KAPITOLA 3

Filtrování, seskupování a třídění

V této kapitole:
- Úvod do zřetězení
- Třídění výstupu z rutiny
- Seskupení výstupu po třídění
- Filtrování výstupu z jedné rutiny
- Filtrování výstupu z jedné rutiny před tříděním
- Shrnutí

Prostředí Windows PowerShell vyniká mimo jiné ve zpřístupnění dat. Obvykle se přitom využívá odeslání dat do řetězce rutin. Řetězení v praxi představuje základní princip prostředí Windows PowerShell a nelze si bez něj představit třídění a seskupování dat a filtrování konkrétních informací z kolekcí jiných údajů. Při práci s prostředím Windows PowerShell budete možnosti zřetězení rutinně využívat.

Úvod do zřetězení

Řetězení prostředí Windows PowerShell přebírá výstup z jednoho příkazu a odesílá jej jako vstup jinému příkazu. Pomocí zřetězení lze zvládnout různé úkoly, například vyhledání všech počítačů v jednom konkrétním umístění a jejich restart. Přitom je potřeba jeden příkaz na vyhledání počítačů v daném umístění a další příkaz, který všechny nalezené počítače restartuje. Díky předávání objektů z jednoho příkazu novému příkazu se prostředí Windows PowerShell v konzole snadno používá, protože není nutné před použitím druhého příkazu přerušit práci a zpracovat výstup prvního příkazu.

Prostředí Windows PowerShell předává objekty pomocí zřetězení. Mimo jiné díky tomu je prostředí Windows PowerShell značně efektivní: Z výsledků spuštění jednoho příkazu přebírá objekt (nebo skupinu objektů) a předává je vstupu dalšího příkazu. Pomocí zřetězení prostředí Windows PowerShell není nutné ukládat výsledky jednoho příkazu do proměnné a poté volat metodu daného objektu, která by provedla požadovanou akci. Následující příkaz například vypne všechny síťové adaptéry v počítači se systémem Windows 8:

```
Get-NetAdapter | Disable-NetAdapter
```

Poznámka: Prostředí Windows PowerShell dodržuje zásady zabezpečení. Chcete-li tedy vypnout síťový adaptér, musíte prostředí Windows PowerShell spustit s právy správce. Další informace o spuštění prostředí Windows PowerShell s právy správce naleznete v kapitole 1.

Kromě vypnutí všech síťových adaptérů je také můžete zapnout. V tomto případě použijte rutinu *Get-NetAdapter* a přesměrujte výsledky do rutiny *Enable-Netadapter*, jak je patrné v následující ukázce:

```
Get-NetAdapter | Enable-NetAdapter
```

Jestliže chcete spustit všechny virtuální počítače v systému Windows 8 nebo Windows Server 2012, použijte rutinu *Get-VM* a přesměrujte výsledné objekty virtuálních počítačů do rutiny *Start-VM* (viz následující příklad):

```
Get-VM | Start-VM
```

Pomocí rutiny *Get-VM* můžete vyhledat virtuální počítače a poté je zastavit přesměrováním výsledných objektů virtuálních počítačů do rutiny *Stop-VM*, jak je zřejmé v následujícím příkladu:

```
Get-VM | Stop-VM
```

V každém z předchozích příkazů je objekt (nebo skupina objektů) vytvořený jedním příkazem přesměrován k dalšímu zpracování do jiné rutiny.

