

整合 Windows PowerShell

D

APPENDIX

D.1 何謂 PowerShell

Windows PowerShell 是微軟為 Windows 作業系統開發的殼層 (Shell) 及腳本語言¹ (Scripting languages)，它強化了管理 Windows 系統的細部控制與自動化能力。PowerShell 整合數種系統管理工具，以簡易一致的語法，讓管理者處理日常工作，如存取檔案系統、登錄資料庫 (Registry)、Active Directory、查詢各種服務、日誌、WMI/CIM 等。

由於 PowerShell 屬於微軟的 CEC (Common Engineering Criteria 2009) 規範的技術，也就是微軟的伺服器產品都要支援 PowerShell。除了 Windows 作業系統本身的資訊外，微軟先前推出的 Exchange Server 以及 System Center Operations Manager 等伺服器都已內建 PowerShell。

¹

腳本語言 (Script) 通常要簡單易用，讓開發者能快速撰寫，不經編譯流程直接執行。一般用在日常的管理工作，而以往 VBScript/JavaScript 所撰寫的 ASP 網頁則是當作應用程式來開發。

2018 年 PowerShell 以 .NET Core 2.0 為基礎推出 6.0 版，是跨平台的版本，可以執行在 Linux、macOS... 等其他作業系統，從此與單純執行在 Windows 的 PowerShell 5.x 版本分道揚鑣。5.x 版不再有大的更版，其執行檔為 powershell.exe，並以 Windows PowerShell 稱呼 5.x 版。而 6.x 版的至今已更版到 7.1（2022/7 月），執行檔為 pwsh.exe，統稱為 PowerShell，不再帶有 Windows 字樣。若要確認你當下用的是哪一版 PowerShell，可以查詢 `$PSVersionTable` 公共變數。

SQL Server 從 2008 版之後也加入了 PowerShell 功能。現今新的服務、平台與新功能，如管理與存取微軟 SharePoint/Office 365（現統稱為 Microsoft 365）內的 SharePoint、管理 Azure，乃至於 SQL Server 2016 後的一律加密（Always Encrypted）的「加密資料行精靈」所產生的 Script... 等，往往都是先有 PowerShell 模組，再提供圖形化的使用者管理介面。

到了 Azure 等雲平台後，由於管理的數量更大，需動態安裝、自我監測與調整設定的需求更多，因此以腳本語言當管理工具的使用更廣。可以預見的是，系統管理人員將以 PowerShell 來統整所有的服務²。

² 當新人進入／離開公司後，可能要為他新增/移除 AD 帳號、Email 帳號、各種資源的存取權限等，當要整合多種系統時，PowerShell 將是最佳的選擇。

現今微軟已將 PowerShell 拓展到 Linux 等非微軟的環境，期待微軟能成功推廣 PowerShell 到異質型平台(同時也反向地將 Linux 廣用的 ssh 移植到 Windows 平台)，則對龐大、分散、複雜的諸多企業 IT 系統，提供集中化、自動化、流程化、一致化的管理將變得可行。

Windows PowerShell 具有以下特點：

- 統一語法、命名原則，一致性存取各種服務和系統資料。
- PowerShell 的腳本語言豐富強大，且能支援現有的腳本程式和命令列工具，亦即 cmd 殼層與 Windows Script Host 提供的腳本語言（如 VBScript、JavaScript）。
- 內含多種稱為 cmdlet（發音如「command-let」，SQL Server 線上叢書將其翻譯為「指令程式」）的標準工具，可處理常見的系統管理工作。
- 整合各種程式存取介面 COM、WMI/CIM、.NET 技術，具擴充能力，獨立軟體商或開發者能自行擴充。
- 可以透過連接（session）穩定地與遠端系統溝通，完成分散式管理與監控。
- 支援流程（workflow）、背景工作、批次排程、呼叫 Web Service/WebAPI/REST API...等，不管是簡單的批次工作，或複雜的管理流程皆可以完成。
- 內建存取多種格式的資料，如 CSV、XML、JSON 等，並可簡單轉成多種格式輸出，如：HTML、JSON...等。
- 可開發重複使用的模組，Windows 10/Windows Server 2016 後內建的 PowerShell 5.0 版，更支援撰寫類別（Class），讓 PowerShell 的開發人員可以深入地設計架構。
- PowerShell 4.0 後的 Desired State Configuration 提供簡單、穩固、可重複執行、可維持狀態地安裝部署。

Windows 7/Windows 2008 R2 後內建 PowerShell 2.0，換句話說，現今我們一般用的 Windows 作業系統都已內建 PowerShell。

PowerShell 預設提供基本的 cmdlet；可視為提供各種單一功能的程式模組，不同版本提供的 cmdlet 數量不同。而微軟提供的各種服務產品，如 SQL Server、SharePoint、Exchange...等，在安裝時也會安裝該產品的 PowerShell 模組。當選取 Windows 作業系統的角色或功能，如 Active Directory、叢集服務、虛擬化服務...等，會安裝所需的 PowerShell 模組。另外，也可以單獨從微軟下載 Azure、Microsoft/Office 365 等雲端平台的 PowerShell 模組，好從企業內直接控管雲端上的服務。而其他廠商的產品或服務，諸如 Amazon AWS、VMWare、Citrix、Cisco、Quest...等，也在自家產品中加入 PowerShell，提供跨平台的管控。此外，也可以從 GitHub 下載許多開源的模組，例如 SQL Server Reporting Services/PowerBI Reporting Server 可用的 ReportingServicesTools，或管理 SQL Server 可用的 dbaTools...等。

透過 PowerShell 撰寫腳本語言（Script）可隨心所欲地整合商業邏輯與各種技術，不必透過特定的編譯器，以隨手取得的編輯軟體就能修改，隨即執行。符合一般企業 MIS 的應用情境，可滿足大部分的需求。而讓日常作業腳本化、自動化、集中化，才得以標準化與精益求精，也才能因應越來越多樣與複雜的 IT 環境。

D.1.1 使用 cmdlet

PowerShell cmdlet 的命名原則是「動詞-名詞」，其名詞皆使用單數，例如 Get-Help、Get-Process 或 Set-MachineName...等，使用時不分大小寫。我們能夠個別使用 cmdlet，或是串起多個模組來執行複雜的工作。也可自行開發 cmdlet，並讓其他使用者共用。

現今的 Windows 作業系統，多可以點選桌面工具列上的 PowerShell 連結，叫出如圖 D.1 的命令提示列或整合指令碼環境（ISE，其執行檔名為 powershell_ise，可以直接輸入檔名執行）：

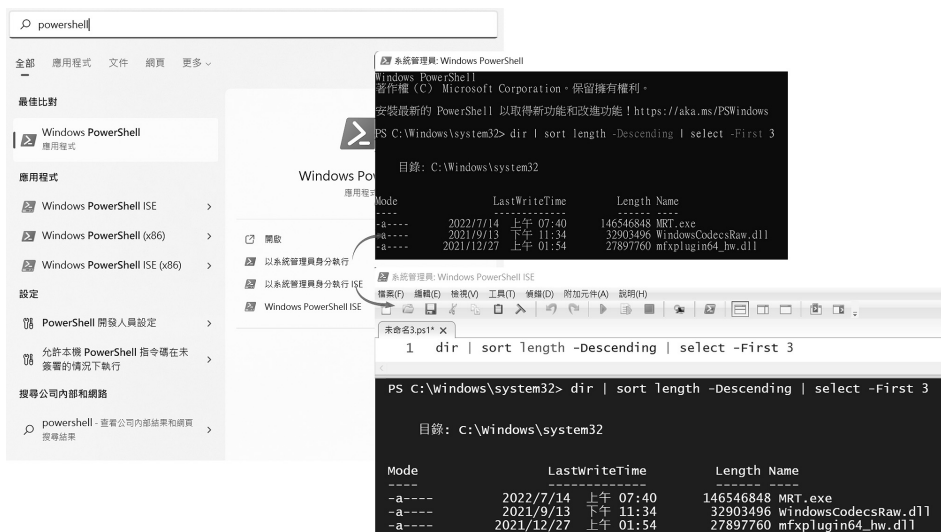


圖 D.1 在 PowerShell 命令提示環境或整合指令碼環境（ISE）執行指令或腳本語言

在圖 D.1 中，列出所在目錄下最大的三個檔案，作法是透過 Get-ChildItem cmdlet（其別名為 Dir）取得檔案目錄資訊，轉交給 Sort-Object cmdlet（其別名為 sort），將資料結果依照 length 屬性，由大到小排序。而後再將排序結果轉給 Select-Object cmdlet（其別名為 select），取回前三筆記錄。

上述的 powershell.exe 和 powershell_ise 內建於 Windows 作業系統，都是針對 5.x 版以下的 powershell。若要採用 powershell 6.x 版以上，微軟採用自家免費的開發工具 Visual Studio Code，下載該軟體後，再加載 powershell 延伸模組，便能夠編輯並單步除錯 powershell script，不需要特定的整合開發環境。

Visual Studio Code 並不侷限於要使用哪個版本的 PowerShell，但編輯時需要切換。按下快捷鍵 `Ctrl+Shift+P` 後輸入「Session」，而後選擇「PowerShell: Show Session Menu」，接著會呈現「Current session」和「Switch to」兩個選項。Current 是當下使用的版本，而 Switch To 則是可以切換至的版本：

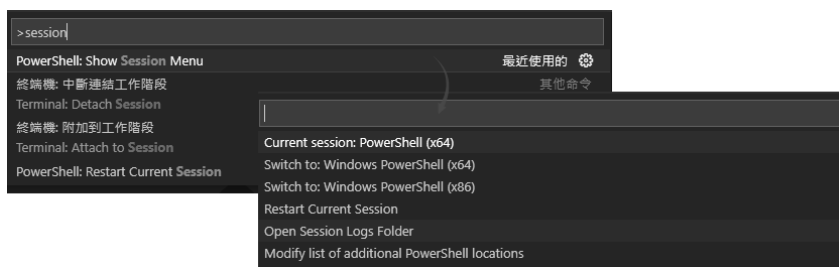


圖 D.2 設定 VS Code 要採用哪個版本的 PowerShell

如圖 D.2 所呈現的，若僅有「PowerShell」字樣，代表是大於 6 之後的版本，而「Windows PowerShell」則是 5 之前的版本。

若想列出與 SQL Server 相關服務的執行狀況，可以簡單執行如下查詢：

```
get-service *sql*
```

針對完整安裝了 SQL Server 各項服務的伺服器，其執行結果如下：

Status	Name	DisplayName
-----	----	-----
Running	MSSQLFDLauncher	SQL Full-text Filter Daemon Launche...
Running	MSSQLLaunchpad	SQL Server Launchpad (MSSQLSERVER)
Running	MSSQLSERVER	SQL Server (MSSQLSERVER)
Running	MSSQLServerOLAP...	SQL Server Analysis Services (MSSQL...
Stopped	SQL Server Dist...	SQL Server Distributed Replay Client
Stopped	SQL Server Dist...	SQL Server Distributed Replay Contr...
Running	SQLBrowser	SQL Server Browser
Stopped	SQLPBDMS	SQL Server PolyBase 資料移動 (MSSQL...

Stopped	SQLPBENGINE	SQL Server PolyBase 引擎 (MSSQLSERVER)
Stopped	SQLSERVERAGENT	SQL Server Agent (MSSQLSERVER)
Running	SQLTELEMETRY	SQL Server CEIP service (MSSQLSERVER)
Running	SQLWriter	SQL Server VSS Writer

透過 Start-Service cmdlet 可以啟動未執行的服務，範例如下：

```
Get-Service 'SQLSERVERAGENT' | Start-Service
```

若想列出 PowerShell 有哪些預設的 cmdlet，可以直接輸入 get-command，列出系統上所有的 cmdlet。透過 get-command 提供的參數，能表列某種類型的 cmdlet，範例如下：

```
#列出名稱有 sql 字樣的 cmdlet  
get-command *sql*  
#列出 sqlps 提供的 cmdlet  
get-command -Module sqlps
```

透過「Help」則可以呈現個別 cmdlet 的說明，它是「get-help」cmdlet 的別名（Alias）。有了 cmdlet 的列表後，可以如下的方式取得個別 cmdlet 的說明：

```
get-help <cmdlet 名稱> -detailed
```

Get-Help 會傳回該 cmdlet 的語法、參數定義、輸入和輸出類型，以及 cmdlet 所執行之動作描述等資訊。範例如圖 D.3 所示：

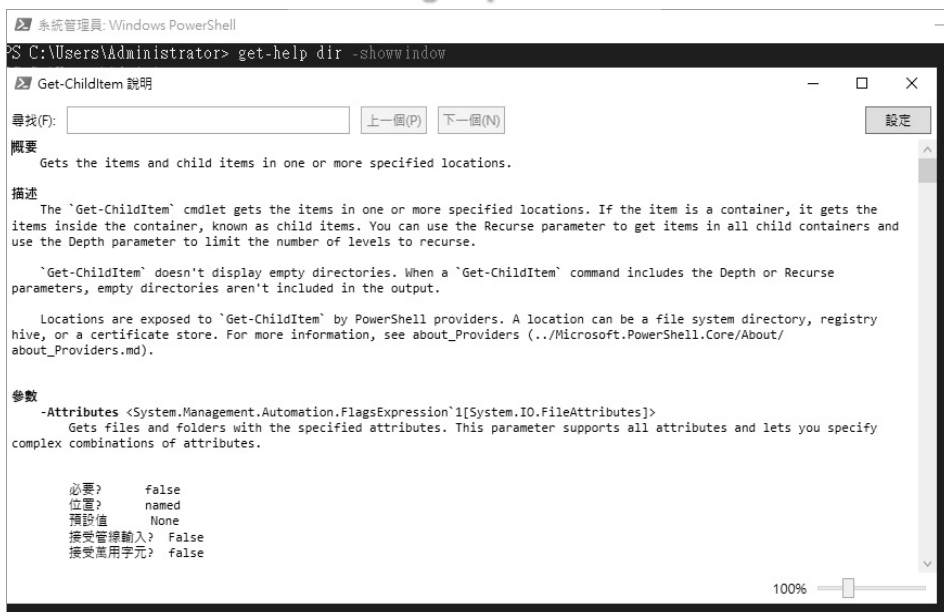


圖 D.3 透過 get-help 檢視各 cmdlet 的說明

在圖 D.3 範例中，簡單查詢 dir 指令的用法，從傳回的資訊可以看到它其實是 Get-ChildItem cmdlet 的別名，而 Get-ChildItem 的用法則詳述在圖 D.3 的下方。由於執行 get-help cmdlet 時搭配了 -ShowWindow 參數，所以採視窗的方式呈現說明，若搭配 online 參數，則會以網頁導覽的方式呈現線上說明。

本章的重點不在於介紹 PowerShell 本身，而是透過 PowerShell 管理 SQL Server。因 PowerShell 功能非常強大，非一兩本書可以說明清楚。若想要深入了解，可以參考 PowerShell 專書。接下來，介紹 SQL Server 所提供與 PowerShell 整合的功能。

D.2 SQL Server 與 PowerShell 的整合

SQL Server 2008 版之後開始提供與 PowerShell 整合的功能，但隨著版本演進，SQL Server 2008/2008 R2 版本所自動安裝的 PowerShell snapin，在 SQL Server 2012 已被 sqlps 模組（module）取代，讓管理者可以在 PowerShell 內存取 SQL Server。

SQL Server 2016 版後改成獨立安裝 Management Studio（SSMS）工具程式，而安裝 SQL Server Management Studio (SSMS) 時，曾經同時安裝 PowerShell 模組 SqlServer。但現今須獨立下載安裝 SqlServer 模組，與安裝 SQL Server 執行個體時同時安裝的 sqlps 不同。依據線上說明，sqlps 模組現今僅是隨附於 SQL Server 安裝，提供回溯相容性但不再更新。

