接着开始使用格式化过程从而显示输出结果。

占位符$_是个特殊产物：之前已经见过（在第 10 章），你将在一个或更多的上下文看到它。该占位符只能在 PowerShell 能查找的特定位置中使用。在我们的示例中，该占位符恰好是在其中一个特定位置。正如你在第 10 章学习到的，句号用于告诉 Shell 不是比较整个对象，而是只比较对象的 Status 属性。

希望你开始看到 Gm 派上用场。它可以让你以快速、简单的方式发现一个对象中包含的所有属性，这样你就可以马上使用这些属性进行类似上面的比较操作。始终牢记，PowerShell 输出的列标题并不总是与属性名称保持一致。例如，运行 Get-Process，可以看见一个叫作 PM(MB) 的列；运行 Get-Process | Gm，发现列名称实际上是 PM。这个区别非常重要：总是使用 Gm 验证属性名称，而不要使用以 Format-开头的命令。

补充说明

PowerShell v3 为 Where-Object 引入了一个新的“简写”语法。当只比较一次时可以使用该语法。如果需要比较多个子项，请保持使用原来的写法。

许多人争论这个简写语法是否有所帮助。该语法如下。

```
Get-Service | Where Status -eq 'Running'
```

显然，该写法更容易阅读：免除了{}并且不需要使用看起来尴尬的占位符$_。但是新语法不是意味着你可以忽略掉旧的语法，因为你仍然需要在复杂的比较中使用它。

```
get-service | where-object {$_.status -eq 'running' -AND
➥ $_.StartType -eq 'Manual'}
```

而且，在过去多年所有有价值的例子都使用旧语法。这意味着你需要知道怎么使用它们。你也必须知道新语法，因为它现在会开始出现在开发人员的示例中。你并不需要知道这两套语法是否足够简洁，但你仍然需要在见到时能够识别它们。

11.5 使用迭代命令行模式

我们现在想为你简单介绍 PowerShell 迭代命令行模型或者称为 PSICLM（并没有为它创建一个首写字母的缩写的理由，但它的读音却很有趣）。PSICLM 的核心思想在于你不需要一开始就创建一个大而复杂的命令行，而是从简单的开始。

比方说，你想计算正在使用虚拟内存排名前十的进程所占用的虚拟内存总和。如果排名前十的进程中包含 PowerShell 进程，而又不想在结果中包含 PowerShell 进程，快速罗列出几个需要的步骤。

（1）获取进程列表；

（2）排除 PowerShell 进程；

（3）按照虚拟内存进行排序；

（4）只保存前 10 个或者最后 10 个，这取决于我们的排序方式；

（5）把剩下进程的虚拟内存相加。

我们相信你知道如何完成前 3 个步骤，第 4 个步骤完全可以使用我们的老朋友：Select-Object。

动手实验：花几分钟时间阅读 Select-Object 的帮助文档。你是否能找到让你在一个集合中保留第一个或最后一个对象的参数？

希望你能找到答案。

最终，需要把所有虚拟内存相加。这里就需要寻找新的命令，或许可以通过 Get-Command 或 Help 加上通配符寻找。可以尝试 add 关键字，或者 sum 关键字，甚至是 Measure 关键字。

动手实验：看看你能不能找到一个可以计算类似虚拟内存总量的命令。使用 Help 或 Get-Command 加上*通配符。

当你尝试这些小任务（而不是提前阅读答案），这会让自己变成一个 PowerShell 专家。一旦你觉得自己有答案了，你可能开始使用迭代的方法。

一开始，你需要获取所有的进程，这很容易满足。

```
Get-Process
```

动手实验：跟随该 Shell，并运行这些命令。验证每一个输出结果，看看你是否能预测在下一次迭代的命令中，你需要修改什么。

下一步，过滤掉不需要的进程。记住，“左过滤”意味着你想尽可能在靠近命令行开始的地方进行过滤。在该示例中，将使用 Where-Object 进行过滤，因为我们希望它成为下一个 Cmdlet。虽然效果没有在第一个 Cmdlet 命令就进行过滤得好，但是总好过在最后的管道中才过滤。

在该 Shell 中，按键盘上的向上箭头键找回你最后的命令，并添加下面的命令。

```
Get-Process | Where-Object -filter { $_.Name -notlike 'powerShell*' }
```

我们不确定进程名称是“powerShell”或“powerShell.exe”，所以使用通配符包含这两种可能。任何与该命令不匹配的进程都会留在管道内。

运行并测试，接着继续使用键盘上的向上箭头键找回上次命令并加上后面的部分。

```
Get-Process | Where-Object -filter { $_.Name -notlike 'powerShell*' } |
➥Sort VM -descending
```

按回车键可以验证你的输入，而键盘上的向上箭头键可以和后面的命令进行拼接。

```
Get-Process | Where-Object -filter { $_.Name -notlike 'powerShell*' } |
➥Sort VM -descending | Select -first 10
```

如果使用默认升序排序，你会想加入这最后的命令之前使用-last 10，而不是-first 10。

```
Get-Process | Where-Object -filter { $_.Name -notlike 'powerShell*' }|
➥Sort VM -descending | Select -first 10 |
➥Measure-Object -property VM -sum
```

即使没有使用完全一致的语法，我们也希望你至少能够找出最后一个 Cmdlet 的名称。

这个模型——运行一个命令、验证结果、键盘上的向上箭头键找回命令并修改、再次尝试——就是 PowerShell 与传统脚本语言的区别。因为 PowerShell 是一个命令行 Shell，可以立即返回结果，并且如果返回的结果不是期望结果，那么可以快速、简单地修改命令。当你将已经掌握的少量的 Cmdlets 命令与刚学到的这一点结合后，你应该可以发现你所能够拥有的能力。

