目次

Windows スクリプティング（WSH、PowerShell）

はじめに

平成３０(2018)年９月７日に経済産業省から「DX レポート～IT システム「2025 年の崖」の克服と DX の本格的な展開～」[1]という文書で、既存システムのブラックボックス化によりデータ活用に支障があり 2025 年以降 最大で年間 12 兆円の経済損失の可能性を述べています。朗報としては、2025 年以降の展望の一つが“（IT 人材平均年収） 2017 年時点の 2 倍程度 （米国並み）”だそうです（2017 年の IT 人材平均年収は約 600 万円…同資料より）。

問題は DX（デジタル・トランスフォーメーション）実現の条件として人材不足の解消や新たなデジタル技術の導入云々を挙げていますが、これは過去何十年間解決できなかった課題であり、最近（2022 年６月）のニュースでも IT 業界は賃金水準が低くて人材が集まっていないとのことです。

DX よりも数年早くバズワードになった単語に RPA(Robotic Process Automation)があります。RPA はオフィスのデジタル化を行う、DX の具体化手段の一つです。日本では 2017 年頃に米国産 RPA ソフトが発売され、導入して成果が出ている（ライセンスを追加した）企業もあります。

RPA で自動化できるのは Web ブラウザを制御してデータを取得、Excel ファイルのデータ処理、送受信メールの処理、SQL クエリの発行 等々の作業ですが、実はここに挙げた機能は全て VBA マクロや AutoIt、Selenium 等のフリー／オープンソフトを使えば実現できます。RPA が独自に追加しているのはプロセスの実行権限の管理やプロセス定義のサーバ集中管理、そしてプロセスの実行を手順（スクリプト）化したボット（製品により呼び名は変わります）という概念です。RPA は登場当初は事務系の人員削減を謳い文句にしていました（ライセンス料は削減した給与を上回りかねないほどでしたが）が、実際に導入効果があったという企業では人員削減効果よりも工場で生産指示を自動化したり、伝票入力のスピードアップ、正確性の向上等に効果を感じていると聞きました。

RPA は VBA マクロをボット化して外部で集中管理ができるようにしましたが、この外部から Excel 等を制御することも Windows の初期から搭載されているスクリプトの機能を使って実現することができます。

1. Windows のスクリプト言語・環境

Windows に組み込まれているスクリプト言語には DOS の時代から存在する bat（ファイル）、当初 Internet Exproler3.0（IE）用に作られその後 WSH（Windows Script Host）で動作するようになった JScript と VBScript、Windows XP 以後は PowerShell が提供されています。

その他にサードパーティ製では、オープンソースの有名なもので Perl、Python、PHP、Ruby、JavaScript/NODE.js 他があります。2014 年以降は PowerShell も.Net と共にオープンソースになり Linux や MacOS でも動作するようになっています。

WSH は PowerShell が発表された時期以降 新規機能の追加は止まっていますが、マイクロソフト社のサイトを見ても廃止するようなコメントは出ておらず、Window 11 にもインストールされています。

---

[1] DX レポート
https://www.meti.go.jp/shingikai/mono_info_service/digital_transformation/pdf/20180907_01.pdf
3 年後に主にコロナの影響を論じた「DX レポート 2 中間報告」
https://www.meti.go.jp/press/2020/12/20201228004/20201228004-3.pdf