在 SQL Server 引擎/Agent Services 所使用的 PowerShell 模組為 sqlps，因為隨著 SQL Server 安裝，所以一定存在而不怕找不到該模組。但在使用者端執行 SSMS 環境裡呼叫的 PowerShell 模組為 SqlServer。若你安裝完 SSMS，但在本機執行 Import-Module sqlserver 時，回傳找不到有效模組檔案錯誤，或是雖然沒有 SSMS 仍想在本機透過 PowerShell 模組管理 SQL Server，可以直接從網路下載安裝（Install-Module SqlServer）：



圖 D.4 安裝 PowerShell sqlserver 模組



或到此網址下載安裝：

```
https://www.powershellgallery.com/packages/SqlServer/21.1.18256
```

若你的電腦無法上網，也可以從其他能上網並已經安裝 SqlServer 模組的電腦內，直接複製 C:\Program Files\WindowsPowerShell\Modules\SqlServer 目錄內容到自身電腦對應目錄下，即可完成安裝 PowerShell 模組。

相關下載與安裝的詳情可以參照如下網址：

```
https://docs.microsoft.com/zh-tw/sql/powershell/download-sql-server-ps-module?view=sql-server-ver16
```

若要使用這兩個模組，可透過命令提示列呼叫，分別使用 sqlps.exe（使用 Sqlps PowerShell 模組）；或啟用 powershell.exe 執行環境後載入 SqlServer 模組（例如呼叫 Powershell.exe、Powershell_ise.exe 乃至於協力廠商提供的 PowerShell 執行環境），可使用如下語法：

```
Import-Module sqlps
```

或

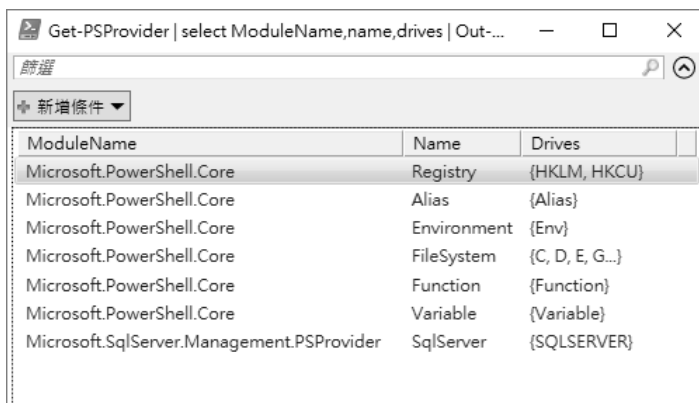
```
Import-Module SqlServer
```

SqlServer 模組提供的功能較 Sqlps 為多，在此先說明兩者共用的部分，包含如下面向：

- SQL Server 提供者（Provider）³：用類似樹狀結構的路徑機制，以 cd、dir 等熟悉的指令來瀏覽物件。也就是在命令提示字元視窗或腳本語法中，用瀏覽目錄的方式檢視 SQL Server 內的物件。並可透過如 ren（rn 別名亦同）命令來更名，或以 del 來刪除物件。簡單用以下的 PowerShell 指令查詢當下所在 PowerShell 環境中的提供者：

```
Import-Module 'sqlps'
Get-PSProvider | select ModuleName,name,drives | Out-GridView
```

其結果如圖 D.5 所示：



ModuleName	Name	Drives
Microsoft.PowerShell.Core	Registry	{HKLM, HKCU}
Microsoft.PowerShell.Core	Alias	{Alias}
Microsoft.PowerShell.Core	Environment	{Env}
Microsoft.PowerShell.Core	FileSystem	{C, D, E, G...}
Microsoft.PowerShell.Core	Function	{Function}
Microsoft.PowerShell.Core	Variable	{Variable}
Microsoft.SqlServer.Management.PSProvider	SqlServer	{SQLSERVER}

圖 D.5 呈現當下 PowerShell 環境所支援的提供者

若沒有載入 SqlServer/sqlps 模組，所呈現的查詢結果將不會出現最下方的 SqlServer 提供者。

- Cmdlet：提供在 PowerShell 指令碼中可用於 SQL Server 相關產品的 cmdlet，例如：透過 Invoke-SQLCmd 命令執行包含 Transact-SQL

³ 在 SQL Server 2008 版時是以獨立的 Snapin 形式提供 SQL Server Provider，名稱為 SqlServerProviderSnapin100。

或 XQuery 陳述式。或是經由 Invoke-PolicyEvaluation 命令；驗證與報告 SQL Server 內的物件集是否符合管理原則（Policy）所定義的條件，或對目標物件重設不符原則的項目。除前述兩者外，尚有編/解碼 SQL Server 識別碼的 Encode-SqlName、Decode-SqlName。以及 Convert-UrnToPath cmdlet，可將 SMO 用於指定物件的 URN 字串轉換成 PowerShell 的路徑表示法，以及操控可用性群組（Availability Group）的多個指令等。

- 支援 Analysis Services 和 Integration Services：SQL Server 2012 後的 sqlps 模組除了包含的上述 SQL Server 2008 版的兩部分外，還擴充了 SQL Server 提供者的功能，使其能存取 Integration Services 以及 Analysis Services。SQL Server 2016 後，支援 Analysis Services 的 Tabular 模型。

Sqlps 模組預設安裝在如下目錄中：

```
C:\Program Files (x86)\Microsoft SQL Server\n\Tools\PowerShell\Modules\SQLPS
```

其中的 n 代表各版 SQL Server，表列最近如下，其餘類推：

- 140：SQL Server 2017
- 150：SQL Server 2019
- 160：SQL Server 2022

SQL Server Management Studio 提供的 SqlServer 模組預設放的位置與 Sqlps 不同，是放在 PowerShell 共用模組的預設位置：

```
C:\Program Files\WindowsPowerShell\Modules\SqlServer
```

存取 SQL Server 服務執行個體時，雖然一般用 SQL Server 各版對應的用戶端元件來執行 SQL Server PowerShell，但它仍可以連接 SQL

Server 2000 SP4 後的各種執行個體。只是當新版 PowerShell 存取過往版本時，會受限於該 SQL Server 版本的功能。

透過 SSMS 可以叫起 PowerShell 的命令提示列環境，啟動後將執行命令的位置自動放在對應的樹狀節點內，如圖 D.6 所示：



圖 D.6 從物件總管的某個節點進入 PowerShell 命令提示環境

圖 D.6 中因為是在 Northwind 範例資料庫節點選擇「啟動 PowerShell」，這時 SSMS 自動幫我們叫起 PowerShell 執行環境，接著出現的命令提示列環境的路徑就落在 Northwind 資料庫內。若尚未安裝 SqlServer 模組，則會出現右上方的錯誤訊息。

另外，若在命令提示列，同樣可以執行 SQLPS.exe/PowerShell.exe 工具程式。不管你是以何種方式進入 PowerShell 的命令提示列環境，執行結果應會類似圖 D.7 所示：

```

系統管理員: C:\Windows\system32\cmd.exe - sqlps
C:\Users\Administrator>sqlps
Microsoft (R) SQL Server (R) PowerShell
版本 16.0.600.9
Copyright (c) 2022 Microsoft. 著作權所有，並保留一切權利。
PS C:\Users\Administrator> cd sqlserver:
PS SQLSERVER:\> dir

Name                Root                Description
-----
SQLRegistration      SQLSERVER:\SQLRegistration  SQL Server 註冊
DAC                  SQLSERVER:\DAC              SQL Server 資料層應用程式元件
DataCollection       SQLSERVER:\DataCollection   SQL Server 資料收集
SQLPolicy             SQLSERVER:\SQLPolicy        SQL Server 原則管理
Utility              SQLSERVER:\Utility          SQL Server 公用程式
SQL                  SQLSERVER:\SQL              SQL Server 資料庫引擎
SSIS                  SQLSERVER:\SSIS             SQL Server Integration Services
XEEvent              SQLSERVER:\XEEvent          SQL Server 擴充事件
DatabaseXEEvent      SQLSERVER:\DatabaseXEEvent  SQL Server 擴充事件

PS SQLSERVER:\> cd sql\sql2022\default\databases
PS SQLSERVER:\sql\sql2022\default\databases> dir

Name                Status              Containment Type Recovery Model CompatLvl Collation              Owner
-----
db                  Normal              None              Full              160 Chinese_Taiwan_Stroke_CI_AS sa
Northwind           Normal              None              Full              160 Chinese_Taiwan_Stroke_CI_AS sa

PS SQLSERVER:\sql\sql2022\default\databases> ren northwind nwind
PS SQLSERVER:\sql\sql2022\default\databases> cd nwind
PS SQLSERVER:\sql\sql2022\default\databases\nwind> invoke-sqlcmd 'create user u without login'
警告: 正在使用提供者內容。伺服器 = sql2022，資料庫 = nwind。
PS SQLSERVER:\sql\sql2022\default\databases\nwind> dir users

Name                Login              Created
-----
u                    u                    2022/7/27 下午 02:17

PS SQLSERVER:\sql\sql2022\default\databases\nwind> del users\u
PS SQLSERVER:\sql\sql2022\default\databases\nwind>

```

圖 D.7 透過 SQL Server 提供的 PowerShell 模組，可讓系統管理者以熟悉的指令管理 SQL Server

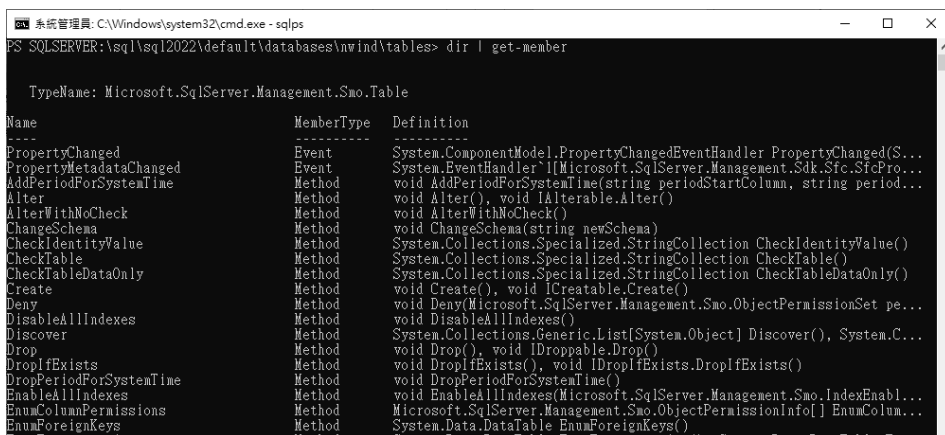
在圖 D.7 中，簡單示範利用 SQL Server 提供的 PowerShell 模組，依樹狀結構瀏覽與變更物件。首先以 Dir 呈現 Provider 提供哪些內容。藉由 Cd 進入到某個執行個體及其資料庫的子目錄後，此處為「SQL 資料庫引擎\sql2022 伺服器\default 執行個體\資料庫」。再以 Dir 呈現某個資料庫內的物件類型，此處呈現所有的資料庫。而後透過 Ren 更名資料庫名稱，從 Northwind 更名成 nwind。接著透過 Cd 進入 nwind 資料庫子目錄，以 Invoke-SqlCmd 執行 T-SQL 語法，建立一個名稱為 u 的使用者，最後使用 Del 刪除資料庫內的使用者。

當 PowerShell 透過管線（Pipe；其符號表示為「|」）在 cmdlet 間轉交資訊時，傳遞的都是物件，舉例而言，在 Management Studio 的物

件總管視窗內，以滑鼠右鍵點選「資料庫」→「Northwind」→「資料表」節點，按滑鼠右鍵選擇「啟動 PowerShell」選項，並在隨後的「SQL Server Powershell」命令提示視窗執行如下的語法：

```
dir | get-member
```

其執行結果如圖 D.8 所示：



```

System.Management.Automation PS SQLSERVER:\sql\sql2022\default\databases\nwind\tables> dir | get-member

TypeName: Microsoft.SqlServer.Management.Smo.Table
-----
Name                                     MemberType  Definition
-----
PropertyChanged                        Event       System.ComponentModel.PropertyChangedEventHandler PropertyChanged(S...
PropertyMetadataChanged               Event       System.EventHandler<I[Microsoft.SqlServer.Management.Sdk.Sfc.SfcPro...
AddPeriodForSystemTime                Method      void AddPeriodForSystemTime(string periodStartColumn, string period...
Alter                                  Method      void Alter(), void IAlterable.Alter()
AlterWithNoCheck                      Method      void AlterWithNoCheck()
ChangeSchema                          Method      void ChangeSchema(string newSchema)
CheckIdentityValue                    Method      System.Collections.Specialized.StringCollection CheckIdentityValue()
CheckTable                            Method      System.Collections.Specialized.StringCollection CheckTable()
CheckTableDataOnly                    Method      System.Collections.Specialized.StringCollection CheckTableDataOnly()
Create                                Method      void Create(), void ICreatable.Create()
Deny                                  Method      void Deny(Microsoft.SqlServer.Management.Smo.ObjectPermissionSet pe...
DisableAllIndexes                     Method      void DisableAllIndexes()
Discover                              Method      System.Collections.Generic.List[System.Object] Discover(), System.C...
Drop                                  Method      void Drop(), void IDroppable.Drop()
DropIfExists                          Method      void DropIfExists(), void IDropIfExists.DropIfExists()
DropPeriodForSystemTime               Method      void DropPeriodForSystemTime()
EnableAllIndexes                      Method      void EnableAllIndexes(Microsoft.SqlServer.Management.Smo.IndexEnabl...
EnumColumnPermissions                 Method      Microsoft.SqlServer.Management.Smo.ObjectPermissionInfo[] EnumColum...
EnumForeignKeys                       Method      System.Data.DataTable EnumForeignKeys()
    
```

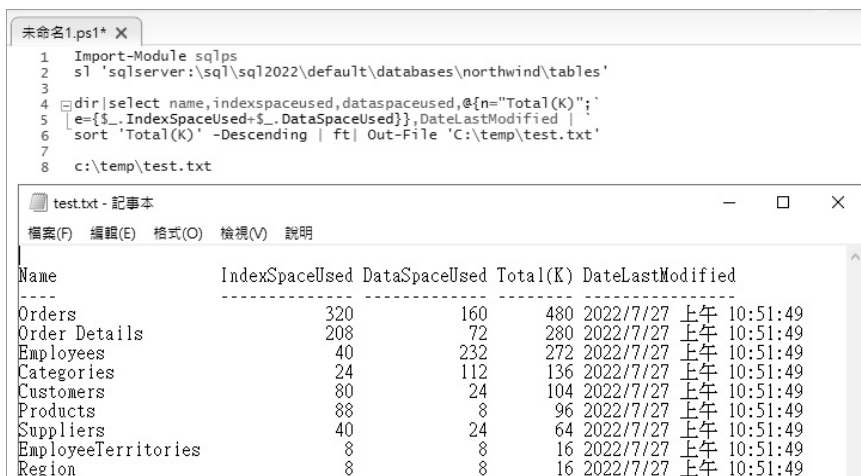
圖 D.8 在資料表節點下查詢到為 Microsoft.SqlServer.Management.Smo.Table 物件

從圖 D.8 可以看到 Dir Cmdlet 實際傳給 Get-Member（別名為 gm）cmdlet 的是 Microsoft.SqlServer.Management.Smo.Table 物件的執行個體，由此可以推測這些 PowerShell Module 是包裝以 .NET 開發的 Sql Server Management Objects（SMO）物件。透過 Get-Member Cmdlet 可以檢視該物件提供的屬性、方法和事件。接著，示範下述語法將各資料表的 name, indexspaceused, dataspaceused, DateLastModified 屬性形成之報表，依照使用量大小排序，直接以文字格式輸出：

```

dir|select name,indexspaceused,dataspaceused,@{n="Total(K)";`
e={$_.IndexSpaceUsed+$_.DataSpaceUsed}},DateLastModified | `
sort Total(K) -Descending | ft| Out-File 'C:\temp\test.txt'
    
```