Třídění výstupu z rutiny

Rutina *Get-Process* generuje přehledné tabulkové zobrazení informací o procesech v konzole prostředí Windows PowerShell. Výchozí zobrazení využívá abecední řazení podle názvu ve vzestupném pořadí. Toto zobrazení se hodí při hledání informací o určitém procesu, ale skrývá důležité detaily, například to, který proces používá nejméně nebo nejvíce virtuální paměti. Chcete-li setřídit výstup z tabulky procesů, zřetězte výsledky z rutiny *Get-Process* s rutinou *Sort-Object* a v parametru -Property uveďte vlastnost, podle níž chcete třídit. Výchozí pořadí řazení je vzestupné (to znamená, že nejmenší čísla se zobrazují na začátku seznamu). Následující příkaz setřídí výstup procesů podle rozsahu virtuální paměti využité každým procesem:

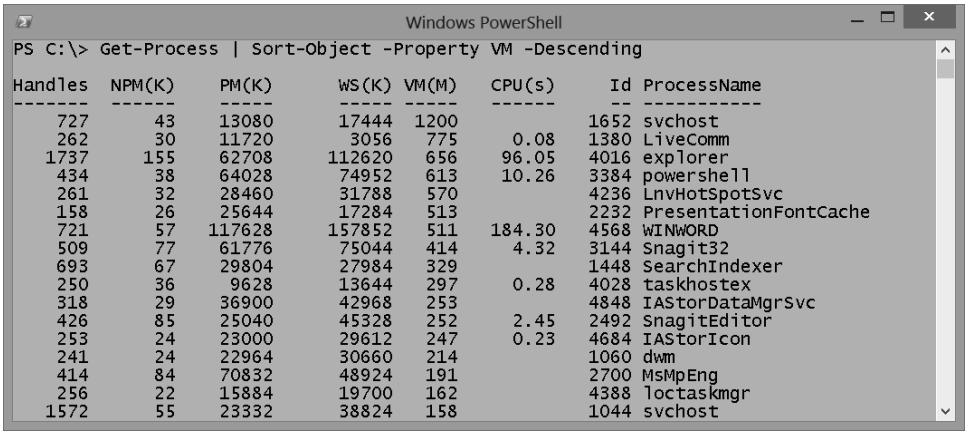
```
Get-Process | Sort-Object -Property VM
```

Procesy, které spotřebovávají nejmenší množství paměti, jsou uvedeny na začátku seznamu.

Pokud vás zajímá, které procesy vyžadují nejvíce virtuální paměti, můžete výchozí pořadí řazení obrátit. K tomu slouží parametr přepínače -Descending, který se uplatňuje v následujícím příkladu:

```
Get-Process | Sort-Object -Property VM -Descending
```

Obrázek 3.1 znázorňuje příkaz, který poskytne seznam procesů setříděných podle virtuální paměti, a výstup tohoto příkazu.



Obrázek 3.1: Pomocí rutiny Sort-Object lze organizovat výstup objektů s ohledem na lepší čitelnost

Příkazy prostředí Windows PowerShell, které obsahují rutinu *Sort-Object*, lze zkrátit. Rutina *Sort-Object* má alias *Sort*. Alias rutiny je zkrácená forma jejího názvu, kterou prostředí Windows PowerShell rozpoznává jako rovnocennou formu úplného názvu rutiny. Některé aliasy lze snadno rozpoznat, například sort pro rutinu *Sort-Object* nebo select pro rutinu *Select-Object*. Jiné aliasy se musíte naučit. K nim patří *?* (symbol otazníku) pro rutinu *Where-Object*. Většina uživatelů systému Windows by předpokládala, že symbol *?* bude alias rutiny *Get-Help*.

Kromě použití aliasu pro název rutiny *Sort-Object* lze příkaz zkrátit díky tomu, že parametr -Property je pro tuto rutinu výchozí, a proto jej můžeme z příkazu vynechat. Následující příkaz poskytne seznam služeb podle jejich stavu s využitím zkrácené syntaxe:

