第24章　使用正则表达式解析文本文件

正则表达式是令人尴尬的主题之一。经常有学生让我们解释这个概念——在解释的过程中才发现他们完全不需要使用正则表达式。正则表达式（regular expression，或 regex）能够非常有效地进行文本解析，你经常会在 UNIX 或 Linux 操作系统中用到。在 Power Shell 中，你会倾向于尽量少用文本解析——我们也发现你很少需要用到正则表达式。也就是说，我们当然知道某些时候在 PowerShell 中，你需要解析一些类似 IIS 日志的文本内容。这也是我们在本文阐述正则表达式的使用方式——用于解析文本文件。

不要错误地理解我们的意思：你可以用正则表达式做更多的事情，我们将在本章结束之前阐述其中一部分。为了确保你有一个正确的期望，我们事先声明，我们在本书中将不会从宽度和深度方面尝试覆盖正则表达式的方方面面。正则表达式可以非常复杂。其自身就是一个完整的技术体系。我们将会把知识以直接应用到实践的方式传授给你，从而帮助你起步。在此之后，我们会阐述一个大方向，帮助你进一步自学。这就足够了。

本章的目标是以最简单的方式介绍正则表达式的语法，并且展示 PowerShell 如何使用正则表达式，如果你希望探索更加复杂的表达式，当然更好。这里我们将教会你如何在 Shell 中使用正则表达式。

24.1　正则表达式的目标

正则表达式需要以特定语言编写，其目标是为了定义文本模型。比如说，IPv4 地址以 1～3 位的数字为一组，一共 4 组。通过正则表达式可以定义该模式。虽然定义后还是会有 211.193.299.299 这样的非法地址，但这属于识别文本模式与数据有效范围的区别。

正则表达式最大的使用场景之一，也就是我们本章涵盖的内容——在一个类似日志这样大的文本文件中检测特定的文本模式。举例来说，通过正则表达式在一个 Web 服务器日志文件中找到代表 HTTP 500 的特定文本，或是在一个 SMTP 服务器日志文件中寻找电子邮件地址。除了检测文本模式之外，还可以使用正则表达式捕捉匹配的文本，从而在日志文件中提取出邮件地址。

24.2 正则表达式入门

最简单的正则表达式就是你所期望匹配的文本字符串。比如“Don”，从技术角度来说，这就是一个正则表达式，在 PowerShell 中能够匹配“DON”“don”“Don”“DoN”等——PowerShell 默认的匹配规则不区分大小写。

某些特定的字符在正则表达式有特殊的含义，这些特定字符可以允许你检测文本变量中的文本模式。下面是一些示例。

- \w 用于匹配“文本字符”，也就是字母、数字以及下划线，但不包含标点符号和空格。正则表达式\won 可以匹配“Don”“Ron”以及“ton”，\w 可以代表任意字母、数字或下划线。
- \W 与 \w 相反（这也是 PowerShell 会区分大小写的一个示例），意思是它将会匹配空格与标点符号——也就是“非字母”。
- \d 用于匹配包括 0 到 9 的任意数字。
- \D 用于匹配任意非数字。
- \s 用于匹配任意空格字符，比如 Tab、空格或者回车符。
- \S 用于匹配任意非空格字符。
- .（句号）代表任意单个字符。
- [abcde]用于匹配在该集合中的任意字符。正则表达式 d[aeiou]n 可以匹配“Don”“Dan”，但不会匹配“Doun”或“Deen”。
- [a-z]匹配在此范围内的一个或多个字符，可以使用逗号分隔列表指定多个范围，比如说[a-f,m-z]。
- [^abcde] 用于匹配不在该集合中的一个或多个字符，意味着正则表达式 d[^aeiou]可以与“dns”匹配，但无法与“don”匹配。
- 将？置于另一个字母或特殊符号之后，可以用于匹配该字符的一个实例。所以正则表达式 do?n 可以与“don”匹配，但不会与“doon”匹配。该正则表达式还可以与“dn”匹配，这是由于？还可以代表空实例。
- * 用于匹配该符号之前任意数量的实例。正则表达式 do*n 将会与“doon”和“don”匹配。该正则表达式还可以与“dn”匹配，这是由于*还可以代表空实例。