其執行結果如圖 D.9 所示：



```

1 Import-Module sqlps
2 sl 'sqlserver:\sql\sql2022\default\databases\northwind\tables'
3
4 dir | select name, indexspaceused, dataspaceused, @{n="Total(K)";
5   e={$_.IndexSpaceUsed+$_.DataSpaceUsed}}, DateLastModified |
6 sort 'Total(K)' -Descending | ft | Out-File 'C:\temp\test.txt'
7
8 c:\temp\test.txt
  
```

Name	IndexSpaceUsed	DataSpaceUsed	Total(K)	DateLastModified
Orders	320	160	480	2022/7/27 上午 10:51:49
Order Details	208	72	280	2022/7/27 上午 10:51:49
Employees	40	232	272	2022/7/27 上午 10:51:49
Categories	24	112	136	2022/7/27 上午 10:51:49
Customers	80	24	104	2022/7/27 上午 10:51:49
Products	88	8	96	2022/7/27 上午 10:51:49
Suppliers	40	24	64	2022/7/27 上午 10:51:49
EmployeeTerritories	8	8	16	2022/7/27 上午 10:51:49
Region	8	8	16	2022/7/27 上午 10:51:49

圖 D.9 查詢資料表的各種屬性，並將結果以 txt 格式輸出

這些與資料表相關的資訊會散在不同的系統檢視內，若透過 T-SQL 查詢，需要了解如何 Join 不同的系統檢視，並計算欄位內容得出上述的結果。對於 Windows/網路的系統管理者但非 DBA 而言，勢必困難許多，Sqlps/SqlServer 則提供了簡單的物件屬性，讓一般的系統管理者可以直觀地查詢。

接著來看 SQL Server 引擎/SSMS 為整合 PowerShell 提供的模組之主要功能。

D.2.1 SQL Server 提供者

「SQL Server PowerShell 提供者（Provider）」支援類似於檔案路徑的 SQL Server 物件階層。讓你以階層路徑來尋找物件，然後使用 SQL Server 管理物件（SMO）提供的方法操作。其實，各種 PowerShell「提供者」都會實作一個以上類似磁碟機的存取架構，每個磁碟機都是相關物件階層的根節點。「SQL Server PowerShell 提供者」實作「SQLSERVER:」

磁碟機。該磁碟機根目錄具有九個子目錄。每個目錄及其子目錄都代表可用的 SQL Server 物件集。

當進入某個目錄節點時，就可以使用相關物件的方法。PowerShell 也讓使用者定義稱為 PowerShell 磁碟機或 PSDrives 的虛擬磁碟機。這些磁碟機是路徑陳述式的開始節點。SQL Server PowerShell 提供者實作的目錄如表 D.1 所示：

表 D.1 SQL Server PowerShell 提供者預設實作的 PowerShell 目錄

目錄	物件
SQLSERVER:\SQL	資料庫物件，例如資料表、檢視表和預存程序
SQLSERVER:\SQLPolicy	以原則為基礎的管理物件，例如原則和 Facet
SQLSERVER:\SQLRegistration	已註冊的伺服器物件，例如，伺服器群組和已註冊的伺服器
SQLSERVER:\DataCollection	效能資料收集
SQLSERVER:\XEvent	擴充事件
SQLSERVER:\DatabaseXEvent	資料庫層級擴充事件（線上說明語焉不詳，實際透過 cd 等 cmdlet 存取時，僅有類似不支援的訊息）
SQLSERVER:\Utility	公用程式
SQLSERVER:\DAC	資料層應用程式元件
SQLSERVER:\SSIS	Integration Services 資料整合服務
SQLSERVER:\SQLAS	Analysis Services 資料（SqlServer 模組有而 sqlps 模組沒有）

其說明可以參照：<https://docs.microsoft.com/zh-tw/sql/powershell/sql-server-powershell-provider?view=sql-server-ver16>

簡單的查詢範例如圖 D.10 所示：

```
PS SQLSERVER:\> dir
```

Name	Root	Description
----	----	-----
DAC	SQLSERVER:\DAC	SQL Server 資料層應用程式元件
DataCollection	SQLSERVER:\DataCollection	SQL Server 資料收集
SQLPolicy	SQLSERVER:\SQLPolicy	SQL Server 原則管理
Utility	SQLSERVER:\Utility	SQL Server 公用程式
SQLRegistration	SQLSERVER:\SQLRegistration	SQL Server 註冊
SQL	SQLSERVER:\SQL	SQL Server 資料庫引擎
SSIS	SQLSERVER:\SSIS	SQL Server Integration Services
XEvent	SQLSERVER:\XEvent	SQL Server 擴充事件
DatabaseXEvent	SQLSERVER:\DatabaseXEvent	SQL Server 擴充事件
SQLAS	SQLSERVER:\SQLAS	SQL Server Analysis Services

圖 D.10 查詢 SQL Server PowerShell 提供者預設提供的目錄

其中，SQLSERVER:\SQL 目錄可當作 SMO 物件模型的路徑開頭，前置部分如下：

```
SQLSERVER:\SQL\電腦名稱\執行個體名稱
```

路徑中，一定要指定執行個體名稱。如果是存取預設執行個體，則路徑名稱設為「DEFAULT」。執行個體名稱之後的節點會在物件類別集合（如 Databases 或 Views）和物件名稱（如 Northwind）之間輪替。另外，結構描述（Schema）不是物件類別。在結構描述中指定物件節點（如資料表或檢視表）時，必須使用 <SchemaName>.<ObjectName> 格式來指定物件名稱。以存取伺服器名稱為 SQL2022，SQL Server 執行個體為預設執行個體，資料庫為 Northwind，結構描述為 dbo，資料表名稱為 Employees 之 PowerShell 路徑如下：

```
SQLSERVER:\sql\sql2022\DEFAULT\DATABASES\Northwind\TABLES\dbo.employees
```

當存取路徑中資料庫引擎執行個體時，SQL Server PowerShell 提供者使用 SMO 開啟 Windows 驗證連接，憑藉當下執行 PowerShell 工作階段的 Windows 帳戶，而不會使用 SQL Server 驗證。

PowerShell 藉由 cmdlet 來瀏覽物件階層與執行作業，由於 cmdlet 經常被使用，所以設計了簡短、標準的別名。並定義一組別名將 cmdlet 對應到傳統的「命令提示字元」命令，且有另一組別名適用於 UNIX Shell

命令。SQL Server PowerShell 提供者實作的 cmdlet 與別名如表 D.2 所示：

表 D.2 SQL Server PowerShell 提供者預設提供的 cmdlet

cmdlet	標準的別名	cmd 別名	說明
Get-Location	gl	pwd	取得目前的節點。
Set-Location	sl	cd、chdir	變更目前的節點到目的目錄。
Get-ChildItem	gci	dir	列出儲存在目前節點上的物件。
Get-Item	gu		傳回目前項目的屬性。
Move-Item	mi	move	移動物件。
Rename-Item	rni	rn、ren	重新命名物件。
Remove-Item	ri	del、rd	移除物件。

雖然 SQL Server 提供者有實作表 D.2 的 cmdlet，但畢竟 SQL Server 樹狀階層內的物件不是如目錄檔案結構那麼簡單，若要細緻地操作各種 SQL 物件，還是需要利用個別物件的屬性、方法，再搭配其他的 cmdlet 和 T-SQL 才行。

D.2.2 SQL Server cmdlet

除了表 D.2 所列，由 SQL Server 提供者所支援的 cmdlet，以存取 SQL Server 物件階層結構外，SQL Server PowerShell 尚有一組專門的 cmdlet，以操作 SQL Server。透過如下的指令可以得到簡單的說明，本節先以 Sqlps 為主，下一節再討論 SqlServer 較 Sqlps 為多的部分：

```
Get-command -module SQLPS
```

執行結果如圖 D.11：

```
PS SQLSERVER:\databaseevent\sql2022> Get-command -module SQLPS
```

CommandType	Name	Version	Source
Alias	Decode-SqlName	16.0	SOLPS
Alias	Encode-SqlName	16.0	SOLPS
Function	SQLSERVER:	16.0	SOLPS
Cmdlet	Add-SqlAvailabilityDatabase	16.0	SOLPS
Cmdlet	Add-SqlAvailabilityGroupListenerStaticIp	16.0	SOLPS
Cmdlet	Backup-SqlDatabase	16.0	SOLPS
Cmdlet	ConvertFrom-EncodedSqlName	16.0	SOLPS
Cmdlet	ConvertTo-EncodedSqlName	16.0	SOLPS
Cmdlet	Convert-UrnToPath	16.0	SOLPS
Cmdlet	Disable-SqlAlwaysOn	16.0	SOLPS
Cmdlet	Enable-SqlAlwaysOn	16.0	SOLPS
Cmdlet	Get-SqlCredential	16.0	SOLPS
Cmdlet	Get-SqlDatabase	16.0	SOLPS
Cmdlet	Get-SqlSmartAdmin	16.0	SOLPS
Cmdlet	Invoke-PolicyEvaluation	16.0	SOLPS
Cmdlet	Invoke-Sqlcmd	16.0	SOLPS
Cmdlet	Join-SqlAvailabilityGroup	16.0	SOLPS
Cmdlet	New-SqlAvailabilityGroup	16.0	SOLPS
Cmdlet	New-SqlAvailabilityGroupListener	16.0	SOLPS
Cmdlet	New-SqlAvailabilityReplica	16.0	SOLPS
Cmdlet	New-SqlBackupEncryptionOption	16.0	SOLPS
Cmdlet	New-SqlCredential	16.0	SOLPS
Cmdlet	New-SqlHADREndpoint	16.0	SOLPS
Cmdlet	Remove-SqlAvailabilityDatabase	16.0	SOLPS
Cmdlet	Remove-SqlAvailabilityGroup	16.0	SOLPS
Cmdlet	Remove-SqlAvailabilityReplica	16.0	SOLPS
Cmdlet	Remove-SqlCredential	16.0	SOLPS
Cmdlet	Restore-SqlDatabase	16.0	SOLPS
Cmdlet	Resume-SqlAvailabilityDatabase	16.0	SOLPS
Cmdlet	Set-SqlAvailabilityGroup	16.0	SOLPS
Cmdlet	Set-SqlAvailabilityGroupListener	16.0	SOLPS
Cmdlet	Set-SqlAvailabilityReplica	16.0	SOLPS
Cmdlet	Set-SqlCredential	16.0	SOLPS
Cmdlet	Set-SqlHADREndpoint	16.0	SOLPS
Cmdlet	Set-SqlSmartAdmin	16.0	SOLPS
Cmdlet	Suspend-SqlAvailabilityDatabase	16.0	SOLPS
Cmdlet	Switch-SqlAvailabilityGroup	16.0	SOLPS
Cmdlet	Test-SqlAvailabilityGroup	16.0	SOLPS
Cmdlet	Test-SqlAvailabilityReplica	16.0	SOLPS
Cmdlet	Test-SqlDatabaseReplicaState	16.0	SOLPS
Cmdlet	Test-SqlSmartAdmin	16.0	SOLPS

圖 D.11 檢視 SQLPS 提供的指令

針對圖 D.11 中的部分 cmdlet 說明如下：

❖ Invoke-Sqlcmd

Invoke-Sqlcmd 支援 sqlcmd 工具程式的指令碼，以及 Transact-SQL 或 XQuery 陳述式。它可接受 sqlcmd.exe 工具程式的輸入參數；或是要開啟的指令碼檔案名稱。

❖ Invoke-PolicyEvaluation

Invoke-PolicyEvaluation 會回報 SQL Server 物件集合是否符合管理原則定義的條件。也可以使用此 cmdlet，對目標物件重新設定不符合原則條件的選項。在之後的範例中，將包含上述兩個 cmdlet，在此就不再另做示範。

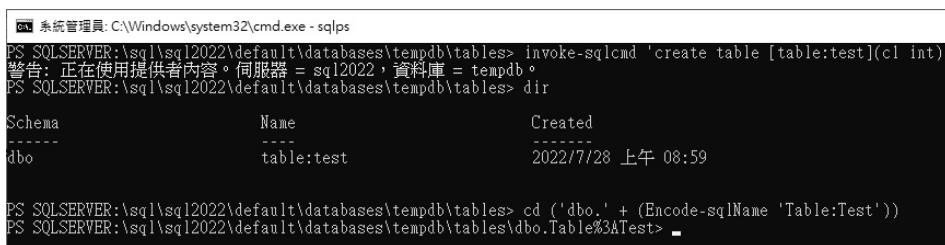
❖ 編／解碼 SQL Server 物件識別碼

「識別碼」意指 SQL Server 物件的名稱，其使用的字元範圍比 PowerShell 腳本語言廣。資料庫引擎對於物件名稱所用的字元很少限制。其中某些字元不能靠 PowerShell 逸出字元「`」來規避，造成使用「SQL Server PowerShell 提供者」時可能會發生問題。

Encode-SqlName cmdlet 會將 SQL Server 識別碼當作輸入，重新格式化 PowerShell 語言不支援的字元，轉成可用在 PowerShell 的表示法。Decode-SqlName cmdlet 則將編碼的 SQL Server 識別碼當作輸入，並傳回原始識別碼。例如：

- Encode-SqlName "Table:Test"會傳回"Table%3ATest"。
- Decode-SqlName "Table%3ATest"會傳回"Table:Test"。

其應用範例如圖 D.12 所示：



```

系統管理員: C:\Windows\system32\cmd.exe - sqlps
PS SQLSERVER:\sql\sql2022\default\databases\tempdb\tables> invoke-sqlcmd 'create table [table:test](cl int)'
警告: 正在使用提供者內容。伺服器 = sql2022, 資料庫 = tempdb。
PS SQLSERVER:\sql\sql2022\default\databases\tempdb\tables> dir

Schema      Name                      Created
-----
dbo          table:test                2022/7/28 上午 08:59

PS SQLSERVER:\sql\sql2022\default\databases\tempdb\tables> cd ('dbo.' + (Encode-sqlName 'Table:Test'))
PS SQLSERVER:\sql\sql2022\default\databases\tempdb\tables\dbo.Table%3ATest>
  
```

圖 D.12 透過 Encode-SqlName cmdlet 編碼 SQL Server 物件識別碼

在圖 D.12 中，先透過 Invoke-Sqlcmd cmdlet 執行如下的 T-SQL 語法，建立一個名為 Table:Test 的資料表：

```
Invoke-Sqlcmd CREATE TABLE [Table:Test] (C1 INT)'
```

由於資料表的名稱包含 PowerShell 的關鍵字冒號「:」，讓你無法使用如下的語法：

```
cd dbo.[Table:Test]
```

會有如下的錯誤：

```
Set-Location : 找不到磁碟機。名為 'dbo.[Table]' 的磁碟機不存在。  
位於 行:1 字元:3  
+ cd <<<< dbo.[Table:Test]
```

必須要透過 Encode-SqlName cmdlet 來編碼，範例如下：

```
cd ('dbo.' + (Encode-SqlName 'Table:Test'))
```

❖ 將 URN 轉換成路徑

SQL Server 管理物件（SMO）會賦予它的物件「統一資源名稱（URN）」。每個 URN 都有與物件所在路徑相同的資訊，但格式不同。例如，以下為資料表在 PowerShell 的路徑：

```
SQLSERVER:\SQL\MyComputer\DEFAULT\Databases\AdventureWorks\Tables\Person.Address
```

以下是相同物件在 SMO 的 URN 表示法：

```
Server[@Name='MyComputer']\Database[@Name='AdventureWorks']\Table[@Name='Address' and @Schema='Person']
```

Convert-UrnToPath cmdlet 會將 SMO URN 字串轉成 PowerShell 路徑。如果節點名稱包含 PowerShell 路徑名稱不支援的擴充字元，Convert-UrnToPath 會以 16 進位表示法編碼這些字元。例如，「"My:Table"」會以「"My%3ATable"」形式傳回。

❖ 建立與管理可用性群組

下列 PowerShell 範例會建立及設定一個名為 PowerShellAG 的簡單可用性群組，其內包含三個可用性複本（N1、N2 和 N3）和兩個可用性資料庫（Northwind,myDB）。

