第 19 章　输入和输出

到现在为止，在本书中，我们主要依赖 PowerShell 源生的能力来输出表格和列表。当你开始将多个命令整合成更复杂的脚本时，你可能想要更精确地控制展示的结果。你可能也希望能提示用户进行输入。在本章中，你将会学习到如何收集输入以及如何展示期望的输出结果。

19.1　提示并显示信息

PowerShell 如何展示信息和进行对应的提示，依赖于 PowerShell 运行的方式。你可以看到，PowerShell 被内置为一种底层的引擎。

与你进行交互的对象称为主机应用程序。当运行 PowerShell.exe 时，你看到的命令行控制台称为控制台主机。图形化的 PowerShell ISE 通常被称为 ISE 主机或者图形化主机。其他非微软的应用程序也可以调用 PowerShell 的引擎。你与主机应用程序进行交互，之后主机应用程序将执行的命令传递给该引擎。主机应用程序会展现引擎产生的结果集。

图 19.1 说明了 PowerShell 引擎和多种主机应用程序之间的关系。每个主机应用程序负责图形化展现引擎产生的任何输出结果，同时负责通过界面收集引擎需要的任何输入信息。也就意味着，PowerShell 可以通过多种方式展现执行结果和收集输入信息。实际上，控制台主机和 ISE 使用不同的方法来收集输入信息：控制台主机会在命令行中展现一个文本的提示框，但是 ISE 会弹出一个会话框，该会话框中包含文本区域一个“OK”按钮。

我们希望指出这些差异点，因为有些时候可能会使得初学者非常困惑。为什么一个命令在命令行中的行为与在 ISE 中的行为大相径庭？这是因为你与 Shell 交互的方式由

主机应用程序决定，并不是由 PowerShell 本身决定。我们即将展示给你的命令会显示使用不同的行为，这些行为主要依赖于你在哪里执行这些命令。

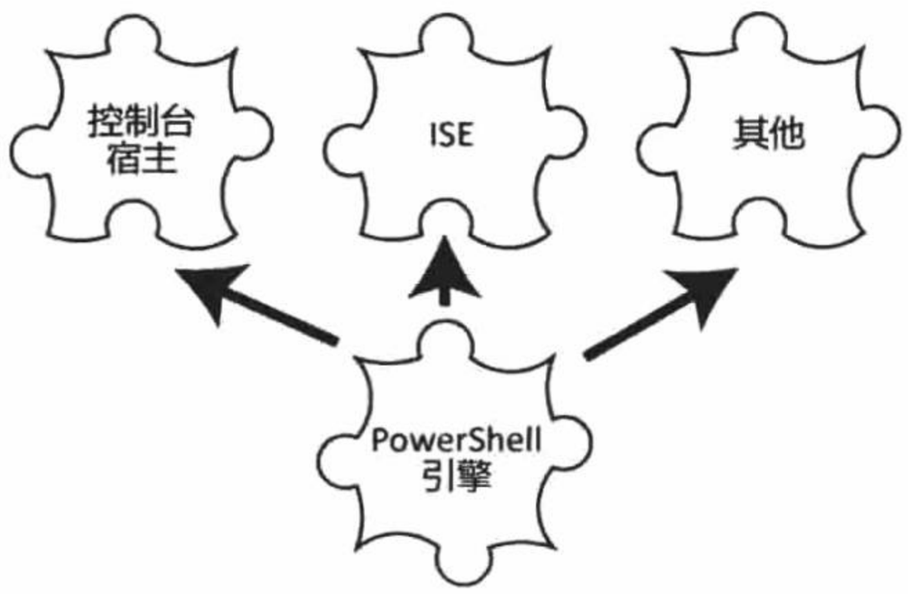

图 19.1 多种应用程序都可以使用 PowerShell 引擎

19.2 Read-Host 命令

PowerShell 的 Read-Host Cmdlet 的功能是展示一个文本提示框，然后收集来自用户的输入信息。因为在前一章节中，你第一次看到我们使用这个 Cmdlet，所以你会觉得语法比较熟悉：

```
PS C:\> Read-Host "Enter a computer name"
Enter a computer name: SERVER-R2
SERVER-R2
```