11.6 常见误区

每当介绍 Where-Object 时，通常会遇到两个主要的困惑。我们试图在前面的讨论中涉及这些概念。但是如果你有任何疑问，将在这里得到解决。

11.6.1 请左过滤

你会希望你的过滤条件越接近开始的命令行越好。如果能在第一个 Cmdlet 后就完成过滤，那就这么做。如果不行，尝试在第二个 Cmdlet 命令后过滤，这样将尽可能减少后面 Cmdlet 命令的工作。

另外，尝试在尽可能靠近数据源的地方完成过滤。例如，你需要从一台远程计算机查询服务并使用 Where-Object——正如本章的一个例子——考虑利用 PowerShell 的远程调用在远程计算机上进行过滤，这比把所有的对象都获取到本地之后再进行过滤要好得多。在第 13 章将会接触远程调用，并且会使用该方法重新过滤数据源。

11.6.2 何时允许使用$_

特殊的$_占位符只有在 PowerShell 知道如何寻找它时才有效。当它有效时，它一次只包含一个从管道传输到该 Cmdlet 的对象。请记住，不同的 Cmdlet 运行和产生结果的同时，在管道传输的生命周期中，管道中包含的内容也不断变化。

同样需要小心嵌套的管道——那些出现在括号内的命令。例如，下面的示例可能会难以理解。

```
Get-Service -computername (Get-Content c:\names.txt |
➥Where-Object -filter { $_ -notlike '*dc' }) |
➥Where-Object -filter { $_.Status -eq 'Running' }
```

让我们慢慢梳理。

(1) 我们看到命令是以 Get-Service 开始，但它却不是第一个运行的命令。这是由于圆括号内的 Get-Content 先运行。

(2) Get-Content 通过管道将输出结果（由简单的 String 对象组成）传递给 Where-Object。Where-Object 和过滤器处在圆括号内，$_表示从 Get-Content 管道传输过来的 String 对象。只要字符串不是以“dc”结尾的，都会被保留并通过 Where-Object 输出。

(3) Where-Object 的输出成为圆括号内的结果，因为 Where-Object 是圆括号内的最后一个 Cmdlet 命令。因此，所有不是以“dc”结尾的计算机名称会被发送到 Get-Service 的-computername 参数中。

(4) 现在运行 Get-Service，并且产生的 ServiceController 对象将会传输到 Where-Object。该实例中 Where-Object 会一次放置一个服务到$_占位符，它会只保留那些-status 属性为 Running 的服务。

有时候，我们觉得自己的眼睛会忽略所有的花括号、句号和圆括号，但是 PowerShell 就是这么工作的。而如果你能训练自己小心阅读命令，你将会理解命令做了哪些工作。

11.7 动手实验

注意：对于本次动手实验来说，你需要一台 Windows 8 或 Windows Server 2012 或更新操作系统版本的计算机，同时需要 PowerShell v3 或更新版本。

记住，不是只有 Where-Object 方式可以过滤，它甚至不应该是你第一个想到的命令。我们已经使得本章尽量保持简短，以便让你有更多的时间进行动手实验。所以，记住左过滤的原则，尝试完成下面的内容。

1. 导入 NetAdapter 模块（存在于最新版本的客户端或服务器版本的 Windows 中）。使用 Get-NetAdapter 命令显示一个非虚拟网络适配器列表（换言之，适配器的 Virtual 属性为 false，PowerShell 使用专用常量$False）。

2. 导入 DnsClient 模块（存在于最新版本的客户端或服务器版本的 Windows 中）。使用 Get-DnsClientCache 命令显示一个从缓存中读取的 A 和 AAAA 列表。提示：如果你的缓存是空的，尝试浏览一些 Web 页面从而强制将一些项存入缓存中。

3. 显示所有位于 C:\Windows\System32 下且大于 5MB 的 EXE 文件。

4. 显示属于安全更新的补丁列表。

5. 使用 Get-Service 是否可以显示一个自动启动类型且当前没有在运行的服务列表？请仅回答是或否。你不需要编写一个命令来完成该内容。

6. 显示一个管理员安装过的补丁列表，并列出哪些是更新补丁。如果没有任何补丁，请尝试找出由 System 账户安装的补丁。注意，有些补丁包没有“installed by”这个值，不过这没关系。

7. 显示名称为“Conhost”或“Svchost”且状态为“运行”的进程列表。

11.8 进一步学习

熟能生巧，所以尝试对你学习过的命令的输出结果进行过滤，比如 Get-Hotfix、Get-EventLog、Get-Process、Get-Service 甚至是 Get-Command。例如，可以尝试对 Get-Command 的输出过滤，只剩下部分 Cmdlet 命令。或者使用 Test-Connectionping 服务器，并且只有在没有应答的情况下显示结果。我们不建议你在每个实例中都使用 Where-Object，但是你应该在适当的时候进行练习。

11.9 动手实验答案

1.
```
import-module NetAdapter
get-netadapter -physical
```
2.
```
Import-Module DnsClient
Get-DnsClientCache -type AAAA,A
```
3.
```
Dir c:\windows\system32\*.exe | where {$_.length -gt 5MB}
```
4.
```
Get-Hotfix -Description 'Security Update'
```
5.
```
get-hotfix -Description Update | where {$_.InstalledBy -match
"administrator"}
```
或下述任一命令：
```
get-hotfix -Description Update | where {$_.InstalledBy -match
"system"}
get-hotfix -Description Update | where {$_.InstalledBy -eq "NT
Authority\System"}
```
6.
```
get-process -name svchost,conhost
```