範例程式 D.1：透過 PowerShell 提供的 cmdlet 建立可用性群組

```
<#
範例步驟：
建立個別複本上的端點
備份 MyDatabase 及其交易記錄。
使用 MyDatabase 選項還原 -NoRecovery 和其交易記錄。
建立主要複本，此主要複本將由 SQL Server 本機執行個體（名為 N1\Default）所裝載。
建立次要複本，此次要複本將由 SQL Server 執行個體（名為 N2\Default, N3\Default）所裝載。
建立名為 PoserShellAG 的可用性群組。
將次要複本聯結至可用性群組。
將次要資料庫加入可用性群組。
#>
param(
[string]$AgName='PowerShellAG'
[string[]]$DBs= @('Northwind','myDB')
[string]$ShareFolder='\\N1\DB'
[string]$MainInstance='N1'
[string[]]$Replicas=@('N2','N3')
[string]$Domain='i.com'
[int]$Port=5022
[bool]$CreateEndpoint=$true #定義是否要建 EndPoint
)
Import-Module sqlps
$DomainPort=$Domain + ':' + $Port.ToString()
# 對每一個執行個體建立端點
if($CreateEndpoint -eq $true)
{
```

```

$endpoint = New-SqlHadrEndpoint AG -Port $Port -Path "SQLSERVER:\SQL\
$MainInstance\Default"
# 啟動 EndPoint
Set-SqlHadrEndpoint -InputObject $endpoint -State "Started"
foreach($i in $Replicas)
{
    $endpoint = New-SqlHadrEndpoint AG -Port $Port -Path "SQLSERVER:\SQL\
$i\Default"
    # 啟動 EndPoint
    Set-SqlHadrEndpoint -InputObject $endpoint -State "Started"
}
}
# 完整與紀錄備份主要執行個體的資料庫
foreach($db in $DBs)
{
    Backup-SqlDatabase -Database $db -BackupFile "$ShareFolder\$db.bak"
-ServerInstance $MainInstance -Initialize
    Backup-SqlDatabase -Database $db -BackupFile "$ShareFolder\$db.log"
-ServerInstance $MainInstance -BackupAction Log -Initialize
}
'完成備份'
# 使用 No Recovery 還原到其他參與 AG 的執行個體
foreach($i in $Replicas)
{
    foreach($db in $DBs)
    {
        Restore-SqlDatabase -Database $db -BackupFile "$ShareFolder\$db.bak" `
        -ServerInstance $i -NoRecovery
        Restore-SqlDatabase -Database $db -BackupFile "$ShareFolder\$db.log" `
        -ServerInstance $i -RestoreAction Log -NoRecovery
    }
}
'完成還原'
# 建立主要副本定義，AVersion 參數對應的是 SQL 主版號
$primaryReplica = New-SqlAvailabilityReplica -Name $MainInstance -EndpointURL
"TCP://$MainInstance.$DomainPort" `
-AvailabilityMode 'SynchronousCommit' -FailoverMode 'Automatic' -Version 15
-AsTemplate

# 建立次要副本定義
[System.Collections.ArrayList]$ReplicaTemplates=New-Object
System.Collections.ArrayList
[void]$ReplicaTemplates.Add($primaryReplica)
foreach($i in $Replicas)

```



```
{
    $secondaryReplica = New-SqlAvailabilityReplica -Name $i -EndpointURL
    "TCP://$i.$DomainPort" `
        -AvailabilityMode "SynchronousCommit" -FailoverMode "Automatic" -Version
    15 -AsTemplate
    [void]$ReplicaTemplates.Add($secondaryReplica)
}
'完成主/次要副本定義'
# 建立 availability group
New-SqlAvailabilityGroup -Name $AGName -Path
"SQLSERVER:\SQL\$MainInstance\default" `
    -AvailabilityReplica $ReplicaTemplates -Database $DBs
'主要副本建立 AG'
# 將次要副本加入到 availability group
foreach($i in $Replicas)
{
    Join-SqlAvailabilityGroup -Path "SQLSERVER:\SQL\$i\default" -Name $AGName
    n-SqlAvailabilityGroup -Path "SQLSERVER:\SQL\$i\default" -Name $AGName
    # 將次要資料庫加入到 availability group
    Add-SqlAvailabilityDatabase -Path
    "SQLSERVER:\SQL\$i\default\AvailabilityGroups\$AGName" -Database $DBs
}
'次要副本加入 AG 並同步資料'
```

透過 SSMS 檢視上述範例建立的
的可用性群組如圖 D.13 所示：



圖 D.13 以 SSMS 檢視可用性群組的狀態

透過 SSMS 可以滑鼠右鍵點選某個可用性群組的主要節點，而後選擇「顯示儀表板」選項，便可以觀察可用性群組當下的狀況。另外，也可以透過如下的 PowerShell 查詢：

範例程式 D.2：透過 PowerShell 檢查可用性群組各組成元素的健康情形

```
Param(
    $AgName='PowerShellAG'
)
#確定可用性群組的 Primary Replica 是哪一個節點，在該節點上查詢才有意義。要只取名稱字串，所以取到 OwnerNode 物件後，還要取其內的屬性
$Node=(Get-ClusterResource | where name -EQ $AgName | select OwnerNode).
OwnerNode.Name

Out-Default -InputObject '可用性群組的健全狀況：Test-SqlAvailabilityGroup'
Get-ChildItem "SQLSERVER:\Sql\$Node\default\AvailabilityGroups" |
Test-SqlAvailabilityGroup #| Where-Object { $_.HealthState -eq "Error" }

Out-Default -InputObject '可用性複本的健全狀況：Test-SqlAvailabilityReplica'
dir
"SQLSERVER:\sql\$Node\default\availabilitygroups\$AgName\availabilityreplicas"
| Test-SqlAvailabilityReplica | ft

Out-Default -InputObject '所有聯結可用性複本之可用性資料庫的健全狀況：
Test-SqlDatabaseReplicaState'
dir
"SQLSERVER:\sql\$Node\default\availabilitygroups\$AgName\DatabaseReplicaStates
" | Test-SqlDatabaseReplicaState | ft
```

上述範例的執行結果如圖 D.14 所示：

```

系統管理員: Windows PowerShell
PS C:\ps> .\checkAg.ps1
可用性群組的健全狀況：Test-SqlAvailabilityGroup

HealthState      Name
-----
Healthy          PowerShellAG
可用性複本的健全狀況：Test-SqlAvailabilityReplica

HealthState      AvailabilityGroup  Name
-----
Healthy          PowerShellAG       N1
Healthy          PowerShellAG       N2
Healthy          PowerShellAG       N3

所有聯結可用性複本之可用性資料庫的健全狀況：Test-SqlDatabaseReplicaState

HealthState      AvailabilityGroup  AvailabilityReplica  Name
-----
Healthy          PowerShellAG       N1                  myDB
Healthy          PowerShellAG       N1                  Northwind
Healthy          PowerShellAG       N2                  myDB
Healthy          PowerShellAG       N2                  Northwind
Healthy          PowerShellAG       N3                  myDB
Healthy          PowerShellAG       N3                  Northwind
    
```

圖 D.14 以 PowerShell 查詢「可用性群組」的狀態

圖 D.14 呈現可用性群組、群組內所有的複本、各複本內資料庫都是正常的。

D.2.3 SqlServer PowerShell 模組

由於伺服器與使用者端分開安裝，自然有些使用者端需要的功能不見得伺服器端的 SQL Server 引擎需要，或是 SQL Server 引擎也不像管理工具頻繁更新⁴，造成引擎的不穩定或不相容。因此，目前看到 SSMS 提供的「SqlServer」PowerShell 模組內含的 cmdlet 較 SQL Server 引擎提供的 Sqlps 模組為多。透過 get-command 查詢如圖 D.15 所示：

⁴ 由於 SSMS 都可能有更新，因此我們撰寫本章時，能抓的是 2022/7 月更新，而參照的文件是 2022/7 月。當你在閱讀本章時，可能看到 SSMS 提供的 SQLServer powershell 模組會稍有不同。

```

SQL Server Powershell
PS C:\> Get-Command -Module sqlserver | sort Name | Format-Wide -Column 2

Add-RoleMember                                Add-SqlAvailabilityDatabase
Add-SqlAvailabilityGroupListenerStaticIp      Add-SqlAzureAuthenticationContext
Add-SqlColumnEncryptionKeyValue              Add-SqlFirewallRule
Add-SqlLogin                                  Backup-ASDatabase
Backup-SqlDatabase                           Complete-SqlColumnMasterKeyRotation
ConvertFrom-EncodedSqlName                   ConvertTo-EncodedSqlName
Convert-UrnToPath                             Decode-SqlName
Disable-SqlAlwaysOn                          Enable-SqlAlwaysOn
Encode-SqlName                                Export-SqlVulnerabilityAssessmentBaselineSet
Export-SqlVulnerabilityAssessmentScan         Get-SqlAgent
Get-SqlAgentJob                               Get-SqlAgentJobHistory
Get-SqlAgentJobSchedule                      Get-SqlAgentJobStep
Get-SqlAgentSchedule                         Get-SqlAssessmentItem
Get-SqlBackupHistory                         Get-SqlColumnEncryptionKey
Get-SqlColumnMasterKey                       Get-SqlCredential
Get-SqlDatabase                             Get-SqlErrorLog
Get-SqlInstance                             Get-SqlLogin
Get-SqlSensitivityClassification              Get-SqlSensitivityRecommendations
Get-SqlSmartAdmin                           Grant-SqlAvailabilityGroupCreateAnyDatabase
Import-SqlVulnerabilityAssessmentBaselineSet Invoke-ASCmd
Invoke-PolicyEvaluation                      Invoke-ProcessASDatabase
Invoke-ProcessCube                          Invoke-ProcessDimension
Invoke-ProcessPartition                     Invoke-ProcessTable
Invoke-SqlAssessment                        Invoke-Sqlcmd
Invoke-SqlColumnMasterKeyRotation            Invoke-SqlNotebook
Invoke-SqlVulnerabilityAssessmentScan        Join-SqlAvailabilityGroup
Merge-Partition                             New-RestoreFolder
New-RestoreLocation                         New-SqlAvailabilityGroup
New-SqlAvailabilityGroupListener              New-SqlAvailabilityGroupReplica
New-SqlAzureKeyVaultColumnMasterKeySettings New-SqlBackupEncryptionOption
New-SqlCertificateStoreColumnMasterKeySettings New-SqlCngColumnMasterKeySettings
New-SqlColumnEncryptionKey                  New-SqlColumnEncryptionKeyEncryptedValue
New-SqlColumnEncryptionSettings              New-SqlColumnMasterKey
New-SqlColumnMasterKeySettings               New-SqlCredential
New-SqlCspColumnMasterKeySettings            New-SqlHADREndpoint
New-SqlVulnerabilityAssessmentBaselineSet    New-SqlVulnerabilityAssessmentBaselineSet
Read-SqlTableData                           Read-SqlViewData
Read-SqlXEEvent                             Remove-RoleMember
Remove-SqlAvailabilityDatabase                Remove-SqlAvailabilityGroup
Remove-SqlAvailabilityReplica                 Remove-SqlColumnEncryptionKey
Remove-SqlColumnEncryptionKeyValue            Remove-SqlColumnMasterKey
Remove-SqlCredential                         Remove-SqlFirewallRule
Remove-SqlLogin                             Remove-SqlSensitivityClassification
Restore-ASDatabase                          Restore-SqlDatabase
Resume-SqlAvailabilityDatabase                Revoke-SqlAvailabilityGroupCreateAnyDatabase
Save-SqlMigrationReport                     Set-SqlAuthenticationMode
Set-SqlAvailabilityGroup                     Set-SqlAvailabilityGroupListener
Set-SqlAvailabilityReplica                   Set-SqlAvailabilityReplicaRoleToSecondary
Set-SqlColumnEncryption                     Set-SqlCredential
Set-SqlErrorLog                             Set-SqlHADREndpoint
Set-SqlNetworkConfiguration                 Set-SqlSensitivityClassification
Set-SqlSmartAdmin                           SQLSERVER:
Start-SqlInstance                           Stop-SqlInstance
Suspend-SqlAvailabilityDatabase               Switch-SqlAvailabilityGroup
Test-SqlAvailabilityGroup                    Test-SqlAvailabilityReplica
Test-SqlDatabaseReplicaState                 Test-SqlSmartAdmin
Write-SqlTableData
  
```

圖 D.15 SqlServer 模組提供的 cmdlet 列表

在此透過 PowerShell Format-Wide cmdlet 搭配 -Column 2 參數，將 SqlServer 模組提供的 cmdlet 以兩欄方式呈現。相較於圖 D.11，可以約略得知 SSMS 提供的 SqlServer PowerShell 模組比 SQL Server 引擎提供的 PowerShell 模組多了存取 Azure、Analysis Services、Agent Job、Error

Log 和一律加密⁵...等功能。例如透過 Get-SqlErrorLog cmdlet 可以簡單地讀取某個時間之後，SQL Server 寫到 errorlog 的紀錄，範例語法如下：

```
$Credential = New-Object -VTypeName  
System.Management.Automation.PSCredential -VArgumentList 'byron',  
( ConvertTo-SecureString "!QAZxsw2" -VAsPlainText -Force)  
Get-SqlErrorLog -ServerInstance . -After (Get-Date).Date.AddDays(-7)  
-Credential $Credential | ft
```

範例中，以 SQL Server 自訂帳號登入，並透過 Get-SqlErrorLog cmdlet 讀取本機預設執行個體（以「.」代表）一星期內的紀錄。

SQL Server 2016 後新增「一律加密」功能，透過.NET4.6 版之後的函式庫在應用程式端加/解密資料。只需在資料表設定，ADO.NET 會自動先加密資料後再存入 SQL Server，讀取時自動解密，但無須額外撰寫程式，藉此簡化保護資料和遵循法規的設計與實作。

在此，簡單示範透過 SSMS 將未採用「一律加密」功能的資料表內欄位，透過一律加密精靈產生 SqlServer 模組支援的 PowerShell 語法來轉換這些欄位，讓其內的資料變成加密後的內容。

「一律加密」精靈針對目標資料表要轉換成啟用一律加密功能的資料表時，在最後「回合設定」步驟若要產生指令碼，便是產生 PowerShell 指令碼。首先透過 SSMS 物件總管選擇要啟用一律加密的資料行，如圖 D.16 所示：

⁵ 從圖 D.15 可以看到「一律加密(always encrypted)」在 SQLServer Powershell 提供的 cmdlet 命名採用的是 ColumnEncryption。



圖 D.16 透過一律加密精靈產生加密資料欄位的 PowerShell script

在「資料行選取」步驟選擇欲加密的資料行與加密類型後，在「回合設定」步驟可以產生 PowerShell 語法⁶，內容如下：

範例程式 D.3：透過 PowerShell 設定針對那些資料行啟用一律加密

```

Import-Module SqlServer
# Set up connection and database SMO objects

$sqlConnectionString = "Data Source=sql2022;Initial Catalog=Northwind;Integrated
Security=True;Connect Timeout=30;Encrypt=False;Packet Size=4096;Application
Name='Microsoft SQL Server Management Studio'"
$smoDatabase = Get-SqlDatabase -ConnectionString $sqlConnectionString
  
```

⁶

撰寫本節時擷取「一律加密」精靈所產生的 PowerShell 語法，發現與前版不同。應是微軟持續更新 SSMS 時有改寫此功能。或許，你看到的 PowerShell 語法也會與此處不同。

```
# Change encryption schema

$encryptionChanges = @()

# Add changes for table [dbo].[Customers]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName
dbo.Customers.ContactName -EncryptionType Deterministic -EncryptionKey "cek"
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName
dbo.Customers.Phone -EncryptionType Deterministic -EncryptionKey "cek"

Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges
-InputObject $smoDatabase
```

PowerShell Script 的 Set-SqlColumnEncryption cmdlet 約略完成了如下的動作：

- 依精靈所收集的欄位加密設定建立新的資料表。
- 將舊資料表的資料批次 Bulk 載入到新資料表，藉以透過 ADO.NET 4.6 以上版本加密資料內容。
- 刪除舊資料表。
- 重新命名新資料表成舊的名稱。
- 若舊資料表有條件約束、索引或設定權限，則為新的資料表建立同樣的定義。