该示例突出了 Cmdlet 的两个重要的事实：

- 提示信息的最后添加了一个冒号。
- 用户键入的任何信息都会作为该 Cmdlet 的返回结果（严格来说，键入的信息被放进了管道）。

你经常会将该输入信息传递给一个变量，类似下面这样：

```
PS C:\> $ComputerName=Read-Host "Enter a computer name"
Enter a computer name: SERVER-R2
```

动手实验： 现在请开始跟着这些示例学习吧。此时，$ComputerName 变量中应该存在一个有效的计算机名称。除非使用的计算机名称是 Server-2，否则请不要使用 Server-2。

正如前面提到的，第二版的 PowerShell ISE 会展现一个对话框，而不是直接在命令行中进行提示，如图 19.2 所示。其他的主机应用程序，比如 PowerGUI、PowerShell Plus 或者 PrimalScript 等脚本编辑器，均使用各自对应的方式执行 Read-Host。请记住，第三版的 PowerShell ISE 仅会展示更简单的两个窗格。不像第二版的 PowerShell ISE 那样，

第三版的 PowerShell ISE 会像常规的控制台窗口一样展示一个命令行的提示窗口。

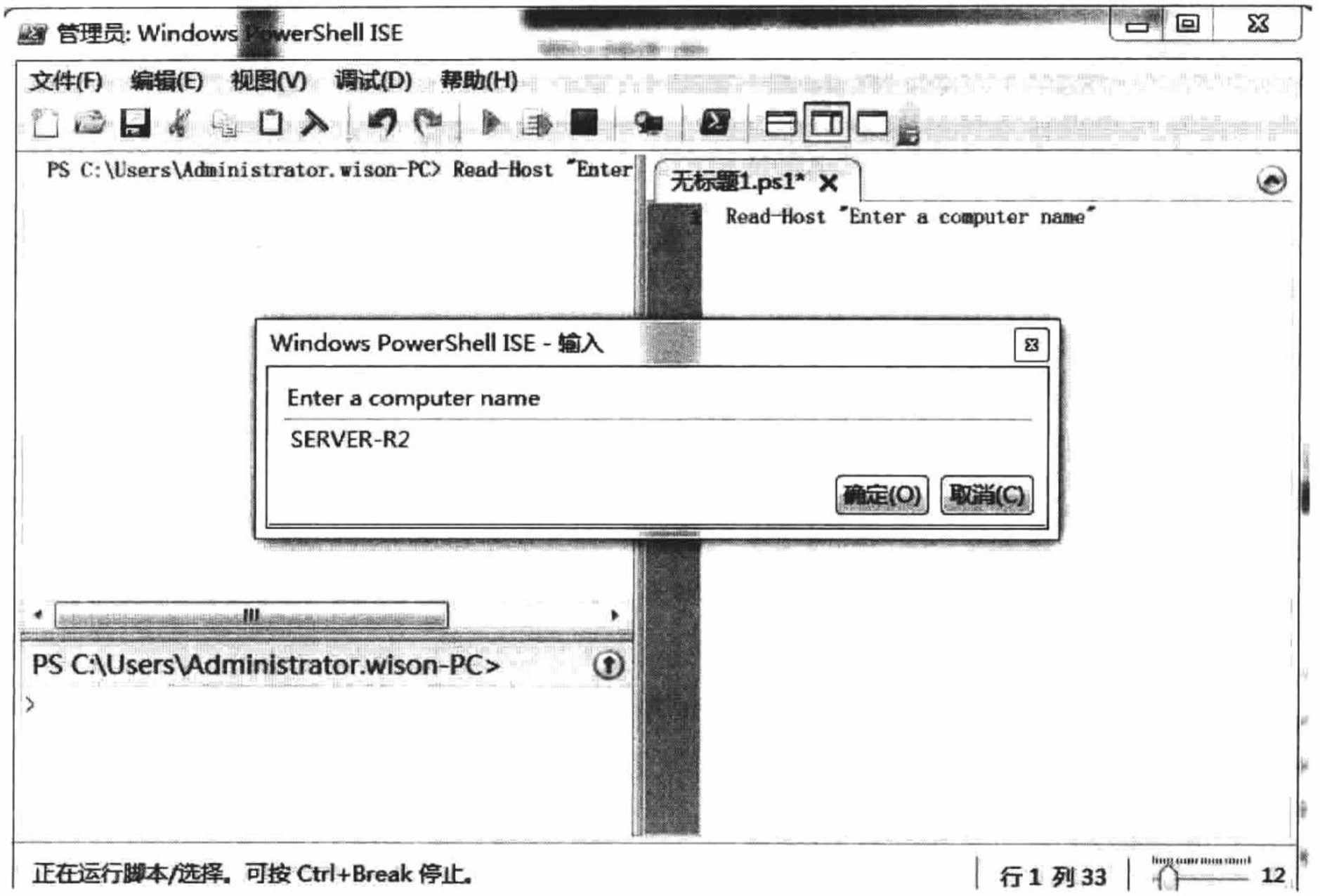

图 19.2 第二版的 ISE 会为 Read-Host 命令弹出一个对话框

关于 Read-Host 命令也没什么好再多谈的了：它是一个很有用的 Cmdlet，但是并不是一个让人很兴奋的 Cmdlet。实际上，在大多数课堂讲解了 Read-Host 命令后，总有人会问我们：“是否有其他方法可以始终展现一个图形化的输入框？”很多管理员会给用户部署一些 PowerShell 脚本，但是又不希望用户必须在命令行界面输入信息（毕竟，这并不是很“Windows 风格”）。我们想说的是可以实现，但是稍微复杂。最终的结果如图 19.3 所示。

为了创建一个图形界面的输入框，你必须借助于.Net Framework 本身。使用下面的命令：

```
PS C:\> [void][System.Reflection.Assembly]::LoadWithPartialName('Microsoft.
➥VisualBasic')
```

你只需要在某个 PowerShell 会话执行一次即可，但是即使执行多次，也不会有什么影响。

该命令会载入.Net Framework 中的一个组件 Microsoft.VisualBasic，实际上 PowerShell 并不会自动载入该组件。该 Framework 组件包含了大量的 VisualBasic 核心的框架元素，其中就包括图形化输入框。