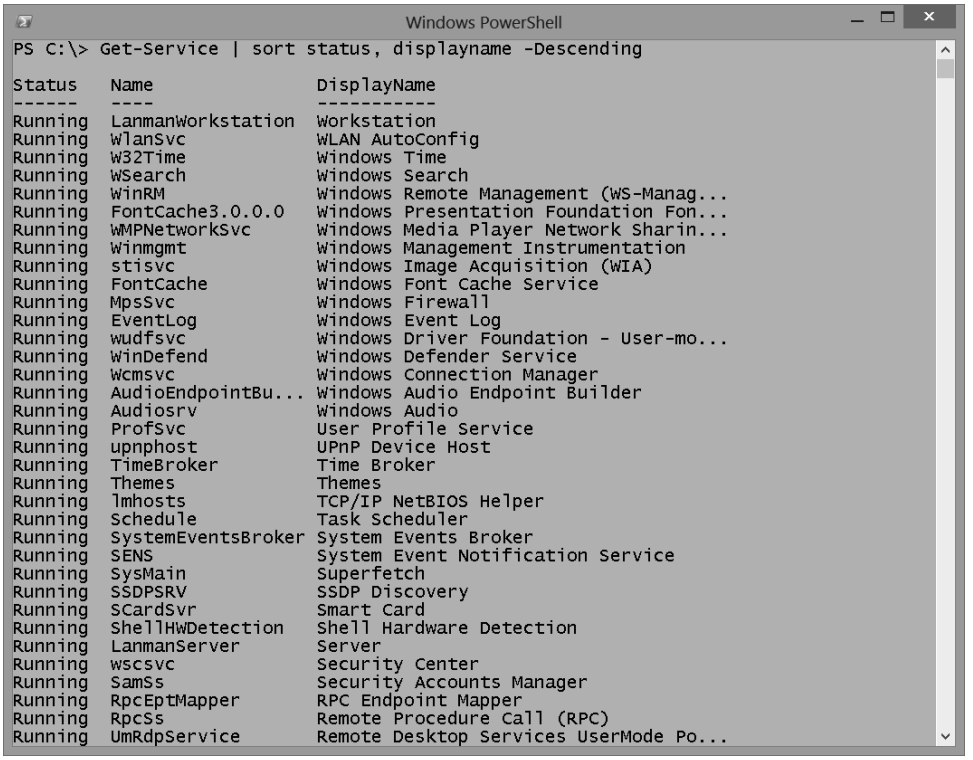
```
Get-Service | sort status
```

Je možné třídit podle více vlastností. Přitom je nutné dbát opatrnosti, protože v některých případech nelze dodatečné vlastnosti třídit. U služeb dává vícenásobné třídění smysl, protože existují dvě široké kategorie stavu: Running (spuštěno) a Stopped (zastaveno). Je tedy vhodné se pokusit o další uspořádání výstupu, aby bylo možné najít určité zastavené či spuštěné služby. Jeden způsob, který usnadňuje nalezení služeb, spočívá v abecedním třídění podle vlastnosti displayname každé služby. Následující příklad setřídí objekty služeb získané pomocí rutiny *Get-Service* podle stavu a následně v rámci stavu podle hodnoty *displayname*:

```
Get-Service | sort status, displayname -Descending
```

Výstup se zobrazuje v sestupném pořadí namísto výchozího seznamu ve vzestupném řazení.

Obrázek 3.2 představuje příkaz, který setřídí služby podle stavu a hodnoty *displayname*, a také výstup tohoto příkazu.



Obrázek 3.2: Uspořádání výstupu rutiny Get-Service pomocí rutiny Sort-Object

Seskupení výstupu po třídění

Jakmile setřídíte objekty předávané pomocí zřetězení, můžete je seskupit. Je důležité setřídit objekty ještě před seskupením. Tím dosáhnete nejvyššího výkonu a nejpřesnějších výsledků. Chcete-li seskupit počet spuštěných nebo zastavených služeb, načtěte objekty služeb pomocí rutiny *Get-Service*. Zřetězte výsledné objekty služeb do rutiny *Sort-Object* a setřiďte je podle vlastnosti status. Nakonec zřetězte setříděné objekty služeb do rutiny *Group-Object* a uveďte, že je chcete seskupit podle vlastnosti status. Následující ukázka představuje podobu výsledného příkazu a příslušný výstup:

```
PS C:\> Get-Service | Sort-Object status  | Group-Object -Property status

Count   Name                Group
-----   ----                -----
   99   Stopped             {PNRPsvc, p2pimsvc, ose, TrustedInstaller...}
   83   Running             {vmms, wudfsvc, Wcmsvc, stisvc...}
```