- + 用于匹配该符号之前任意数量的实例。你会经常见到该字符和括号一起使用，从而创建了一种子表达式。举例来说，正则表达式（dn）+o 可以与“dndndndno”匹配，这是由于该正则表达式可以重复匹配子表达式“dn”。
- \（反斜杠）是正则表达式转义字符。将该字符置于在正则表达式中有特殊意义的字符之前，从而使得该字符变为该字符的字面意思。比如，正则表达式\.仅仅匹配一个句号，而不是像正常情况那样用于代表任意单个字符。如果希望匹配反斜杠，那么在反斜杠之前再加一个反斜杠：\\。
- {2}用于匹配该符号之前特定数量的实例。比如，\d{1}用于匹配 1 个数字。使用 {2,} 匹配 2 或多个数字，使用{1,3}匹配至少 1 个但不超过 3 个实例。
- ^用于匹配字符串开始部分。比如，正则表达式 d.n 既可以匹配“don”，又可以匹配“pteranodon”。而正则表达式^d.n 只能匹配“don”，而无法匹配“pteranodon”。这是由于^使得匹配只能从字符串开始部分匹配，而^与[]共同使用时表达取匹配的反义。
- $用于匹配字符串结尾部分。比如，正则表达式.icks 既可以与“hicks”匹配，又可以与“sticks”（本例中该匹配其实匹配的是“ticks”）匹配，还能够与“Dickson”匹配。但正则表达式.icks$无法与“Dickson”匹配，这是因为$表示字符“s”应该是该字符串的最后一个字符。

总之，你快速查看了一遍正则表达式的语法。正如我们在开始所写的那样，正则表达式还有大量内容，但这些内容足够你完成基本工作。让我们来看一些正则表达式的例子。

- \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}可以匹配 IPv4 地址的模式，但该表达式可以接受“432.567.875.000”这样的非法地址，也可以接受“192.169.15.12”这样的合法地址。
- \\\\\w+(\\\w+)+可以匹配通用命名惯例（UNC）路径。大量的反斜杠使得该正则表达式难以阅读，这也是为什么在将正则表达式部署到生产环境之前对正则表达式进行调试和调整。
- \w{1}\.\w+@company\.com 可以匹配特定类型的电子邮件地址：首先是一个字母，然后是句号，最后是“@company.com”。比如 d.jones@company.com 可以与该正则表达式进行匹配，“donald . jones@company. com.org”也能够匹配。我们将正则表达式能够匹配的部分进行加粗——正则表达式允许在匹配文本的开始或结尾存在额外的字符。在这种情况下就可以考虑使用^或$。

注意：你可以通过在 PowerShell 运行 help about_regular_expressions，发现更多关于正则表达式的基本语法。在本章末尾，我们将为你更进一步学习提供一些额外的资源。

24.3 通过-Match 使用正则表达式

PowerShell 包含一个比较运算符-Match，以及一个区分大小写的版本-Cmatch。通过这两个运算符与正则表达式进行比较。下面是一些示例。

```
PS C:\> "don" -match "d[aeiou]n"
True
PS C:\> "dooon" -match "d[aeiou]n"
False
PS C:\> "dooon" -match "d[aeiou]+n"
True
PS C:\> "djinn" -match "d[aeiou]+n"
False
PS C:\> "dean" -match "d[aeiou]n"
False
```

虽然使用正则表达式的方法很多，但我们主要依靠-Match 测试正则表达式并确保正则表达式能够正确生效。如你所见，左边是你希望测试的字符串，右边是正则表达式。如果两端匹配，那么输出 True；如果两端不匹配，那么输出 False。

动手实验：是时候停止阅读并尝试使用-Match 运算符了。运行一些之前我们在语法小节给你的示例，并确保你能够在 Shell 中将-Match 运算符运用得得心应手。

24.4 通过 Select-String 使用正则表达式

现在我们终于到了本章的精华之处。我们使用一些 IIS 日志文件作为示例，这是由于 IIS 日志是纯文本，而这正是正则表达式的用武之地。如果能将这些日志以更面向对象的风格读取到 PowerShell 中，那再好不过。可惜不能……所以只能使用正则表达式。