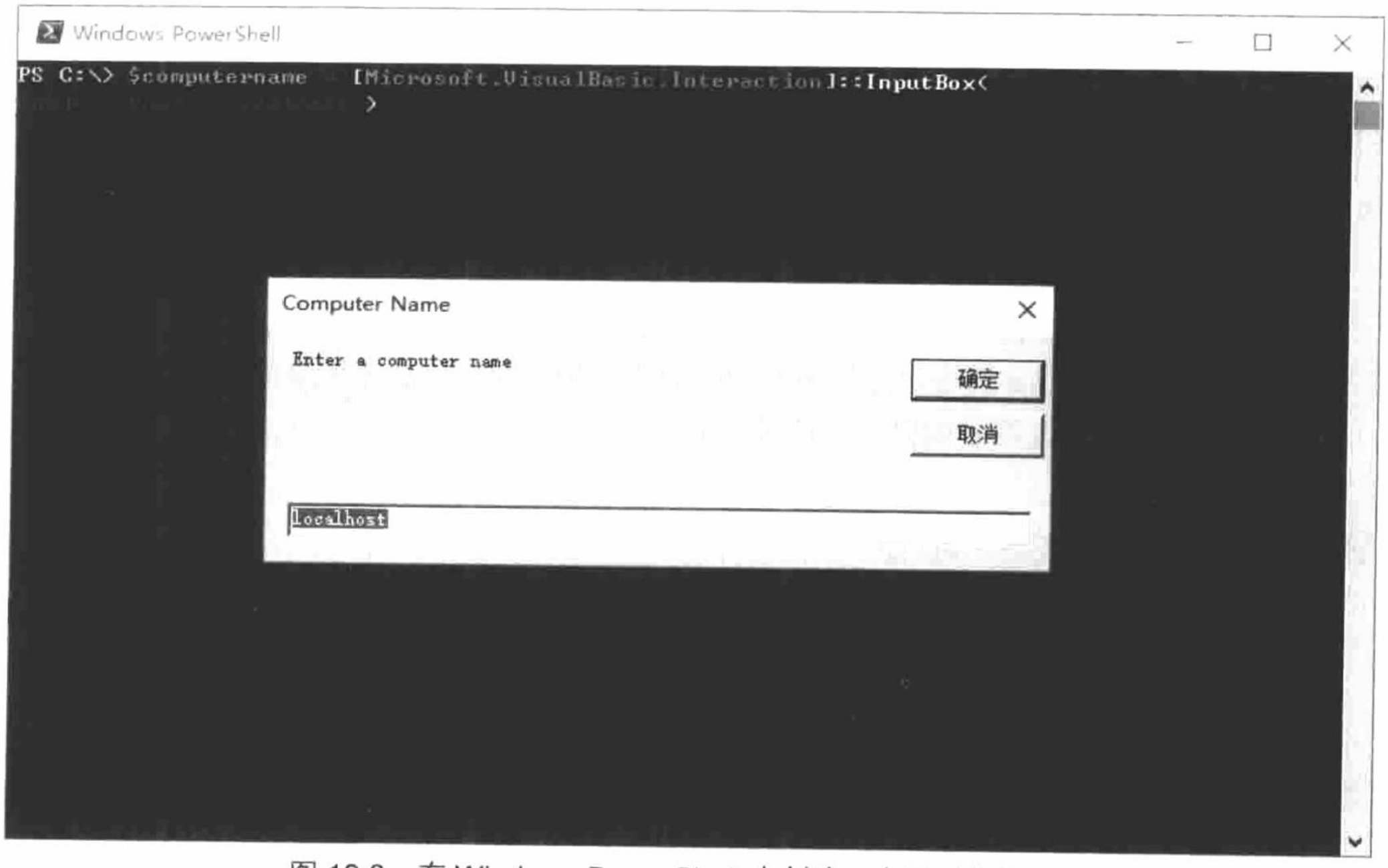

图 19.3 在 Windows PowerShell 中创建一个图形化输入框

让我们看看该命令是怎么实现的：

- [Void]部分将命令的返回结果转化为[Void]类型。在前面的章节中，你已经学过如何转化整型数据；Void 数据类型是一种特定的类型，意味着“抛弃产生的结果”。我们不想看到该命令的执行结果，所以我们将该结果转化为 Void 类型。实现该目的的另外一种方法是将该结果集通过管道传递给 Out-Null。
- 接下来我们会访问 System.Reflection.Assembly 类型，该类型代表了我们的应用程序（在这里就是 PowerShell）。我们将该类型名称放在一个方括号内，犹如我们申明了一个该类型的变量。但是我们这里并不是真正申明一个变量，而是用了两个冒号来访问该类型的静态方法。静态方法并不依赖于我们创建一个该类型的实例而存在。
- 我们这里使用的静态方法是 LoadWithPartialName()，该方法会接收我们希望添加的 Framework 组件名称。

如果你觉得很难理解，也没关系；你可以照搬该命令，不需要理解它们的原理。一旦该 Framework 组件被载入，你可以通过下面的命令来使用它。

```
PS C:\> $ComputerName = [Microsoft.VisualBasic.Interaction]::InputBox('Enter a
➥computer name','Computer Name','localhost')
```

在该示例中，我们再次使用了一个静态方法，这一次是 Microsoft.VisualBasic.InterAction 类型。我们使用前面的命令将其载入到内存中。再次说明，如果你对“静态方法”感到很难理解，也没关系——直接照搬命令即可。

这里你可以修改的地方是 InputBox()方法的三个参数。

- 第一个参数是提示框中的文本信息。
- 第二个参数是提示对话框的标题。
- 第三个参数——可以是空白或者完全省略，是你想显示在输入框中的默认值。

使用 Read-Host 命令可能比前面示例的步骤稍微简单，但是如果你仍然坚持使用对话框，该示例就说明了如何创建该对话框。

19.3 Write-Host 命令

既然你可以收集输入信息，那么也会希望了解一些展示返回结果的方法。Write-Host 命令就是其中的一种方法。这并不总是最好的一种方法，但是你可以使用它，并且重要的是，你需要了解它的工作原理。