Poznámka: Používáte-li rutinu *Group-Object*, je zásadně důležité, abyste vybrali konkrétní vlastnost, podle které chcete objekty seskupit. Vlastnost, podle níž budete seskupovat, by měla odpovídat vlastnosti, podle které třídíte. Pokud neuvedete objekt, podle něhož chcete objekty seskupit, příkaz bude zdánlivě funkční, ale bude poskytovat nekonzistentní výsledky.

Délku příkazu s rutinou *Group-Object* lze zkrátit pomocí aliasu *Group*, díky němuž není nutné vypisovat celý název *Group-object*. Kromě toho parametr -Property je výchozí a lze jej vypustit. Následující příklad představuje zkrácenou verzi příkazu, který zobrazí počet spuštěných a zastavených služeb:

```
PS C:\> Get-Service | sort status | group status

Count   Name                    Group
-----   ----                    -----
   95   Stopped                 {PNRPsvc, p2psvc, p2pimsvc, ose...}
   87   Running                 {wudfsvc, SysMain, Wcmsvc, wuauserv...}
```

Seskupení informací bez dat prvků

Rutina *Group-Object* standardně zobrazuje tři vlastnosti: *Count*, *Name* a *Group*. Vlastnost *Group* obsahuje data související s hodnotami, které se objevují pod vlastností *Name*. Data ve sloupci Group jsou často užitečná, protože poskytují příklady typů hodnot, které jsou k dispozici pro seskupení. Občas je však tento výstup rušivý. Předchozí příklad s použitím rutiny *Group-Object* obsahuje sloupec, který nabízí malou nápovědu ohledně seskupených položek. Prostředí Windows PowerShell standardně ve sloupci Group zobrazuje čtyři položky, což obvykle poskytuje dostatečnou představu o tom, jak příkaz proběhl. Na druhou stranu takový sloupec plný závorek a trojic teček často jen zabírá místo na obrazovce. Když například pracujete s protokolem událostí, ve sloupci skupin se zobrazují datové typy položek. Tyto informace jsou pro všechny položky protokolu podobné a proto jsou zbytečné. Následující příklad znázorňuje pět nejnovějších událostí chyb z protokolu aplikací:

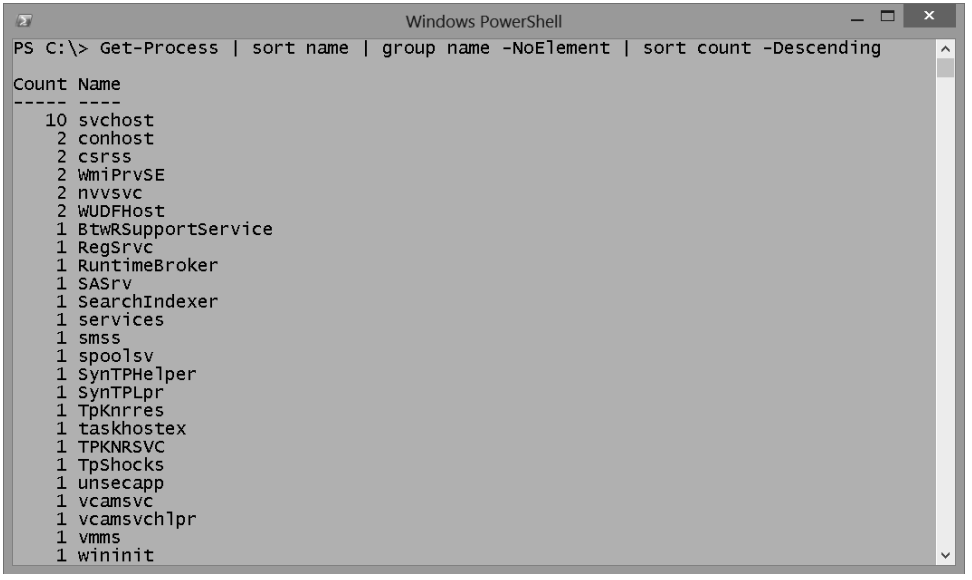
```
12:47 C:\> Get-EventLog -Log application -EntryType error -new 5|
sort message | group message

Count   Name                        Group
-----   ----                        -----
    1   (GetHomepage()): Faile...   {System.Diagnostics.EventLogEntry}
    1   (GetHomepages()): Fail...   {System.Diagnostics.EventLogEntry}
    1   (PerformTasks()): Fail...   {System.Diagnostics.EventLogEntry}
    2   App DefaultBrowser_NOP...   {System.Diagnostics.EventLogEntry,
                                     System.Diagnos...
```