❖ 整合 Analysis Services

除了 SQL Server 資料庫引擎外，SqlServer 模組也為 Analysis Services 服務提供了 cmdlet。簡單列出與 AS 相關的指令：

```
Get-Command *-as* -Module sqlserver
```

得到：Backup-ASDatabase、Invoke-ASCmd、Restore-ASDatabase。

可以透過如 Invoke-ASCmd cmdlet 對 Analysis Services 執行指令：

```
Invoke-ASCmd -Server 'localhost\tabular' -Query '{
  "refresh": {
    "type": "clearValues",
    "objects": [
      {
        "database": "TabularDemo"
      }
    ]
  }
}
```

上述指令會對本機 Analysis Services 的 tabular 模型執行個體傳送 TMSL (Tabular Model Scripting Language) 指令，而範例中是對 TabularDemo 資料庫進行「處理清除 (clearValues)」，也就是清掉所有該資料庫先前處理的內容。若上述「type」鍵值指定「automatic」，則是告知 Analysis Services 依其預設方式決定如何處理資料庫。

由於 Analysis Services 超出本書的範圍，在此就不再討論。

D.3 綜合應用

SQL Server 支援 PowerShell 有何用處呢？其實，除了如前文透過「SQL 提供者」提供 DIR、CD、REN、DEL 等指令，讓不熟 T-SQL 的管理者可以批次查詢 SQL Server 外，藉由 Invoke-SQLCmd cmdlet 可以執行各種 SQL 語法，整合 PowerShell 的 Script 功能後，在程式應用上將變得更有彈性。以一個簡單的範例說明如下：

```
gc 'server.txt' | % {Invoke-SQLCmd -Server $_ -Query "select @@servername as
[Server], count(*) as [Blocked]
from master.dbo.sysprocesses where blocked <> 0" }
ft -AutoSize
```

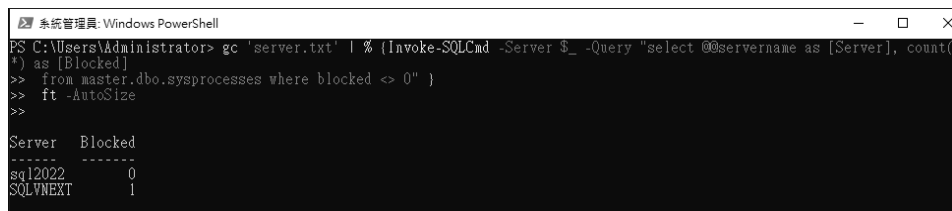

由於本章並不是在討論 PowerShell 的腳本語言寫作，因此僅描述上列範例的語法意義。透過 gc (Get-Content cmdlet 的別名) 一次讀取一行執行目錄下的 server.txt 檔案中之文字，藉由管線傳給後續的運算。而「%」代表 Foreach-Object cmdlet 的迴圈運算，它會將 gc 傳來的字串帶入大括號內「{ }」的 script，這些字串將出現在「\$_」的位置。因此，整句查詢是對所有 server.txt 檔案表列的 SQL Server 服務執行個體；查詢其內有多少被鎖定的狀況：

```
select @@servername as 'Server', count(*) as 'Blocked' from
master.dbo.sysprocesses where blocked <> 0
```

我們示範用的 Servers.txt 文字檔內容如下：

```
sql2022
sqlvnext
```

上述透過 PowerShell 統一對多個 SQL Server 執行個體查詢 Blocked 的狀況，其結果如圖 D.17 所示：



```
PS C:\Users\Administrator> gc 'server.txt' | % {Invoke-SQLCmd -Server $_ -Query "select @@servername as [Server], count(*) as [Blocked]
>> from master.dbo.sysprocesses where blocked <> 0" }
>> ft -AutoSize
>>

Server    Blocked
-----
sql2022   0
SQLVNEXT  1
```

圖 D.17 透過 PowerShell 搭配 T-SQL 查詢多個服務執行個體的鎖定/被鎖定狀況

可將上述的語法以一般文字檔案格式，存成 PowerShell 使用的批次指令檔案（副檔名為.ps1），例如在 C 磁碟 temp 目錄下建立 test.ps1，其內容如下：

範例程式 D.4：自行撰寫 PowerShell 指令檔案 test.ps1

```
# 載入 SQL Server Module
Import-Module "sqlps" -DisableNameChecking
#取得叫起 ps1 script 執行的檔案路徑
$path=split-path $MyInvocation.MyCommand.Path
gc "$path\Servers.txt" | `
% { Invoke-SQLCmd -Server $_ -Query "select @@servername as [Server], count(*) as
[Blocked]
from master.dbo.sysprocesses where blocked <> 0" } | `
ft -AutoSize
```

其中 Import-Module⁷ 指令是要求 PowerShell 載入 SQL Server 所提供的模組，這才能解釋 Invoke-SQLCmd cmdlet。

因為安全疑慮，PowerShell 要求待執行的批次指令檔案需要簽章。另一個簡單的方式是設定本機的 PowerShell 腳本檔不需要任何的安全限制，凡是遠端的 Script 須被信任的發證單位簽章過，才可以執行⁸。在命令提示列執行 PowerShell.exe 後，執行如下的語法：

```
Set-ExecutionPolicy RemoteSigned
```

PowerShell 的執行規範（Execution Policy）是避免在不知的情況下，執行到不良意圖的 Script。但不限制自己主動要執行的 Script，所

⁷ SQL Server 2008 會自動載入 snap-in，但在 SQL Server 2012 則需手動載入模組。且 SQL Server 2012 提供的 Cmdlet 中有部分指令命名未遵照 PowerShell 的要求，所以載入模組時會有警告：「警告：有些匯入的命令名稱包含未核准的動詞，因此可能不易搜尋。請使用 Verbose 參數取得詳細資訊，或輸入 Get-Verb 查看核准的動詞清單。」。若不想看到警語，可以設定-DisableNameChecking 參數。SQL Server 2016 後則沒有這些問題。

⁸ 可以設定 Set-ExecutionPolicy Unrestricted，不限制執行 PowerShell Script，但因為 PowerShell 功能非常強大，為了安全還是搭配 RemoteSigned 或是 AllSigned(一律簽章過的腳本檔才可以執行) 較佳。這可以透過群組原則對多台電腦統一設定：

電腦設定→原則→系統管理範本→Windows 元件→Windows PowerShell→開啟指令碼執行

以使用者可以透過 Set-ExecutionPolicy; 任意改變當下 PowerShell 執行的規範。

若是在 Windows Vista 之後的作業系統啟動 PowerShell 執行上述語法，需要透過「以系統管理員身分」啟動 PowerShell 執行環境，否則會有如下的錯誤：

```
Set-ExecutionPolicy : 拒絕存取登錄機碼  
'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell'。  
位於 行:1 字元:20  
+ Set-ExecutionPolicy <<<< Unrestricted
```

由於 PowerShell 在執行批次指令檔案時，一定要告知批次檔案所在路徑，以避免居心不良的人將腳本檔名命名為 dir、ILoveYou...等，誤導受害者點擊或輸入檔名便執行，而 PowerShell 又功能強大，讓惡意者有機可乘。所以若是直接以檔案總管雙擊 test.ps1，或在命令提示列直接敲檔名執行.ps1 腳本檔，或是執行類似如下的語法：

```
C:\powershell.exe -f test.ps1
```

將會得到如下的錯誤訊息：

```
無法辨識 'test.ps1' 詞彙是否為 Cmdlet、函數、指令檔或可執行程式的名稱。請檢查名稱拼字是否正確，如果包含路徑的話，請確認路徑是否正確，然後再試一次。  
位於 行:1 字元:9  
+ test.ps1 <<<<  
+ CategoryInfo          : ObjectNotFound: (test.ps1:String) [], CommandNot  
FoundException  
+ FullyQualifiedErrorId : CommandNotFoundException
```

若搭配檔案的相對路徑，以「.\」代表本目錄下的腳本檔：

```
C:\powershell.exe -f .\test.ps1
```

或是完整路徑皆可正確執行：

```
C:\powershell.exe C:\test.ps1
```

指令中的 -f 參數代表 -file 參數的簡寫，用以指定 script 檔案的路徑檔名。

D.3.1 在 SQL Server Agent 作業步驟使用 PowerShell

SQL Server Agent 作業步驟有好幾種類型。每一個類型都與特定環境的子系統有關，例如，複寫代理程式或命令提示字元環境。SQL Server 2008 版後，Agent 服務加上 PowerShell 子系統，其作業步驟可執行 PowerShell 指令碼。換句話說，你可以編寫 PowerShell 指令碼，然後透過 SQL Server Agent 服務排程執行包含這些指令碼的作業，或是回應 SQL Server 事件。SQL Server Agent PowerShell 子系統會載入及註冊 SQL Server PowerShell 嵌入式管理單元，以執行 SQL Server PowerShell 指令碼。

接著以範例說明，因為會使用到 SQL 的管理原則⁹，故請先建立一個停用 SQL Server 各項介面功能的原則。設定方式請參考圖 D.18：

⁹

SQL Server 2016 提供內建的「SQL Server 最佳作法」為 SQL Server Best Practices Analyzer(BPA)的相關原則，「SQL Server 最佳作法」可以協助你檢視、分析資料庫伺服器的整體環境，並提供適當的建議作為修正之用。預設的安裝目錄是 C:\Program Files\Microsoft SQL Server\130\Tools\Policies 內，你可以檢視並且匯入其所提供「SQL Server 最佳作法」之原則 (*.xml)。但 SQL Server 2022 版本已經移除，你若需要參考，可以自行到舊版所在的伺服器，於上述目錄內取用。



圖 D.18 建立 SQL Server 的管理原則

圖 D.18 在 Facet 選取「介面區組態」，並於下方的「運算式」選單視窗之「欄位」選擇除了 @WebAssistantEnabled 外；該 Facet 所有的項目，因為 SQL Server 2022 不再支援這個屬性。在「值」欄位全部設定「False」，亦即要求 SQL Server 停用該功能。回到「建立新原則」對話窗，於下方的「排程」窗格新增作業排程，請參考圖 D.19 所示：



圖 D.19 為檢測原則建立排程

完成這個新建立的原則前，確定有勾選上方「已啟用」格，其後才能定期檢測該原則。

接著便檢視這個排程為原則定義的 Sql Agent 作業，藉以說明 Invoke-PolicyEvaluation cmdlet。

檢視上一步精靈自動產生的「作業」，其「步驟」內容如圖 D.20 所示：



圖 D.20 透過 Agent Services 服務的 PowerShell 作業類型執行 PowerShell 指令碼

圖 D.20 下方的「命令」窗內容如下：

```
dir SQLSERVER:\SQLPolicy\sql2022\default\Policies | where { $_.ScheduleUid -eq
"82C7F352-7C3D-4E52-BE19-A575F403BE67" } | where { $_.Enabled -eq 1 } | where
{ $_.AutomatedPolicyEvaluationMode -eq 4 } | Invoke-PolicyEvaluation
-AdHocPolicyEvaluationMode 2 -TargetServerName sql2022
```

上述程式碼中，先用 dir(Get-ChildItem 的別名為) 找尋 SQLSERVER:\SQLPolicy\SQL2022\DEFAULT\Policies 目錄下的原則，其中 SQL2022 是伺服器名而 Default 是 SQL Server 執行個體名稱。在該 SQL Server 2022 執行個體中，已建立的原則，也就是 Microsoft.SqlServer.Management.Dmf.Policy 物件所成的集合。再藉由 Where-Object cmdlet

根據 PolicyCategory 屬性；篩選符合排程定義，且已經啟用，評估模式為「按排程時間」。

Where-Object cmdlet 會將物件逐一丟給以大括號「{}」括起來的指令碼區塊，而「\$」則代表每一個物件，因此「\$.ScheduleUid」就是取 dir cmdlet 傳來的某個 Microsoft.SqlServer.Management.Dmf.Policy 物件執行個體的 ScheduleUid 屬性，據此來當作過濾條件。

一系列的管線轉交物件後，最終到了 Invoke-PolicyEvaluation cmdlet。Invoke-PolicyEvaluation 會取用 Where-Object 輸出的物件，針對「SQL2022」伺服器的預設執行個體驗證各個原則，並將結果輸出到 console 與 msdb 系統資料庫。

在此簡單地執行 Invoke-PolicyEvaluation cmdlet 與輸出結果如圖 D.21：

```
PS SQLSERVER:\sqlpolicy\sql2022\default\policies> dir | Invoke-PolicyEvaluation -TargetServerName sql2022
```

ID	Policy Name	Result	Start Date	End Date	Messages
1	serviceArea	False	2022/7/30 上午 10:15	2022/7/30 上午 10:15	
1	test	True	2022/7/30 上午 10:15	2022/7/30 上午 10:15	

圖 D.21 簡單驗證所有的原則

若要將結果輸出到檔案，可以繼續在上述語法後接上 Out-File 一類的輸出。或是透過 SSMS 直接檢視驗證原則的結果，如圖 D.22：

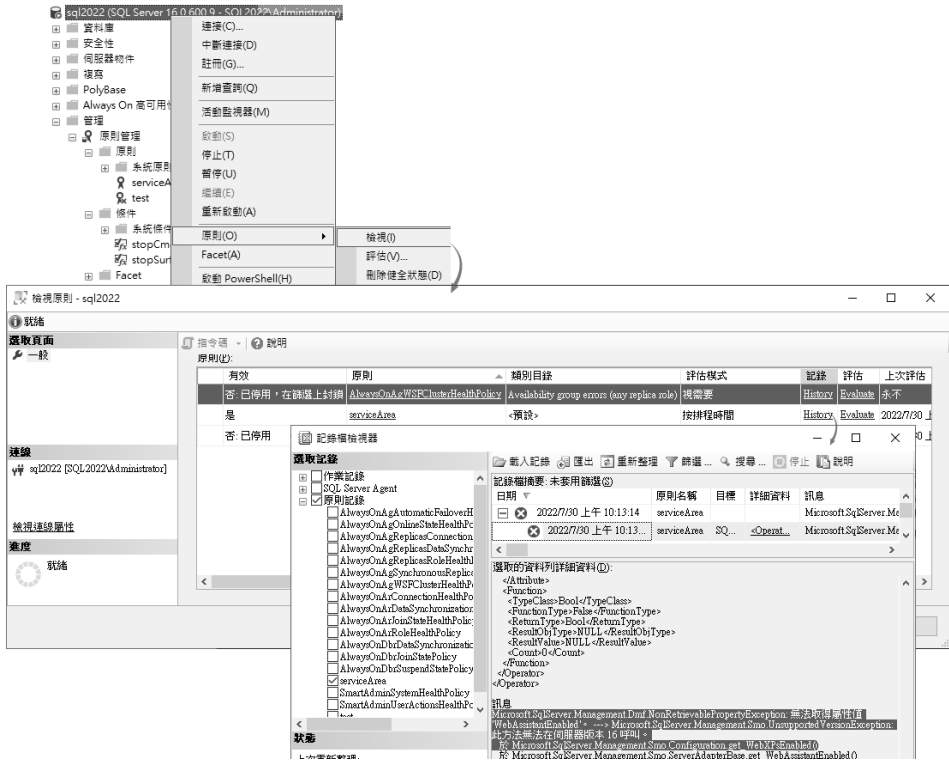


圖 D.22 檢視驗證原則後的結果

在我們撰寫本章時（2022/7），SQL Server 2022 的 Invoke-Policy Evaluation cmdlet 在驗證「介面區組態」Facet 選取會有「無法取得屬性值 'WebAssistantEnabled'」的 bug，但透過 SSMS 直接驗證「介面區組態」Facet 提供的條件卻不會有問題。希望當你閱讀到本章時，該 bug 已經解決。

D.3.2 設定環境

在前幾個小節當中，已經介紹了 SQL Server 2008 後所引進的 PowerShell 功能，在接下來的小節中，將繼續舉些例子，並對使用到的 PowerShell 語彙稍做解釋，讓你可以整合 PowerShell 的能力到日常管理應用中。