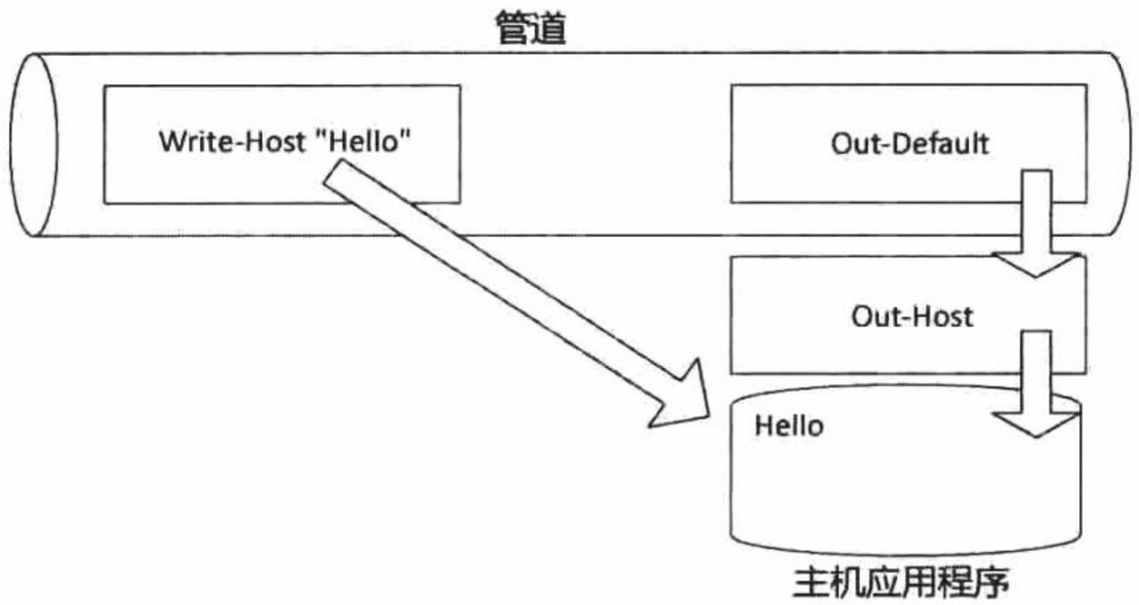

图 19.4 Write-Host 会绕开管道，直接写到主机应用程序的显示界面

如图 19.4 所示，Write-Host 会和其他 Cmdlet 一样使用管道，但是它并不会放置任何数据到管道中。相反，它会直接写到主机应用程序的界面。正因为可以这样做，所以我们可以使用命令行中的-ForegroundColor 和-BackgroundColor 参数来将前景和背景设置为其他颜色。

```
PS C:\> Write-Host "COLORFUL!" -Fore Yellow -Back Magenta
COLORFUL!
```

动手实验：你需要运行该命令来查看带有色彩的结果集。

注意：不是每个使用 PowerShell 的应用程序都支持其他颜色，也并不是每个应用程序都支持整系列的颜色。当你尝试在某个应用程序中设置颜色时，通常会忽略掉不喜欢或者不能显示的颜色。这也是我们需要避免依赖于特定颜色的一个原因。

当需要展示一个特定的信息，比如使用其他颜色来吸引人们的注意力时，你应该使

用 Write-Host 命令。但是针对使用脚本或者命令来产生常规的输出结果而言，这并不是一个恰当的方法。

例如，你永远都不应该使用 Write-Host 命令来手动格式化一个表格——你能找到更好的方法来产生输出结果，比如使用那些让 PowerShell 可以实现处理格式化功能的技巧。在本书中我们不会讲到这些技巧，因为它们更多属于较为复杂的脚本以及工具制作领域。但是，你可以通过 *Learn PowerShell Toolmaking in a Month of Lunches (Manning, 2012)* 来学习这些输出技巧的全部知识。针对产生错误信息、警告信息、调试信息等而言，Write-Host 命令也不是最好的方法——再次申明，你可以找到更合适的方法来实现这些功能。当然本书中也会讲到这些。如果你恰当地使用 PowerShell，那么你可能不会多次使用到 Write-Host 命令。

注意： 我们经常看到有人使用 Write-Host 命令来显示“温暖和模糊”的信息一比如“nowconnecting to Server-2”，“testing for folder”等。请不要这样做，有更恰当的方法来实现这些功能，就是 Write-Verbose。

补充说明

我们将在第 22 章中深入讲解 Write-Verbose 以及其他的一些 Write Cmdlet。但是如果你现在尝试使用 Write-Verbose 命令，你可能会很沮丧地发现该命令不会返回任何的结果，准确地说是默认情况下不会返回。

如果你计划使用 Write Cmdlet，诀窍就是首先打开它们。例如，设置$VerbosePreference="Continue"将会启用 Write-Verbose，$VerbosePreference="SilentlyCon tinue"会截断其输出。你会看到针对 Write-Debug($DebugPreference)和 Write-Warning（$WarningPreference）命令也存在类似的“Preference”变量。