让我们先在日志文件中查找 40x 错误。这类错误主要是“找不到文件”以及其他错误，我们希望为 Web 开发人员生成一个缺失文件的报表。日志文件中，每一个 HTTP 请求为一行，每行又被分为以空格分割的域。我们还有一些文件名称中包含“401”等，比如“error401.html”，我们不希望这部分结果出现在我们的结果中。我们将会指定一个类似\s40[0-9]\s 的正则表达式，因为通过在 40x 错误之前和之后匹配空格，该表达式将能够匹配从 400 到 499 的错误。下面是我们使用的命令。

```
PS C:\logfiles> get-childitem -filter *.log -recurse | select-string -pattern
      "\s40[0-9]\s" | format-table Filename,LineNumber,Line -wrap
```

注意，我们将当前目录变更为 C:\logfiles，开始运行命令。我们通过寻找所有以.log 结尾的文件，并递归查找子目录。这可以确保所有的日志文件都可以被包含在输出结果之内。接下来我们使用 Select-String，提供正则表达式作为参数。该命令的结果是一个类型为 MatchInfo 的对象；这里使用 Format-Table 命令，使得显示结果包含文件名称、行号以及包含匹配结果的文本。这使得找到缺失文件非常容易。然后我们将报表给予 Web 开发人员。

接下来，我们希望扫描所有被基于 Gecko 浏览器访问过的文件。开发人员告诉我们，使用该类浏览器访问我们的网站的用户会遇到一些问题，他们希望找到具体被访问的文件。他们还希望将问题范围缩减为使用 Windows NT6.2 操作系统运行浏览器的用户，这意味着我们需要在 user-agnet 中寻找类似下面的字符串。

```
(Windows+NT+6.2;+WOW64;+rv:11.0)+Gecko
```

开发人员强调是否为 64 位操作系统无关紧要，因此我们不希望 User-agent 中仅是包含“WOW64”的结果。最终我们得到这个正则表达式：6\.2;[\w\W]+\+Gecko——让我们对其进行分解。

- 6\.2; ——这就是“6.2”；我们使用转义字符将句号变为字面意思上的句号，而不是作为单字符的通配符。
- [\w\W]+ ——一个或多个字符或非字符——换句话说是任何内容。
- \+Gecko ——也就是字面意义上的加号，然后是“Gecko”。

下面是从日志文件返回匹配行的命令，还包含前几行的返回结果。

```
PS C:\logfiles> get-childitem -filter *.log -recurse |
➥select-string -pattern "6\.2;[\w\W]+\+Gecko"

W3SVC1\u_ex120420.log:14:2012-04-20 21:45:04 10.211.55.30 GET
      /MyApp1/Testpage.asp - 80 - 10.211.55.29
      Mozilla/5.0+(Windows+NT+6.2;+WOW64;+rv:11.0)+Gecko/20100101+Firefox/11.0
      200 0 0 1125
W3SVC1\u_ex120420.log:15:2012-04-20 21:45:04 10.211.55.30 GET /TestPage.asp-
      80 - 10.211.55.29
      Mozilla/5.0+(Windows+NT+6.2;+WOW64;+rv:11.0)+Gecko/20100101+Firefox/11.0
      200 0 0 1 109
```

这次我们保持输出结果为默认格式，而不是将结果发送给用于格式化的 Cmdlet。

在最后一个例子中，将 IIS 日志文件变为 Windows 安全日志。事件日志实体中包含 Message 属性，该属性中包含关于事件信息的细节。遗憾的是，该信息并没有良好的格式化以便于人们阅读，也不易于计算机解析。我们希望查找所有事件 ID 为 4624 的事件，该事件代表账户登录事件（该 ID 代表的含义可能根据 Windows 版本的不同而有所不同；我们的示例是在 Windows Server 2008 R2 上）。但我们只希望查看账户名称以“WIN”开头的登录信息，这些账户都与在域中的计算机账户关联。另外，我们还要求