為了節省篇幅，也就不將這些指令碼執行後的畫面貼圖到文章中，你可以自行參照本書的範例程式檔，而後在 PowerShell 環境中執行即可看到結果。

首先撰寫指令檔 01SetupEnv.ps1，嘗試載入 SqlServer 或 SqlPs PowerShell 模組，再宣告與設定通用的變數，如電腦名稱、SQL Server 執行個體、自行撰寫 PowerShell 指令檔的存放目錄等，藉以提供其他指令檔呼叫，內容如下：

範例程式 D.5：宣告與設定通用的變數

```
# 設定發生錯誤後，要如何進行，可以設定：continue(預設)、silentlycontinue 以及 stop
$ErrorActionPreference = 'Stop'
#先嘗試載入 sqlserver module 再嘗試載入 sqlps，若都不存在，就丟出錯誤
Import-Module 'sqlserver' -ErrorAction SilentlyContinue;
if( !(Get-Module | where {$_.name -eq 'sqlserver'}) )
{
    Import-Module 'sqlps' -DisableNameChecking -ErrorAction SilentlyContinue
    -WarningAction SilentlyContinue ;

    if( !(Get-Module | where {$_.name -eq 'sqlps'}) )
    {
        #丟出例外，停止執行
        throw '未安裝 sqlps 或 sqlserver 模組'
    }
}

# 設定通用的變數
Set-Variable -scope Global -name machineName -Value "$env:ComputerName";
#透過 $env: 命名空間，取得系統環境變數 ComputerName 的值，也就是本機電腦名稱
# 預設的執行個體名稱可用 Default
Set-Variable -scope Global -name instanceName -Value "Default";
Set-Variable -scope Global -name scriptPath -Value
'C:\BookSamples\SQL2022Management\appendixD'
```

在 PowerShell 中，井字號「#」代表之後文字是說明，出現的位置不一定要在行首。而在範例程式內，我們盡量加註說明，讓你讀取指令碼時較容易理解。

PowerShell 有些參數稱為「一般參數(Common Parameters)」，由 PowerShell 引擎所控制，為各 cmdlet 所共有，不因為 cmdlet 不同而有不同的行為模式。這些「一般參數」包括 WhatIf、Confirm、Verbose、Debug、Warn、ErrorAction、ErrorVariable、OutVariable 和 OutBuffer 等。你可以在命令提示列直接輸入以下語法，以取得說明：

```
Get-help about_commonparameters
```

若是出現類似 "Get-Help 在此工作階段的說明檔中找不到 about_commonparameters" 的錯誤訊息，代表系統未建立說明。Windows 系統雖內建 PowerShell，但並非所有人都需要使用 PowerShell，進而閱讀相關說明，所以需要手動下 update-help，才會安裝說明檔。

或是線上搜尋 about_commonparameters。

而在上述範例程式 D.5 中，僅是經由 ErrorAction 告知 Import-Module cmdlet，若發生任何錯誤繼續執行而不管錯誤（SilentlyContinue）。接下來，簡單示範一些撰寫指令碼時常用的作法。例如，從三種常用的格式：一般文字檔案、XML 或 JSON 檔案中，讀出相關的設定：

範例程式 D.6：從設定檔讀出定義

```
Invoke-Expression (Join-Path (Split-Path $MyInvocation.MyCommand.Path)
'appendixD D.5：宣告與設定通用的變數.ps1')

# Text Driver
# databases.txt 檔案內容如下：
# Northwind
# AdventureWorks
Push-Location
# 呈現檔案內每一行的內容
get-Content ($scriptPath + "\databases.txt") | foreach {write-Host "操作資料庫 " $_ }
# 例如，使用 SQL Server 提供的 Provider，表列資料庫內的資料表
get-Content ($scriptPath + "\databases.txt") | foreach {
```

```
{CD sqlserver:\sql\%machineName%\%instanceName%\databases\%$ \tables; `
write-Host "資料庫內的資料表：" $_ ; DIR | Where {$_.Name -like 't*' } | MORE;}
Pop-Location

# 透過 XML Driver 讀取 XML 文檔內容
$doc = [xml]( get-Content ($scriptPath + "\Objects.xml") )
# 透過 XML DOM 取得節點內容：
$doc.SelectNodes('//servers/server[@servername="SQL2022"]/databases').InnerXML
# 透過 XML DOM 取得節點內容：
$doc.SelectNodes('//servers/server[@servername="SQL2022"]/databases/database[@
databasename="dbDemo"]/table')
# 遞迴表列節點...
foreach ($database in $doc.SelectNodes("//servers/server/databases/database"))
{$database.databasename}

#透過 ConvertFrom-Json 讀取 Json 內容，再返序列化回物件
$json=(gc ($scriptPath + "\Objects.json")) | ConvertFrom-Json
#以物件集合屬性來讀取 Json
$json.Servers[0].Databases[1]
```

上述範例中所參照的 Objects.xml 檔案內容如下。由於自訂的 \$scriptPath 公用變數是指到 C:\BookSamples\SQL2022Management\appendixD 目錄，所以這些存放設定的範例檔案需全部放到該目錄下：

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <server servername = "SQL2022">
    <databases>
      <database databasename = "dbDemo">
        <table tablename = "tbTest"/>
      </database>
      <database databasename = "Northwind">
        <table tablename = "Customers"/>
      </database>
    </databases>
  </server>
</servers>
```

範例程式 D.6 範例中：

```
$doc = [xml]( get-Content ($scriptPath + "\Objects.xml") )
```

透過 [xml] 轉型後，\$doc 變數承接的是 .NET System.Xml.XmlDocument 物件，可以透過 SelectNodes 搭配 XPath 查詢節點內容。

除了以 xml 格式外，透過 JSON(JavaScript Object Notation) 格式存放定義也行。範例中透過 ConvertFrom-Json cmdlet 讀取 Objects.json 文字檔，再以物件集合的方式取得某項定義值，該檔案內容如下：

```
{
  "Servers": [
    {
      "ServerName": "SQL2022",
      "Databases": [
        {
          "DatabaseName": "dbDemo",
          "Tables": [
            { "TableName": "tbTest" }
          ]
        },
        {
          "DatabaseName": "Northwind",
          "Tables": [
            { "TableName": "dbo.Customers" }
          ]
        }
      ]
    }
  ]
}
```

上述範例程式 D.6 中，透過 ConvertFrom-Json cmdlet 將 Json 字串內容轉回物件，而後透過讀取物件屬性的方式便可以取得設定值。

在此要提醒的是，PowerShell 的逸出字元是反括號「`」¹⁰，並非一般 C 系列語言的反斜線「\」，因為反斜線在殼層環境易與路徑標示混淆。若要使字元成為特殊控制碼，可在字元前面加上「`」。若要指定反括號「`」本身，就要使用連續兩個「`」（也就是``）。而 PowerShell 預設搭配逸出字元的控制碼如表 D.3 所示：

表 D.3 PowerShell 預設搭配逸出字元的控制碼

逸出字元	功能
``	單一個反括號「`」
`0	Null
`a	警示
`b	退格鍵
`f	換頁字元
`n	換行
`r	換行符號
`t	Tab 鍵
`v	垂直引號
`換行	空白或換行

說明表 D.3 中的「`換行」逸出字元。若是一般的運算式，「`換行」會被當成空白，因此它可比作撰寫 VB 等程式語言（C 系列的程式語言一般不需要換行符號）時的換行與連接符號。

¹⁰ 該符號的按鍵一般在鍵盤的「Tab」鍵上方。

```
PS C:\PowerShellScript> 'Hello ' + `
>> 'World'
>>
```

其執行結果如下：

```
Hello World
```

上例中，「`」加上換行的運算結果等同空白，所以在 Hello 和 World 兩個字中間多了一個空白。但若是字串內的運算，則「`換行」依然是換行。

```
PS C:\PowerShellScript> "Hello
>> `
>> World"
>>
```

其執行結果如下：

```
Hello

World
```

而此處又引發了另一個有趣的議題，用單引號所括起來的內容僅當作純文字，而雙引號括起的内容還會進行運算。所以「\$a=1;"\$a"」會傳回「1」，而「\$a=1;'\$a'」則是傳回「\$a」。

在範例程式 D.6 中，透過[xml]宣告，將 get-Content 從文字檔案讀入的內容直接轉型 (Cast) 成 XML DOM 物件使用。在此列出 PowerShell 預設提供，可用做轉型的常用關鍵字，如表 D.4 所示：

表 D.4 PowerShell 所提供可轉型的物件類型

宣告關鍵字	型別意義
int	32-bit signed 整數
long	64-bit signed 整數

宣告關鍵字	型別意義
string	Fixed-length string of Unicode 字串
char	Unicode 16-bit 字元
byte	8-bit unsigned 字元
bool	Boolean True/False value
decimal	128-bit decimal value
single	Single-precision 32-bit 浮點數
double	Double-precision 64-bit 浮點數
xml	Xml 物件
array	值所形成的陣列
hashtable	Hashtable 物件
...	... (還有許多，此處僅是舉出常用的宣告)

若直接在表 D.4 中，定義型別的字串後加上中括號，就是宣告該類型的陣列，例如 `int[]`，代表宣告整數陣列。因此，直接在 PowerShell 提示命令輸入如下指令：

```
PS C:\> [int[]][char[]]"Hello"
```

得到的結果如下：

```
72
101
108
108
111
```

因為將字串轉成字元所組成的陣列，而後再把每個陣列元素解釋成整數。

D.3.3 定義函數

PowerShell 允許建立自訂函數，以直接在互動介面或指令碼中使用¹¹，呼叫時如同一般指令。定義函數最簡單的方式就是以 Function 關鍵字宣告函數名稱，並以大括號包裹著函數內容。若要賦予函數的參數，其方式有三：

- 藉由 \$args 取得動態數量的參數。
- 經由 param 取得參數。
- 宣告函數時，一併宣告參數。

宣告函數的範例如下：

範例程式 D.7：宣告自訂函數

```
# 函數搭配參數
Function objectPath {"路徑為" + $args[0] + "\" + $args[1] }
# 呼叫函數，因為如同一般命令，所以參數間不以逗號分隔
objectPath a b

Function objectPath2 ($x, $y)
{
    $ObjectPath = $x + "\" + $y
    write-Host "路徑為 $ObjectPath"
}
objectPath a b

# 函數使用 "param"
# param 的宣告必須在第一行，在它之前只能有空白或註解
Function objectPath3
{
    param ($x, $y)
    $ObjectPath = $x + "\" + $y
    write-Host "路徑為 $ObjectPath"
}
objectPath3 a b
```

¹¹ PowerShell 提供四種可執行的命令：cmdlet、外部執行檔、函數和 ScriptBlock。在此簡單地利用函數來提供相同功能；可被重複執行叫用的能力。

與一般程式撰寫習慣不同的是，Powershell 有命令模式和腳本語言模式，在命令模式呼叫函數時，多個參數間不以逗號分隔「,」，而是以空白字元。若使用逗號分隔，是代表陣列元素。例如，以下範例定義函數內容是取「\$x」變數的前兩個陣列元素相加：

```
Function objectPath4($x){$x[0] + "\" + $x[1]}
```

則呼叫該函數時，可以用逗號分隔參數：

```
objectpath4 a,b
```

另外，需要提醒的是：當別名、函數或變數加入到 PowerShell 時，只是將它們加入到目前的 PowerShell 工作階段。若是結束該工作階段或是關閉 PowerShell，這些變更就會遺失。若要保留這些變更，可以建立 PowerShell 設定檔，然後將那些別名、函數和變數加入到設定檔中。此設定檔會在每次啟動 PowerShell 時載入。而為了載入設定檔，PowerShell 執行原則必須允許載入。如果未能載入或載入設定檔失敗，則會顯示錯誤訊息。而關於設定檔的解釋與使用，可以參照如下的網址：

```
https://learn.microsoft.com/zh-tw/previous-versions/technet-magazine/cc895642\(v=msdn.10\)
```

除了修改 PowerShell 設定檔外，若僅是在本次環境中執行過指令檔一次後，就希望重複叫用該指令檔的函數，可以在呼叫前加一個點「.」。以上述的指令碼檔為例，在 PowerShell 的提示符號後，先寫入點，空一格後，再寫完整的指令碼檔案路徑，範例如下：

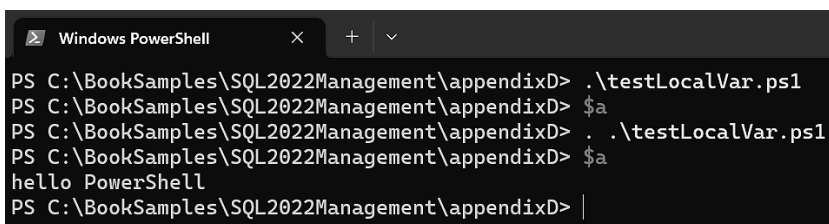
先建立一個測試的 Script 檔案 testLocalVar.ps1，內容如下

```
$a='hello PowerShell'
```

而後在 PowerShell 命令提示列執行：

```
PS C:\BookSamples\SQL2022Management\appendixD> . .\testLocalVar.ps1
```

一旦執行前加點（dotting），原本的區域函數或變數就變成全域函數或變數，之後也就可以重複叫用。如圖 D.23 所示：



```
Windows PowerShell
PS C:\BookSamples\SQL2022Management\appendixD> .\testLocalVar.ps1
PS C:\BookSamples\SQL2022Management\appendixD> $a
PS C:\BookSamples\SQL2022Management\appendixD> . .\testLocalVar.ps1
PS C:\BookSamples\SQL2022Management\appendixD> $a
hello PowerShell
PS C:\BookSamples\SQL2022Management\appendixD> |
```

圖 D.23 透過「.」將區域變數轉成全域變數

圖 D.23 中，第一次叫用 testLocalVar.ps1 腳本檔案時，未在前方加入「.」，所以該檔案內定義的\$a 變數在腳本執行完畢後就釋放了，因此在當下的環境中再次叫用\$a 變數（圖 D.23 的第二行），是未定義的任意變數。

但第二叫用 testLocalVar.ps1 腳本檔案時，前方加入了「.」，所以\$a 變數變成全域變數，所以雖然腳本檔執行完畢，但可以在環境中再次使用\$a 變數（圖 D.23 的第四行）。

接下來，簡單示範將訊息記錄到 Windows 事件紀錄的函數：

範例程式 D.8：透過 System.Diagnostics.EventLog 物件執行個體將訊息寫到 Windows 事件紀錄

```
#03WirteEventLog.ps1
# 回報紀錄機制
# 事件紀錄(Event Log)
Function log_This
{
    $log = New-Object System.Diagnostics.EventLog('Application')
```

```

$log.set_source($log_ApplicationName)
$log.WriteEntry($log_Message)
}
# 範例 - 呼叫函數傳遞內容時，不用參數只用變數
$log_ApplicationName = "自訂的應用程式名稱"
$log_Message = "輸入需要記錄的資訊"
# 呼叫函數，你可以用系統的事件檢視器檢視結果
log_This

# 讀取事件紀錄：
get-Eventlog application | where-Object {$_.Message -eq "輸入需要記錄的資訊"}
get-Eventlog application | where-Object {$_.source -eq "自訂的應用程式名稱"}

```

在 PowerShell 指令碼中，若要建立 .Net 或 COM 物件的執行個體，可以呼叫 `New-Object CmdLet`，賦予 .Net 的類別名稱或 COM 物件的程式識別元 (ProgID) 即可。而範例程式 D.8 就是建立 `System.Diagnostics.EventLog` 物件執行個體後，利用 `Set_Source` 和 `WriteEntry` 方法來寫入資料，而賦予資料的方式僅是在指令碼內參照同名變數，並未定義函數的參數。

再做一個定義參數的簡單示範，經由 `New-Object` 建立 .NET 所提供的 `StreamWriter` 類別，將資料寫入檔案：

範例程式 D.9：利用 .NET `StreamWriter` 類別，將資料寫入檔案