在第 22 章中会介绍一种更酷的方法来使用 Write-Verbose 命令。

看起来使用 Write-Host 命令会更容易，如果你希望使用该命令，那么也可以。但是请记住，如果使用其他的 Cmdlet，比如 Write-Verbose 命令，你会更加贴近 PowerShell 本身的使用方式，最终得到更一致的体验。

19.4 Write-Output 命令

不像 Write-Host 命令，Write-Output 命令可以将对象发送给管道。因为它不会直接写到显示界面，所以不允许你指定其他任何的颜色。实际上从技术来说，Write-Output（或者它的别名 Write）根本不是设计出来展示结果的。正如我们所讲，它将这些对象发送给管道——也就是最终展示这些对象的管道。图 19.5 展现了对应的工作原理。

快速复习一下第 10 章中的知识点：如何将对象从管道传递给显示界面。下面就是最基本的过程。

（1）Write-Output 命令将 String 类型的对象 Hello 放入到管道中。

（2）因为管道中不存在其他对象，Hello 会到达管道的最末端，也就是 Out-Default 命令的位置。

（3）Out-Default 命令将对象传递给 Out-Host 命令。

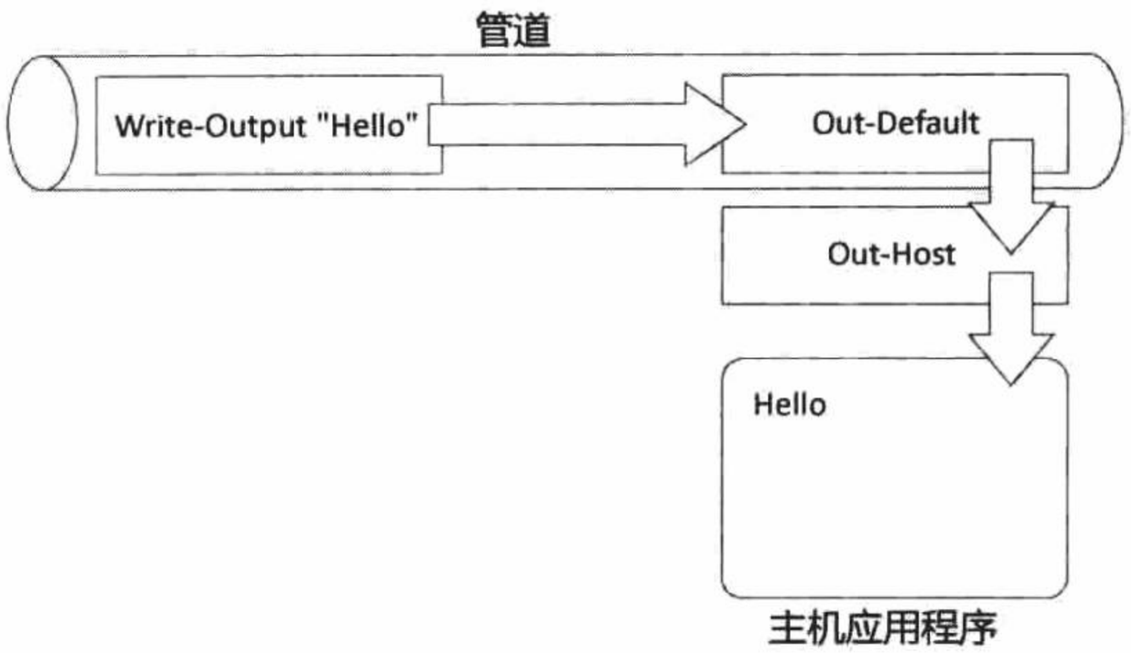

图 19.5 Write-Output 将对象放入管道，在某些情况下，最终会导致对象被展示出来

（4）Out-Host 命令要求 PowerShell 的格式化系统格式化该对象。因为该示例中为简单的 String 对象，所以格式化系统会返回该 String 对象的文本信息。

（5）Out-Host 将格式化的结果集放在显示界面上。

执行的结果类似使用 Write-Host 命令的返回结果，但是该对象通过不同的路径到达最后阶段。该路径是非常重要的，因为在管道中可以包含其他的对象。例如，考虑下面的命令（欢迎你尝试执行该命令）：

```
PS C:\> Write-Output "Hello" | Where-Object { $_.Length -GT 10 }
```