Pět chybových zpráv je setříděno podle hodnoty zprávy a zobrazeno v konzole prostředí Windows PowerShell. Všimněte si, že sloupec Group v zobrazeném výstupu zabírá místo a data, která nesou nejdůležitější informace – ve sloupci Name – jsou zkrácena natolik, že se téměř nedají použít.

Pokud víte, že příkaz vrátí požadované informace, a zajímají vás pouze počty ve skupinách, můžete pomocí rutiny *Group-Object* vrátit pouze skupiny vlastností a počet položek v dané skupině. Klíčem je parametr přepínače -NoElement. Následující příkaz seskupí procesy podle názvu a následně setřídí počty procesů v sestupném pořadí:

```
Get-Process | sort name | group name -NoElement | sort count -Descending
```

Na obrázku 3.3 vidíme příkaz, který zjistí spuštěné procesy, setřídí je podle názvu, tyto názvy seskupí a setřídí počet spuštěných procesů podle názvu. Tento obrázek rovněž představuje výstup příkazu.



Obrázek 3.3: Díky parametru přepínače -NoElement se seskupená data nezobrazují v konzole prostředí Windows PowerShell

Filtrování výstupu z jedné rutiny

Seskupení a třídění dat je užitečné, protože poskytuje obecný přehled o dotazovaných datech. Filtrování dat však umožňuje analyzovat data hlouběji a rozpoznat relevantní vzory. Můžete si například dlouho prohlížet výstup rutiny *Get-Process*, než zjistíte, že nějaký proces v systému používá 1 000 MB virtuální paměti. Díky jednoduchému filtru *Where-Object* je však stejná informace najednou úplně zjevná. Následující ukázka znázorňuje příkaz, který načte informace o procesech a vyfiltruje všechny procesy, které spotřebovávají více než 1 000 MB virtuální paměti:

```
PS C:\> Get-Process | Where-Object vm -gt 1000MB

Handles   NPM(K)     PM(K)       WS(K) VM(M)     CPU(s)      Id ProcessName
-------   ------     -----       ----- -----     ------      -- -----------
    762       46     12824       17140 1203                1656 svchost
```

Filtrování podle data

Při řešení problémů je často potřeba zkontrolovat, co se stalo po určitém datu. Některé rutiny umožňují přímé filtrování podle data, zatímco jiné nikoli. Jestliže rutina nemá parametr, který by dovoloval filtrovat podle data, stačí pouze přesměrovat vrácený objekt do rutiny *Where-Object*.