```

Function WriteFile($Msg)
{
    $sw = New-Object System.IO.StreamWriter
    'C:\BookSamples\SQL2022Management\appendixD\myLog.log', True
    $sw.WriteLine($Msg)
    $sw.Close()
}

#測試呼叫函數
WriteFile('Hello Msg')

#呈現記錄檔內容
gc C:\BookSamples\SQL2022Management\appendixD\myLog.log

```

當然，若僅要寫入檔案，比較方便的作法是透過管線與 Out-File Cmdlet（寫法為「| Out-File」）將資料直接輸入。

接下來，整理出一些有趣且有用的自訂函數，讓你日後在撰寫 PowerShell 指令檔時，可以呼叫這些函數，以記錄事件、寄送郵件或產生網頁等：

範例程式 D.10：將 PowerShell 的執行結果以各種方式輸出

```
# 紀錄錯誤
Function log_This(
[string]$logApp,
[string]$logMsg)
{
    $log = New-Object System.Diagnostics.EventLog('Application')
    $log.set_source($logApp)
    Write-Host $logMsg
    $log.WriteEntry($logMsg)
}

# 透過$error 公共變數呈現錯誤
Function handle_This (
[string]$appName)
{
    # 建立錯誤訊息字串
    $log_Message = "Error Category: " + $error[0].CategoryInfo.Category
    $log_Message = $log_Message + ". 執行的物件：" + $error[0].TargetObject
    $log_Message = $log_Message + " 錯誤訊息：" + $error[0].Exception.Message
    $log_Message = $log_Message + " 錯誤訊息：" + $error[0].FullyQualifiedErrorId
    Write-Host $log_Message
    write-Host "錯誤發生於 " $appName "檢查事件紀錄，以取得更多資訊"

    # 自動傳送訊息給先前撰寫的 Logging 函數：
    log_This $appName $log_Message
}

# 錯誤處理範例，執行某項工作
Function ErrorHappensHere
{
    SomeFun #沒有這個指令或函數，所以發生錯誤
    Trap {
```

```

        # 透過 Trap 集中錯誤處理，若要結構化錯誤處理，可以採用 try catch
        handle_This("SomeFun")
        # 若發生錯誤後，就停下來，可以使用 break
        # break;
        # 或是繼續進行
        continue;
    }
}

# 呼叫上述會發生錯誤的函數
ErrorHappensHere

write-Host "檢查事件紀錄"
# 呈現上述測試 PowerShell 寫入的 Windows 事件紀錄
get-Eventlog application | where-Object {$_.Source -like "SomeFun*"}
write-Host "完成錯誤輸出"

# 透過電子郵件寄發訊息
Function mail_This ($mail_From, $mail_To, $mail_Subject, $mail_Body,
    $mail_Attachment)
{
    # 簡易的電子郵件設定
    # $smtp = new-object Net.Mail.SmtpClient("localhost")
    # $smtp.Send($mail_From, $mail_To, $mail_Subject, $mail_Body)

    $smtpServer = "localhost"
    $msg = new-object Net.Mail.MailMessage
    $att = new-object Net.Mail.Attachment([string]$mail_Attachment)
    $smtp = new-object Net.Mail.SmtpClient $smtpServer
    $msg.From = $mail_From
    $msg.To.Add($mail_To)
    $msg.Subject = $mail_Subject
    $msg.Body = $mail_Body
    $msg.Attachments.Add($att)
    $smtp.Send($msg)
}

mail_This "PowerShell@sql2022" "Someone@sql2022" "來自 PowerShell 的通知" "測試從
PowerShell 發送電子郵件" 'C:\BookSamples\SQL2022Management\appendixD\myLog.log'
#若僅是測試 SMTP 功能，可在 Windows 啟用 SMTP 伺服器功能，並授與本機可透過 SMTP 虛擬伺服器
轉送郵 1 件之權限。在此是寄到範例機器 SQL2022 相同網域，預設可在 C:\inetpub\mailroot\Drop
目錄找到寄發的電子郵件檔。

# System Tray "Balloons"，在右下角工具提示列呈現通知訊息
Function balloon_This ($objNotifyIcon, $balloon_Title, $balloon_Text)

```

```

{
    $objNotifyIcon.Icon = 'C:\BookSamples\SQL2022Management\appendixD\Info.ICO'
    $objNotifyIcon.BalloonTipIcon = 'Info'
    # Info, Warning, Error
    $objNotifyIcon.BalloonTipTitle = $balloon_Title
    $objNotifyIcon.BalloonTipText = $balloon_Text
    $objNotifyIcon.Visible = $True
    $objNotifyIcon.ShowBalloonTip(10000)
}

[void]
[System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")
$objNotifyIcon = New-Object System.Windows.Forms.NotifyIcon
#若要讓 NotifyIcon 物件的 click 事件能觸發執行，則物件不能隨函數結束就消失
#所以建立物件需要在外面的 Script 區塊
#以下兩者都是 PowerShell 註冊物件的事件處理函數的方式
#大括號{}內即為要執行的邏輯腳本，在此僅是簡單地關閉離開
$objNotifyIcon.add_Click( {$objNotifyIcon.Dispose()} )
#Register-ObjectEvent -InputObject $objNotifyIcon -EventName Click -Action
{$objNotifyIcon.Dispose()}

    balloon_This $objNotifyIcon '注意!!' '要開始做些事情。事情進行時，可以透過此傳送些訊息'

# 轉成 HTML 檔案輸出
Function web_This ($web_Title)
{
    # 以 CSS styles 當作 HTML Header
    $formatHTML = "<style>"
    $formatHTML = $formatHTML + "BODY{background-color:white;color:black}"
    $formatHTML = $formatHTML + "TABLE{border-width: 1px;border-style:
solid;border-color: blue;border-collapse: collapse;}"
    $formatHTML = $formatHTML + "TH{border-width: 1px;padding:
0px;border-style: solid;border-color: black;background-color:white}"
    $formatHTML = $formatHTML + "TD{border-width: 1px;padding:
0px;border-style: solid;border-color: black;background-color:white}"
    $formatHTML = $formatHTML + "</style>"
    # 將系統當下執行的 Process 狀態以 HTML 格式輸出
    Get-Process | ConvertTo-Html -head $formatHTML -title $web_Title | Out-File
"C:\BookSamples\SQL2022Management\appendixD\$web_Title.htm"
    #產生 HTML 後，再叫起來這份網頁
    ."C:\BookSamples\SQL2022Management\appendixD\$web_Title.htm"
}
  
```

```
web_This "ProcessDetail"
```

上述範例程式透過各種方式記錄訊息，可以擷取所需的部分結合到你的指令碼中😊。

D.3.4 整合 SMO 物件與存取 SQL Server

接下來，回到 SQL Server 所提供的 PowerShell 模組（Module），透過其內的 cmdlet 或提供者（Provider）支援的命令提示指令，以存取與操作 SQL Server 執行個體內的物件：

範例程式 D.11：整合 SQL Server 提供 PowerShell 的 Module 執行結果以各種方式輸出

```
. ('C:\BookSamples\SQL2022Management\appendixD\D.5：宣告與設定通用的變數.ps1')

# 你可以結合前一段範例，將下述的結果轉成網頁，或是透過電子郵件寄送...
write-Host "
=====
使用 SQL Server 的 Provider，執行 T-SQL 查詢語法"
CD sqlserver:\sql\$machineName\$instanceName\databases\master\
invoke-SQLCMD -Query "SELECT @@VERSION"

write-Host "
=====
呈現使用者定義大過某個容量(Kbyte)的資料表，呈現其空間的使用"
CD sqlserver:\sql\$machineName\$instanceName\databases\Northwind\tables
DIR | where-Object {($_.DataSpaceUsed + $_.IndexSpaceUsed) -gt 200} | sort-Object
-property DataSpaceUsed | select-Object Name, RowCount, DataSpaceUsed,
IndexSpaceUsed

write-Host "
=====
檢查未備份的時間超過一天以上"
CD SQLSERVER:\SQL\$machineName\$instanceName\Databases
DIR | WHERE {(get-date)-($_.LastBackupDate)).days -gt 1} | sort-Object -property
LastBackupDate | select-Object LastBackupDate, Name
```



```

write-Host "
=====
備份資料庫 Northwind"
if ((Test-Path -Path 'C:\temp\dbBackups') -eq $false) {new-item -type directory
-name dbBackups -path 'C:\temp'}
CD sqlserver:\sql\$machineName\$instanceName\databases\
DIR | % {[string]$_ .Name + ' 最後備份日期:' + $_.LastBackupDate}
DIR | where{ (((get-date)-($_.LastBackupdate)).days -gt 1) -and $_.name -eq
"Northwind"} | `
%{$dbname= $_.Name;write-host "$dbname"; $_.Refresh();
invoke-sqlcmd -Server "$machineName" -query "BACKUP DATABASE [$dbname] TO DISK
= N'C:\temp\dbBackups\$dbname.bak' WITH INIT,COMPRESSION";}

write-Host "
=====
檢查 SQL Error Logs · 轉成 HTML"
# web_This "SQLErrors"
(get-item SQLSERVER:\SQL\$machineName\$instanceName).ReadErrorLog() | `
where {$_.Text -like "Start*"} | ConvertTo-Html -property LogDate, ProcessInfo,
Text `
-body "執行個體: $machineName\$instanceName" -title "SQL Server Error Logs" | `
Out-File C:\temp\SQLErrors.htm -Append
#叫起該網頁?
.'C:\temp\SQLErrors.htm'

write-Host "
=====
比較資料庫物件的 Scripts"
CD SQLSERVER:\SQL\$machineName\$instanceName\Databases\Northwind\Tables
# 將資料表的定義寫入檔案中, 指定物件時要帶著 schema
(get-item dbo.Employees).Script() | out-File C:\temp\before.sql
# Time passes, table changes
# 可能會需要更新一下物件定義
# (get-item HumanResources.Employee).Refresh()
invoke-SQLCMD -InputFile
C:\BookSamples\SQL2022Management\appendixD\DropCreateEmployee2.sql
# 建立資料表的 T-SQL Script 到檔案中?
(get-item dbo.Employees2).Script() | out-File C:\temp\after.sql
# 比較兩個檔案內容的異同
compare-object (get-content C:\temp\before.sql) (get-content C:\temp\after.sql)
-IncludeEqual | out-File C:\temp\CompareResult.txt -Width 200

.'C:\temp\CompareResult.txt'

```

範例程式 D.11 最後會透過 T-SQL 指令檔建立 dbo.Employees2 資料表，其檔案位置在 C:\BookSamples\SQL2022Management\appendixD\DropCreateEmployee2.sql，檔案內容如下：

範例程式 D.12：以 T-SQL 指令碼檔案供 invoke-SQLCMD cmdlet 執行

```
drop table if exists dbo.Employees2;
CREATE TABLE dbo.Employees2(
    EmployeeID int IDENTITY(1,1) NOT NULL,
    LastName nvarchar(20) NOT NULL,
    MiddleName nvarchar(10) NOT NULL,
    FirstName nvarchar(10) NOT NULL,
    Title nvarchar(30) ,
    TitleOfCourtesy nvarchar(25) ,
    BirthDate datetime NULL,
    HireDate datetime NULL,
    Address nvarchar(60) ,
    City nvarchar(15) ,
    Region nvarchar(15) ,
    PostalCode nvarchar(10) ,
    Country nvarchar(15) ,
    HomePhone nvarchar(24) ,
    Extension nvarchar(4) ,
    Photo varbinary(max) NULL,
    Notes nvarchar(max) ,
    ReportsTo int NULL,
    PhotoPath nvarchar(255)
)
```

上述範例程式 D.11 中，透過 invoke-SQLCMD -InputFile 指令執行範例程式 D.12 的 T-SQL 語法，而後再輸出 dbo.Employees2 資料表定義到 C:\temp\after.sql 檔案中，最後以 compare-object cmdlet 比較先前產生的 dbo.Employees 資料表指令碼定義 C:\temp\before.sql，並將結果存在 C:\temp\CompareResult.txt 檔案中。在此是新建一個資料表，藉以比較兩個資料表的定義差別，你也可以利用本章最後的範例，將歷來的定義存檔，而後再進行比較。

除了利用 SQL Server 提供 PowerShell 的模組，執行 CD、DIR、invoke-sqlcmd 等指令外，可以再透過 new-object cmdlet 建立物件，例如 ADO.NET、SMO、WMI 等物件的執行個體，來存取 SQL Server 資料庫內資料或操控服務。

從 SQL Server 2005 版後，微軟提供管理 SQL Server 的物件 SQL Server Management Objects (SMO)，讓應用程式 SQL Server Management Studio (SSMS) 呼叫，也可以透過 .NET 程式語言如 C#/VB.NET 呼叫，另外，就是透過 PowerShell 呼叫。

SMO 可以大分成兩類的物件，一是對應 SQL Server 執行個體的組織階層，所以從執行個體一路展開物件階層，透過各種物件集合擷取其內的物件。如範例程式 D.13 透過資料庫集合存取 Northwind 資料庫內的資料表集合，再依據資料表集合內每個資料表物件的 rowcount 屬性排序。最後擷取前十名的資料表，輸出 schema, name, rowcount 三個屬性到檔案中，範例指令碼如下：

範例程式 D.13：經由 SMO 物件，存取 SQL Server 物件

```
. ('C:\BookSamples\SQL2022Management\appendixD\D.5：宣告與設定通用的變數.ps1')

$SqlServer = new-object ("Microsoft.SqlServer.Management.Smo.Server")
"$machineName"
# 表列所有的資料庫
foreach($sqlDatabase in $SqlServer.databases) {$sqlDatabase.name}
# 檢視可以操作的物件屬性方法，量太大，輸出成 html
$SqlServer | get-member | ConvertTo-Html -property Name, MemberType, Definition
-body "Microsoft.SqlServer.Management.Smo.Server 類別" -title "Smo.Server 類別的
屬性方法" | Out-File C:\temp\SmoServer.htm
# 將前十名的資料表名稱輸出到檔案
$SqlServer.Databases["Northwind"].Tables | sort -property rowcount -desc |
Select-Object -First 10 | Format-Table schema, name, rowcount -AutoSize | out-File
C:\temp\TopTenTables.txt

.'C:\temp\TopTenTables.txt'
```

SMO 另一類物件稱為工具類別，有 Transfer、Backup 和 Restore 以及 Scripter 等類別。這些類別不必透過樹狀結構階層組織起來，可以直接建立執行個體以設定屬性、呼叫方法。接下來，透過 SMO 物件來備份資料庫。雖然在前述的範例程式 D.11 也有透過 T-SQL 語法完成備份，但此處示範透過 SMO 物件的 Backup 子物件之屬性、方法，備份所有的使用者資料庫，範例如下：

範例程式 D.14：透過指令碼呼叫 SQL Server SMO 物件，備份執行個體內的資料庫

```
# 對指定的伺服器執行個體內，所有使用者資料庫先做完整備份，
# 再進行交易紀錄備份
Function Backup($ServerInstance)
{

[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SMO") |
out-null

    $s = new-object ('Microsoft.SqlServer.Management.Smo.Server') "$ServerInstance"
    #備份到 SQL Server 執行個體預設的目錄
    #C:\Program Files\Microsoft SQL Server\<執行個體版本>\MSSQLSERVER\MSSQL\Backup
    $bkdir = $s.Settings.BackupDirectory
    $dbs = $s.Databases
    $dbs | foreach-object {
        $db = $_
        if ($db.IsSystemObject -eq $False) {
            $dbname = $db.Name
            Write-Host "備份資料庫 $dbname"
            $dt = get-date -format yyyyMMddHHmmss
            $dbbk = new-object Microsoft.SqlServer.Management.Smo.Backup

            $dbbk.Action = "Database"
            $dbbk.BackupSetDescription = $dbname + " 的完整備份"
            $dbbk.BackupSetName = $dbname + " Backup"
            $dbbk.Database = $dbname
            $dbbk.MediaDescription = "Disk"
            $FileName=$bkdir + "\" + $dbname + "_db_" + $dt + ".bak"
            $dbbk.Devices.AddDevice($FileName, "File")
            #壓縮並初始化