你并没有看到该命令返回任何结果集，图 19.6 解释了其原因。“Hello”字符被放进管道。但是在它到达 Out-Default 命令之前，它必须经由 Where-Object 命令，该命令会去除长度（Length）属性小于或者等于 10 的对象。在该示例中，字符对象是“Hello”，所以此时该对象就会从管道中被筛选掉。由于在管道中不存在任何对象可以被传递给 Out-Default，因此最终也就没有对象传递给 Out-Host 命令，那么也就不会显示任何的信息。

将前一个命令与下面的命令进行对比：

```
PS C:\> Write-Host "Hello" | Where-Object { $_.Length -GT 10 }
Hello
```

这里所做的变更仅是使用 Write-Host 替换了 Write-Output 命令。这时，“Hello”字符会直接被传递给显示界面，而不会进入管道中。Where-Object 命令并没有任何传入数据，因此也就不会有任何信息经由 Out-Default 和 Out-Host 展现出来。但是由于“Hello”字符已经被直接传递给显示界面，所以我们仍然可以看到它。

Write-Output 命令看起来可能是新学习的命令，但是其实你一直都在使用它。它是 PowerShell 默认使用的一个 Cmdlet。当你通知 PowerShell 去完成某项功能（但是又不是使用命令）时，PowerShell 会在底层将你键入的任意信息传递给 Write-Output 命令。