Funkce *Get-WindowsDriver* z modulu DISM (Deployment Image Servicing and Management) v systému Windows 8 načítá seznam všech ovladačů nainstalovaných v počítači. K dispozici je sice několik parametrů, které umožňují filtrovat ovladače na levé straně znaku zřetězení, ale možnost filtrování podle data dostupná není. Chcete-li najít ovladače nainstalované po určitém datu, přesměrujte výsledky rutiny *Get-WindowsDriver* do rutiny *Where-Object* a určete, že máte zájem pouze o ovladače nainstalované po konkrétním datu. Tento příkaz je uveden v následující ukázce:

```
PS C:\> Get-WindowsDriver -Online | where date -gt 10/8/2012

Published Name     : oem26.inf
Original File Name : C:\Windows\System32\DriverStore\
                     FileRepository\ibmpmdrv.inf_amd
                     64_728348017c675c91\ibmpmdrv.inf
InBox              : False
Class Name         : System
Boot Critical      : True
Provider Name      : Lenovo
Date               : 10/9/2012 12:00:00 AM
Version            : 1.66.0.17
```

Poznámka: K použití funkce *Get-WindowsDriver* v systému Windows 8 jsou potřebná zvýšená oprávnění.

Rutina *Sort-Object* dokáže data dobře zpracovat. Díky třídění dat lze snadno identifikovat podobné události. Následující příkaz například vrátí všechny opravy hotfix, které byly do místního počítače nainstalovány po 1. prosinci 2012:

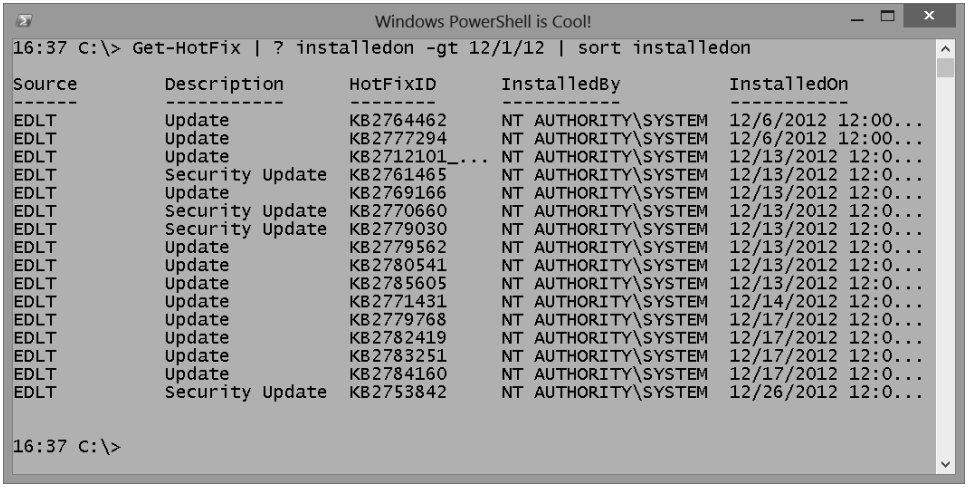
```
Get-HotFix | Where installedon -gt 12/1/12
```

Jediný problém s příkazem, který vypíše opravy hotfix nainstalované po 1. prosinci 2012, spočívá v tom, že od té doby bylo oprav hotfix vydáno mnoho. Kvůli tomu může být těžké najít konkrétní opravu. Jestliže však výsledky přesměrujete do rutiny *Sort-Object*, můžete konkrétní

opravy hotfix snadno identifikovat. Výchozí vzestupné řazení se v praxi pro úlohy související s daty ideálně hodí. Příkaz je uveden v následující ukázce:

```
Get-HotFix | Where installedon -gt 12/1/12 | sort installedon
```

Obrázek 3.4 představuje příkaz, který vyhledá a setřídí všechny opravy hotfix nainstalované po 1. prosinci 2012, a příslušný výstup příkazu.