            $dbbk.CompressionOption=[Microsoft.SqlServer.Management.Smo.BackupCompressionOptions]::On
            $dbbk.Initialize=$true
```

```
#產生 Backup 的 T-SQL 語法
$dbbk.Script($s) | Out-File 'C:\temp\backup.sql' -Append
#真正執行備份
$dbbk.SqlBackup($s)

# 若 Recovery Model 是 Simple，則 Value 屬性值是 3
if ($db.recoverymodel.value__ -ne 3) {
    $dt = get-date -format yyyyMMddHHmmss
    $dbtrn = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
    $dbtrn.Action = "Log"
    $dbtrn.BackupSetDescription = $dbname + " 的交易紀錄備份"
    $dbtrn.BackupSetName = $dbname + " Backup"
    $dbtrn.Database = $dbname
    $dbtrn.MediaDescription = "Disk"
    $dbtrn.Devices.AddDevice($bkdir + "\" + $dbname + "_tlog_" + $dt +
".trn", "File")
    $dbtrn.SqlBackup($s)
}
}

# 以互動的方式詢問需要備份的 SQL Server 執行個體
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic') |
out-null
$instance=[Microsoft.VisualBasic.Interaction]::InputBox("請輸入需要備份的 SQL
Server 執行個體", "執行個體名稱", "localhost")
Backup $instance
```

範例中，透過設定 `Microsoft.SqlServer.Management.Smo.Backup` 物件執行個體的各项屬性，並呼叫 `SqlBackup` 方法執行備份。但也可以僅呼叫 `Script` 方法，產生 T-SQL 的 Backup 語法，不實際執行備份。

範例程式 D.14 如下的語法就是產出對每一個資料庫的 Backup 語法後，附加到 `backup.sql` 檔案中：

```
$dbbk.Script($s) | Out-File 'C:\temp\backup.sql' -Append
```

最後，示範透過 SMO 物件提供的 `Scripter` 物件，為指定的 SQL Server 執行個體建立 T-SQL Script。

範例程式 D.15：透過 SQL Server SMO Scripter 工具物件，產生指定 SQL 物件集合的 T-SQL DDL 定義語法

```
[CmdletBinding()]
param (
    $sqlInstance='localhost'
)
#bcp "exec etlConfig.migrate.spGetRecord @tableName='etl.tbItem',
@startDate='20220801'" queryout "C:\temp\etl.tbItem.csv" -T -w -t0x06 -r0x08
#bcp etlConfig.migrate.tbTmp_etl_tbItem in "C:\temp\etl.tbItem.csv" -T -w -t0x06
-r0x05

#在 PowerShell Script 內定義要利用 T-SQL 產生物件定義的集合，再交由 SMO 的 Scripter 產生
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SMO") |
out-null
#建立連接到某個 SQL Server 執行個體的 SMO.Server 物件
$SMOserver = New-Object ('Microsoft.SqlServer.Management.Smo.Server')
-argumentlist $sqlInstance

$scriptr = new-object ('Microsoft.SqlServer.Management.Smo.Scripter')
($SMOserver)

#存放定義的目錄結構 C:\temp\database\<Instance>\<date>\<InstanceObjectType>
#或 C:\temp\database\<Instance>\<date>\<DatabaseName>\<DatabaseObjectType>
$DateFolder = get-date -format yyyyMMddHHmm
# -replace 用得是 regular expression
$sqlInstance = $sqlInstance -replace "\/|\\|\.:", "_"
$SavePath = "C:\TEMP\Databases\" + $sqlInstance

new-item -type directory -name "$DateFolder" -path "$SavePath"

#Instance 等級要建立的物件集合
#定義要產生 Script 的集合，再逐一交給 Scripter 產生 T-SQL 語法
$ServerObjects = $SMOserver.Logins
$ServerObjects += $SMOserver.Databases
foreach($ScriptThis in $ServerObjects | where {!(($_.IsSystemObject))})
{
    #若存放某種物件型態的目錄不存在，先建立該型態名稱的目錄
    $TypeFolder=$ScriptThis.GetType().Name
    if ((Test-Path -Path "$SavePath\$DateFolder\$TypeFolder") -eq $false)
    {new-item -type directory -name "$TypeFolder"-path "$SavePath\$DateFolder"}

    "產出 $TypeFolder $ScriptThis" | Out-Default
    $ScriptFile = $ScriptThis -replace "\[|\]|\\\""
```

```

$scriptr.Options.FileName = "$SavePath\$DateFolder\$TypeFolder\
$ScriptFile.sql"
[void]$scriptr.Script($ScriptThis)
}

#定義 Scriptr 產生資料庫內物件的相關選項
$scriptr.Options.AppendToFile = $True
$scriptr.Options.AllowSystemObjects = $false
$scriptr.Options.ClusteredIndexes = $True
$scriptr.Options.DriAll = $True
$scriptr.Options.ScriptDrops = $False
$scriptr.Options.IncludeHeaders = $True
$scriptr.Options.ToFileOnly = $True
$scriptr.Options.Indexes = $True
$scriptr.Options.Permissions = $True
$scriptr.Options.WithDependencies = $False
foreach($db in $SMOserver.databases)
{
    #不建立系統資料庫物件
    if(-not $db.IsSystemObject)
    {
        #DB 等級要建立的物件
        $Objects = $db.Tables
        $Objects += $db.Views
        $Objects += $db.StoredProcedures
        $Objects += $db.UserDefinedFunctions
        $Objects += $db.Users

        #建立目錄結構
        new-item -type directory -name $db.Name -path "C:\TEMP\Databases\
$sqlInstance\$DateFolder"

        $SavePath = "C:\TEMP\Databases\$sqlInstance\$DateFolder\" + $($db.Name)

        foreach ($ScriptThis in $Objects | where {!(($_.IsSystemObject))}) {
            #為不同類型物件建立不同的目錄名稱
            $TypeFolder=$ScriptThis.GetType().Name

            if ((Test-Path -Path "$SavePath\$TypeFolder") -eq $false)
            {new-item -type directory -name "$TypeFolder"-path "$SavePath"}

            "產出 $TypeFolder $ScriptThis" | Out-Default
            $ScriptFile = $ScriptThis -replace "\\[|\\]|\\\"
            $scriptr.Options.FileName = "$SavePath\$TypeFolder\$ScriptFile.sql"
            #每次產生一個物件的 T-SQL 定義

```

```
[void]$scripttr.Script($ScriptThis)
    }
}
}
```

在範例程式 D.15 中，若想要增加產生的執行個體等級物件，例如「連結同的服器」，只需要在 `$ServerObjects` 集合變數中再加入代表連結同服器的集合物件，範例如下：

```
$ServerObjects += $SMOServer.LinkedServers
```

同理，若要建立資料庫內的物件集合，以「結構描述」為例，只需要增加 `$Objects` 集合變數內代表「結構描述」的集合物件，範例如下：

```
$Objects += $db.Schemas
```

範例程式 D.15 執行完畢後，會依據目錄結構與產出時間分別擺放，你可以定期產生所需物件的 DDL 定義，藉此管理與追蹤 SQL Server 執行個體的各種定義。

D.4 dbaTools 模組

dbaTools 是開源的 PowerShell 模組，由 SQL Server MVP Chrissy LeMaire 起始，隨後受到大量開源開發者的支持，累積成日益完備的工具集。

由於資料庫管理師（DBA）日常的維運中，隨著各自的用途不同，架設與整合的環境各異，需要設定與監控的工作雖有大者相似，例如：監控容量、效能、確定備份排程正常、解讀錯誤事件...等，但也有許多各自的差異，例如：要整合 AD Kerberos 的帳號身分 Delegation；做跨同服器的認證，需要正確設定 Service Principal Names（SPN）。這並

非一般 DBA 需要完成的，大多數環境也不需要，也因此若一旦要設定，往往 DBA 需要翻找許多文件，而 dbaTools 就將其整合成一組 Get/Remove/Set/Test-Dbaspn 指令，讓 DBA 免去熟悉 AD 相關工具的麻煩。

對大多數 DBA 而言，在豐富的工具組內翻找相似的指令結構，解決大不相同的工作內容，去異求同在 IT 永遠是個好選擇。

在管理者工作的機器上（不需要在 SQL Server 伺服器安裝，一般管理者是遠端透過指令完成管理工作），直接透過網際網路安裝 dbaTools PowerShell 模組，可以簡單執行指令：

```
Install-Module dbatools
```

如某台機器一旦下載安裝了 dbaTools 模組，其他機器也可簡單複製以下目錄內容，就能完成安裝 PowerShell 模組：

```
C:\Program Files\WindowsPowerShell\Modules\dbatools
```

或是參照以下網頁的說明：

```
https://dbatools.io/download/
```

在撰寫本章時（2022/8），以如下的指令簡單估算 dbaTools 模組已有的指令達 679 個：

```
Get-Command -Module dbatools | measure
```

而在以下的網頁有分門別類列出各項指令：

```
https://dbatools.io/commands/
```

或是

<https://docs.dbatools.io/>

在 <https://docs.dbatools.io> 右方的網頁中，以字型大小呈現了各大類型指令的數量，如圖 D.24 所示：

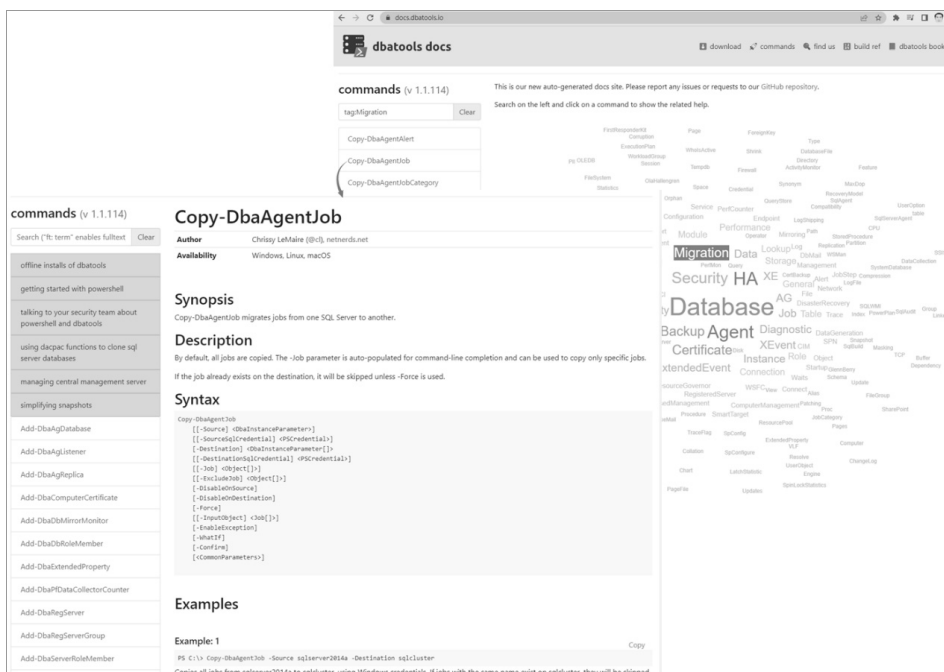


圖 D.24 在右方點選某項大類工作後，左方視窗會動態表列所屬指令

在右方網頁點選代表該類型的字，如圖 D.24 點選 Migration，左方列表便會呈現相關的指令。而後點選任一指令連結，也可以檢視到該指令的使用說明。相信隨著 SQL Server 2022 版正式面市後，將會有更多的指令被眾人開發進去。

dbatools 的 cmdlet 指令命名規則採：

<動詞>-Dba<名詞>

換句話說，所有的名詞前接有 DbA 字樣，這也就不會與其他模組（如本章前述介紹的 sqlps 或 sqlserver 等模組）的指令混淆。

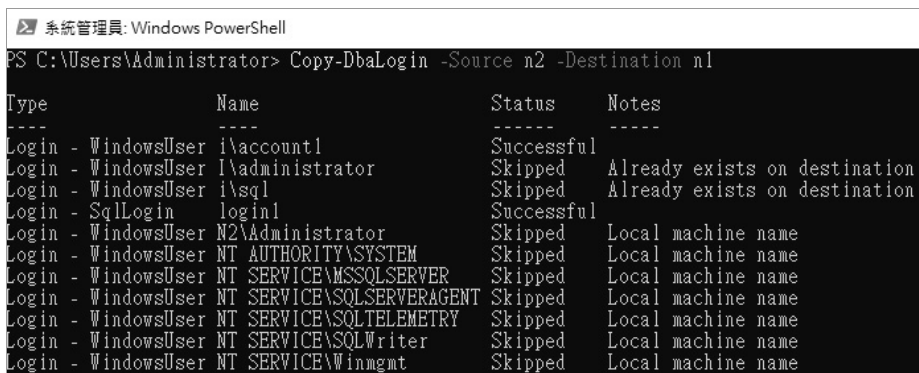
簡單示範 dbaTools 模組提供的 Copy-DbLogin，從 n2 伺服器將登入複製到 n1 伺服器。首先在 n2 建立兩個登入：

```
create login login1 with password='!QAZxsw2'
create login [i\account1] from windows
```

而後在 PowerShell 命令提示列可輸入以下指令：

```
Copy-DbLogin -Source n2 -Destination n1
```

執行結果如圖 D.25 所示：



```

系統管理員: Windows PowerShell
PS C:\Users\Administrator> Copy-DbLogin -Source n2 -Destination n1

Type           Name           Status      Notes
----           -
Login - WindowsUser i\account1      Successful
Login - WindowsUser I\administrator    Skipped    Already exists on destination
Login - WindowsUser i\sql               Skipped    Already exists on destination
Login - SqlLogin login1                 Successful
Login - WindowsUser N2\Administrator    Skipped    Local machine name
Login - WindowsUser NT AUTHORITY\SYSTEM Skipped    Local machine name
Login - WindowsUser NT SERVICE\MSSQLSERVER Skipped    Local machine name
Login - WindowsUser NT SERVICE\SQLSERVERAGENT Skipped    Local machine name
Login - WindowsUser NT SERVICE\SQLTELEMETRY Skipped    Local machine name
Login - WindowsUser NT SERVICE\SQLWriter Skipped    Local machine name
Login - WindowsUser NT SERVICE\Winmgmt Skipped    Local machine name
    
```

圖 D.25 透過 Copy-DbLogin 同步兩個 SQL Server 執行個體的登入

從圖 D.25 可以看到先前建立的兩種登入都複製到 n1 伺服器內。在 SSMS 以 sqlcmd 模式執行以下查詢：

```
select @@servername serverName, name,sid,convert(varchar(256),password_hash,1)
hashPwd from sys.sql_logins where name='login1'
```

其查詢結果如圖 D.26 所示：

```

:connect n1
select @@servername, name, sid, convert(varchar(256), password_hash, 1) hashPwd from sys.sql_logins where name='login1'
go
:connect n2
select @@servername, name, sid, convert(varchar(256), password_hash, 1) hashPwd from sys.sql_logins where name='login1'

```

結果			
N1	login1	0x124DAD9429C28644BC103E521A533DA4	0x020021A399F969BAE5636908A9F55EC21420EFD0DAF0F7FD04F03AEB854CBD76786B1347695BC
(沒有資料行名稱)			
N2	login1	0x124DAD9429C28644BC103E521A533DA4	0x020021A399F969BAE5636908A9F55EC21420EFD0DAF0F7FD04F03AEB854CBD76786B1347695BC

圖 D.26 透過 sys.sql_logins 系統檢視確認 sid 和密碼雜湊值是否相同

由於移轉 SQL Server 自訂帳號時，要能保持 sid 和密碼雜湊值相同，在目的伺服器環境才能沿用該登入，而由圖 D.26 的查詢結果可以確認 Copy-DbaLogin 指令可以維持兩者的一致。

dbaTools 指令眾多，而各項工作又都與相關的管理工作有更多的理論連結，在此篇幅有限不進一步介紹，僅能請你依自己的需求組織腳本，讓日常的工作得以自動化地完成。