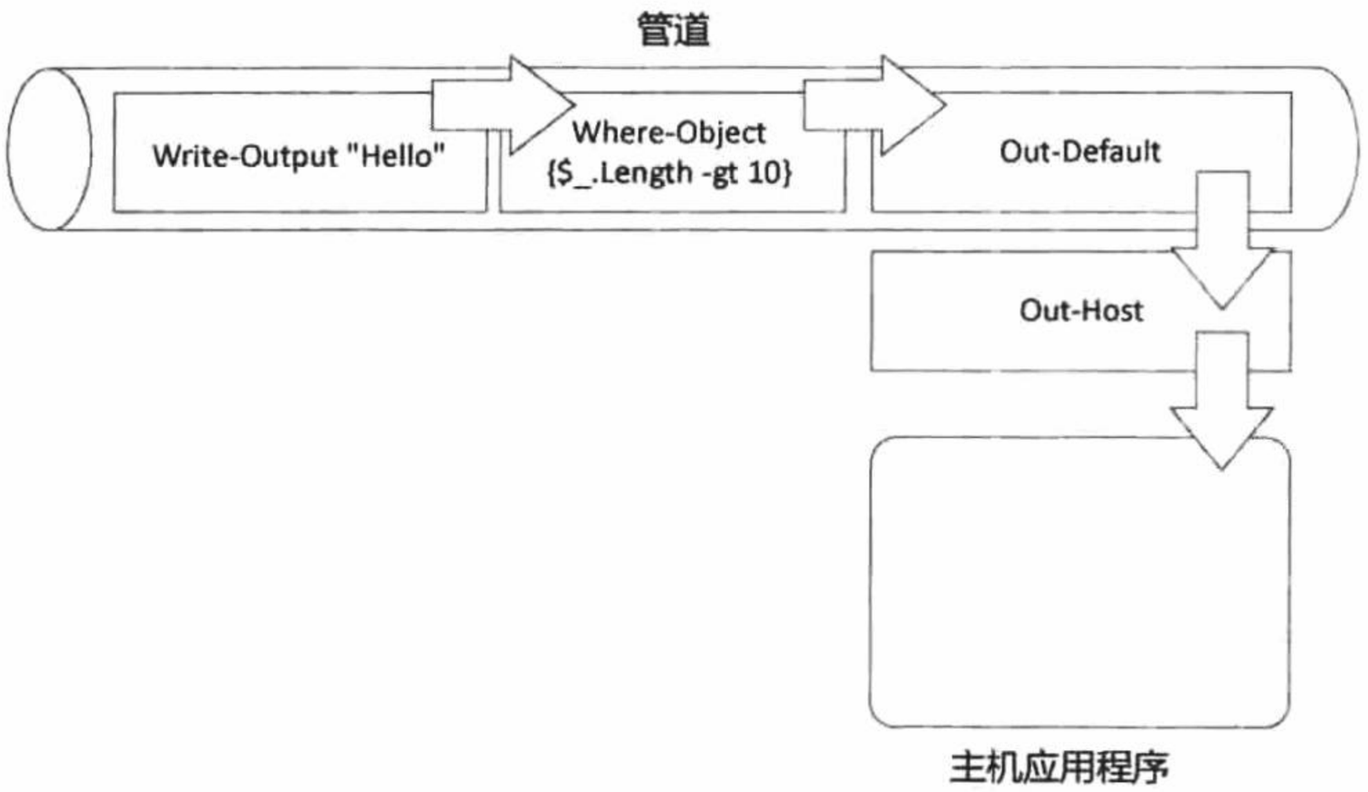

图 19.6 将对象放进管道，也就意味着它们在显示之前可以被过滤掉

19.5 其他写入的方式

PowerShell 中也存在其他方法来产生输出结果。这些方法都不会像 Write-Host 那样向管道写入某些信息，它们看起来更像是 Write-Host 命令。但是它们可以通过可被截断的方式产生结果。

PowerShell 针对每种输出方法都有对应的内置配置变量。如果配置变量设置为“Continue”，那么我们即将展示给你的命令就会真正产生输出结果。如果配置变量被设置为“SilentlyContinue”，那么关联的输出命令就不会产生任何信息。表 19.1 包含了这些 Cmdlet 的列表。

表 19.1 可选的输出 Cmdlet

Cmdlet	作用	配置变量
Write-Warning	显示警告信息，默认会以黄色字体显示，同时前面带有“警告：”字样	$WarningPreference（默认为 Continue）
Write-Verbose	显示详细信息，默认会以黄色字体显示，同时前面带有“详细信息：”字样	$VerbosePreference（默认为 SilentlyContinue）
Write-Debug	显示调试信息，默认以黄色字体显示，同时前面带有“调试：”字样	$DebugPreference (默认为 SilentlyContinue)
Write-Error	产生一个错误信息	$ErrorActionPreference (默认为 Continue)

Write-Error 命令会有点不一样，因为它会将错误信息写入 PowerShell 的错误流中。

另外，PowerShell 还存在一个 Cmdlet Write-Progress，该 Cmdlet 可以展示进度条，但是实现原理完全不一样。你可以阅读其帮助文档来获取更多的信息以及示例。本书中不会涉及该命令。