Obrázek 3.4: Třídění dat po filtrování podle data umožňuje snadno identifikovat datové vzory

Objekt, který vrací data, se označuje jako *DateTime*. Objekt *DateTime* sestává z mnoha metod a vlastností. Vlastnosti objektu *DateTime* se využívají při filtrování. Chcete-li zobrazit vlastnosti objektu *DateTime*, zřetězte datum do rutiny *Get-Member*. Následující ukázka prezentuje příkaz, který vrátí vlastnosti objektu *DateTime*, a výstup tohoto příkazu:

```
16:59 C:\> Get-Date | Get-Member -MemberType Property

   TypeName: System.DateTime

Name          MemberType    Definition
----          ----------    ----------
Date          Property      datetime Date {get;}
Day           Property      int Day {get;}
DayOfWeek     Property      System.DayOfWeek DayOfWeek {get;}
DayOfYear     Property      int DayOfYear {get;}
Hour          Property      int Hour {get;}
Kind          Property      System.DateTimeKind Kind {get;}
Millisecond   Property      int Millisecond {get;}
Minute        Property      int Minute {get;}
Month         Property      int Month {get;}
Second        Property      int Second {get;}
Ticks         Property      long Ticks {get;}
TimeOfDay     Property      timespan TimeOfDay {get;}
Year          Property      int Year {get;}
```

Čas spuštění procesu je oznámen formou objektu *DateTime*. Rutina *Get-Member* tyto informace poskytuje na základě následujícího příkazu:

```
17:03 C:\> Get-Process | Get-Member -Name starttime

   TypeName: System.Diagnostics.Process

Name      MemberType Definition
----      ---------- ----------
StartTime Property   datetime StartTime {get;}
```

Vzhledem k tomu, že vlastnost StartTime obsahuje instanci objektu *DateTime*, lze vytvořit filtr z libovolné vlastnosti daného objektu. Chcete-li filtrovat čas podle hodin, minut, sekund nebo libovolné jiné vlastnosti objektu *DateTime*, potřebujete k tomu složitější formu rutiny *Where-Object*. Tato forma začíná a končí výrazem *ScriptBlock*. Blok *ScriptBlock* je uzavřen do složených závorek ({}). V rámci bloku *ScriptBlock* se používá automatická proměnná *$_*, která umožňuje získat přístup ke každé položce procházející zřetězením. Poté funguje podobně jako kterýkoli jiný filtr. Zvolte vlastnost z objektu a pomocí operátoru větší než (>) určete, podle které minuty chcete filtrovat (viz následující příklad):

```
17:06 C:\> Get-Process | Where { $_.starttime.minute -gt 55} |
select name, starttime

Name                 StartTime
----                 ---------
IAStorIcon           1/5/2013 2:56:32 PM
WINWORD              1/5/2013 2:58:25 PM
```

Filtrování vlevo

Z hlediska výkonu je nejvhodnější provádět většinu akcí s daty na levé straně znaku zřetězení. Důvod spočívá v tom, že spuštěná rutina může vrátit hodně dat. I když pracujete lokálně, vrácení velkého objemu dat může klást značné nároky na paměť, čas procesoru a vstupně-výstupní diskové operace. Vracíte-li data po síti, pak neefektivní dotaz navíc způsobuje potíže s kapacitou sítě. Obecně platí, že pracujete-li lokálně (nebo vzdáleně s rychlým síťovým připojením), optimalizace postupně dává stále menší smysl. Jestliže například strávíte 10 minut zdokonalováním dotazu, jehož spuštění trvá jen několik sekund (bez optimalizace), pravděpodobně jste promarnili 9,9 minuty, které jste mohli využít efektivněji. Pokud na druhou stranu neoptimalizovaný dotaz běží 10 minut a budete jej spouštět na 100 vzdálených serverech, z nichž některé mají značně omezenou šířku pásma, pak bude rozumné věnovat na optimalizaci dotazu několik dní.

Jak lze tedy filtrovat vlevo od znaku zřetězení? Umožňují to filtrovací možnosti samotné rutiny. Následující příkaz například vrátí všechny položky protokolu událostí z protokolu aplikací:

```
Get-EventLog -LogName application
```