账户结尾必须是从 TM20$到 TM40$的字符，这些是我们感兴趣的特定计算机。我们需要的正则表达式大概如下：WIN[\W\w]+TM[234][0-9]\$ ——注意我们需要使用转义符号将末尾的$进行转义，因此该符号不会被解释成字符串结尾标记。我们需要包含[\W\w]（非字符和字符），这是由于我们的账户名称中可能包含连字符，该连字符无法与\w 字符类匹配。因此最终下面是我们的命令。

```
PS C:\> get-eventlog -LogName security | where { $_.eventid -eq 4624 } |
➥select -ExpandProperty message | select-string -pattern
➥"WIN[\W\w]+TM[234][0-9]\$"
```

在开始部分，我们使用 Where-Object，从而仅使得 ID 为 4624 的事件被筛选出来。然后我们将 Message 属性的内容存入纯字符串，并通过管道将其传输给 Select-String。注意，这将会输出匹配的信息文本；如果我们的目标是输出所有匹配的事件，我们需要使用另一种方式。

```
PS C:\> get-eventlog -LogName security | where { $_.eventid -eq 4624 -and
    ➥$_.message -match "WIN[\W\w]+TM[234][0-9]\$" }
```

这里，我们不是输出 Message 属性的内容，而是查找 Message 属性匹配正则表达式的记录——接下来输出整个 Event 对象。接下来所使用的命令取决于结果希望输出的形式。

24.5 动手实验

注意： 对于本次动手实验来说，你需要 Windows8 或 Windows Server 2012 或更新版本的操作系统，从而运行 PowerShell v3 或更新版本。

请不要会错意，正则表达式的复杂程度可以让你头痛，所以请不要开始就尝试创建复杂的正则表达式——从简单开始。下面一些练习可以帮助你入门。使用正则表达式和运算符完成下列任务。

1. 获取活动目录中所有名称包含两位数字的文件。

2. 获得计算机中所有来自微软的进程，并显示进程 ID、名称以及公司名称。提示：通过管道将 Get-Process 传递给 Get-Member，从而显示属性名称。

3. 在 Windows Update 日志中，该日志通常位于 C:\Windows，你只希望显示代理开始安装文件的日志行。你或许需要在记事本中打开日志文件，从而找出你需要选择的字符串。

4. 使用 Get-DNSClientCache 这个 cmdlet 显示列表，该列表仅显示 Data 属性为 IPV4 地址的条目。

24.6 进一步学习

你将会在 PowerShell 的其他地方发现使用正则表达式，其中很多地方包含本书未提

到的 Shell 元素。下面是一些示例。

- Switch 脚本构造器中包含一个参数，使得其值可以与一个或多个正则表达式进行比较。
- 高级脚本和函数（脚本 Cmdlets）可以使用一个基于正则表达式的输入验证工具防止无效的参数值。
- -Match 运算符（在本章简单介绍）将字符串与正则表达式进行对比。还有一部分未做介绍——抓取匹配的字符串存入一个自动的$matches 集合。

PowerShell 使用业界标准的正则表达式。如果你希望更深入地学习，我们推荐你阅读 Jeffrey E.F. Friedl 的著作 *Mastering Regular Expressions* (O'Reilly 出版社，2006)。市场上还有大量关于正则表达式的书籍，其中一部分只面向 Windows 和.NET（也就面向 PowerShell），其中一部分书籍专注针对具体场景构建正则表达式，等等。请浏览你喜欢的在线书店，从而查找是否存在吸引你或满足你特定需求的书籍。

我们也使用免费的在线正则表达式资源：http://RegExLib.com。该网站包含用于不同目的的大量正则表达式示例（电话号码、邮件地址、IP 地址等）。我们还使用 http://RegExTester.com 这个网站测试我们的正则表达式，从而确保正则表达式能够满足我们的需求。

24.7 动手实验答案

1. `dir c:\windows | where {$_.name -match "\d{2}"}`
2. `get-process | where {$_.company -match "^Microsoft"} | Select Name,ID,Company`
3. `get-content C:\Windows\WindowsUpdate.log | Select-string "Start[\w+\W+]+Agent: Installing Updates"`
4. 你可以通过匹配以 1～3 位数字后跟着一个句号为开头的模式获得结果，如下：

   ```
   get-dnsclientcache | where { $_.data -match "^\d{1,3}\."}
   ```

   或者你可以匹配整个 IPv4 地址字符串：

   ```
   get-dnsclientcache | where
   { $_.data -match "^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"}
   ```