为了使用这些 Cmdlet，首先你需要确保关联的配置变量设置为“Continue”。（如果上面列表中的两个 Cmdlet 的配置变量保留默认值 SilentlyContinue，你不会看到任何的输出结果。）之后，就可以使用该 Cmdlet 来输出一些信息。

注意：分 PowerShell 的主机应用程序会在不同的位置展现这些 Cmdlet 的输出信息。比如在 PrimalScript 中，调试信息会写入到另外一块输出窗格中，而不是脚本的主输出窗格，这样可以更容易将调试信息独立开来进行分析。在本书中，我们不会深入讲解调试相关的知识，但是如果你感兴趣，可以阅读 PowerShell 帮助文档中该 Cmdlet 对应的部分。

19.6 动手实验

注意：对于本次动手实验环节，需要运行 3.0 版本或者之后版本的 PowerShell。

Write-Host 和 Write-Output 命令可能使用起来更为棘手。试试看，你可以完成下面列表中的几个任务。如果无法完成其中某些任务，那么可以参考 MoreLunches.com 网站上的示例答案。

1. 使用 Write-Output 命令来返回 100 除以 10 的结果。
2. 使用 Write-Host 命令来返回 100 除以 10 的结果。
3. 提示用户输入姓名，然后以黄色字体显示该姓名。
4. 提示用户输入姓名，并且仅当长度大于 5 时才显示该姓名。请使用单行命令完成——不要使用变量。

这就是本章动手实验环节的全部任务。因为这些 Cmdlet 都很简单，我们希望你能自行花更多的时间来测试它们。请保证一定要测试——在接下来的部分，我们会提供一些建议。

动手实验：完成本章节的动手实验环节后，请尝试完成本书附录中的实验回顾 3。

19.7 进一步学习

请花费一定的时间来熟悉本章中所有的 Cmdlet。确保你可以通过这些 Cmdlet 显示详细信息，接收输入数据，甚至可以显示图形的输入框。从现在起，你将会使用本章中所讲的 Cmdlet，因此你应该阅读它们对应的帮助文档，甚至简单记下它们的简单语法提示，以便后续查找。