2. WSH（VBScript）

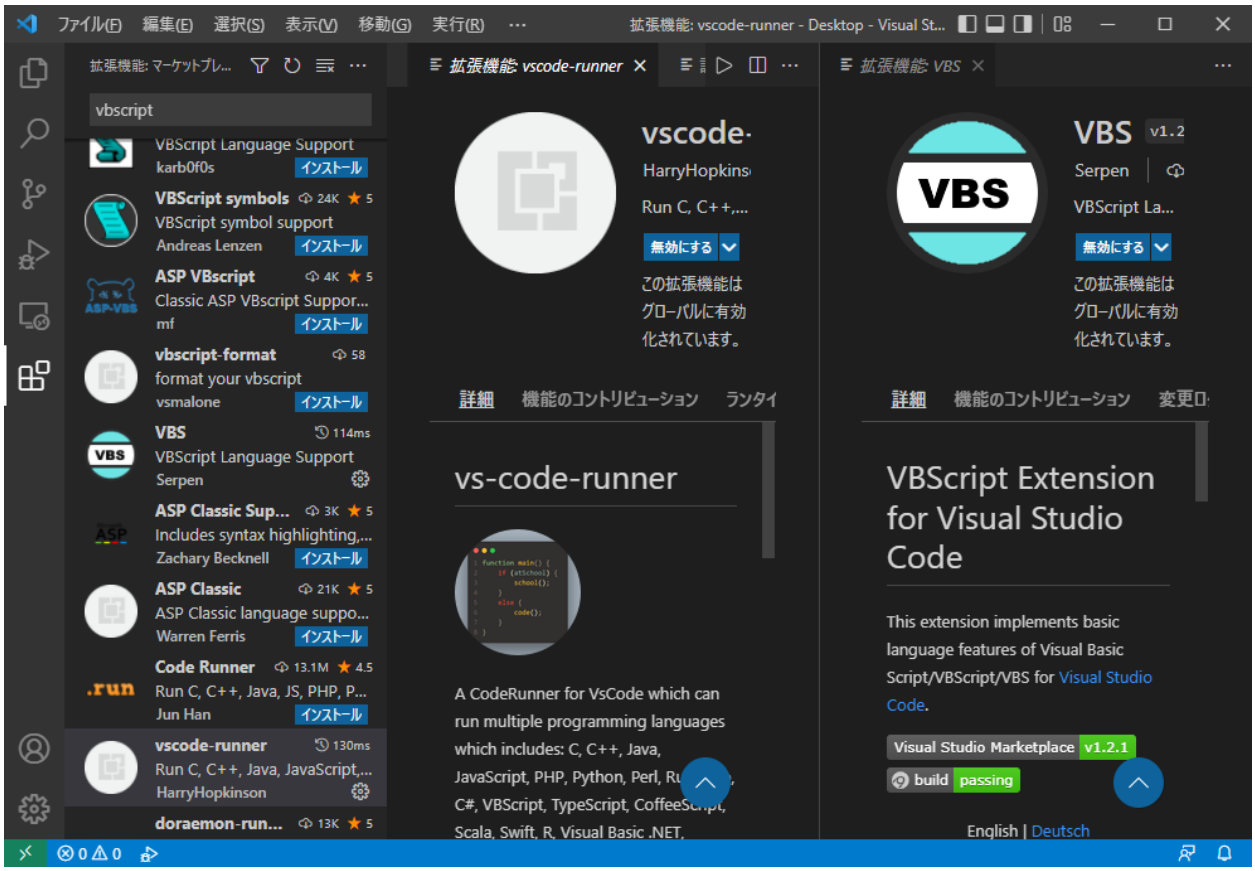
WSH の環境は VBScript と JScript の動作環境が用意されています。JScript は JavaScript と同じく ECMAScript という言語規約の実装を謳っていますが、WSH 自体 2009 年の Ver. 5.6 から機能追加は行われていないのでその後に進化した JavaScript との互換性は将来にわたって期待できません。

VBScript は Excel 等の Office 製品に組み込みの VBA（Microsoft Visual Basic for Applications）言語のサブセット[2]で、元となった Visual Basic が既に開発を終わっているので今後 機能追加は行われないでしょうが、VBA と同じコードが動作します。以下、使用事例を示します。

2.1. 開発環境

（1）エディターと機能拡張

WSH 専用の開発環境というものは無く汎用的なエディターで作成します。VBScript はメモ帳で作れる程度のものに向いていますが、少し長くなってきたらマイクロソフト社がオープンソースで公開しているエディターの VSC(Visual Studio Code)向けに VBScript 用の機能拡張が幾つか開発されています。



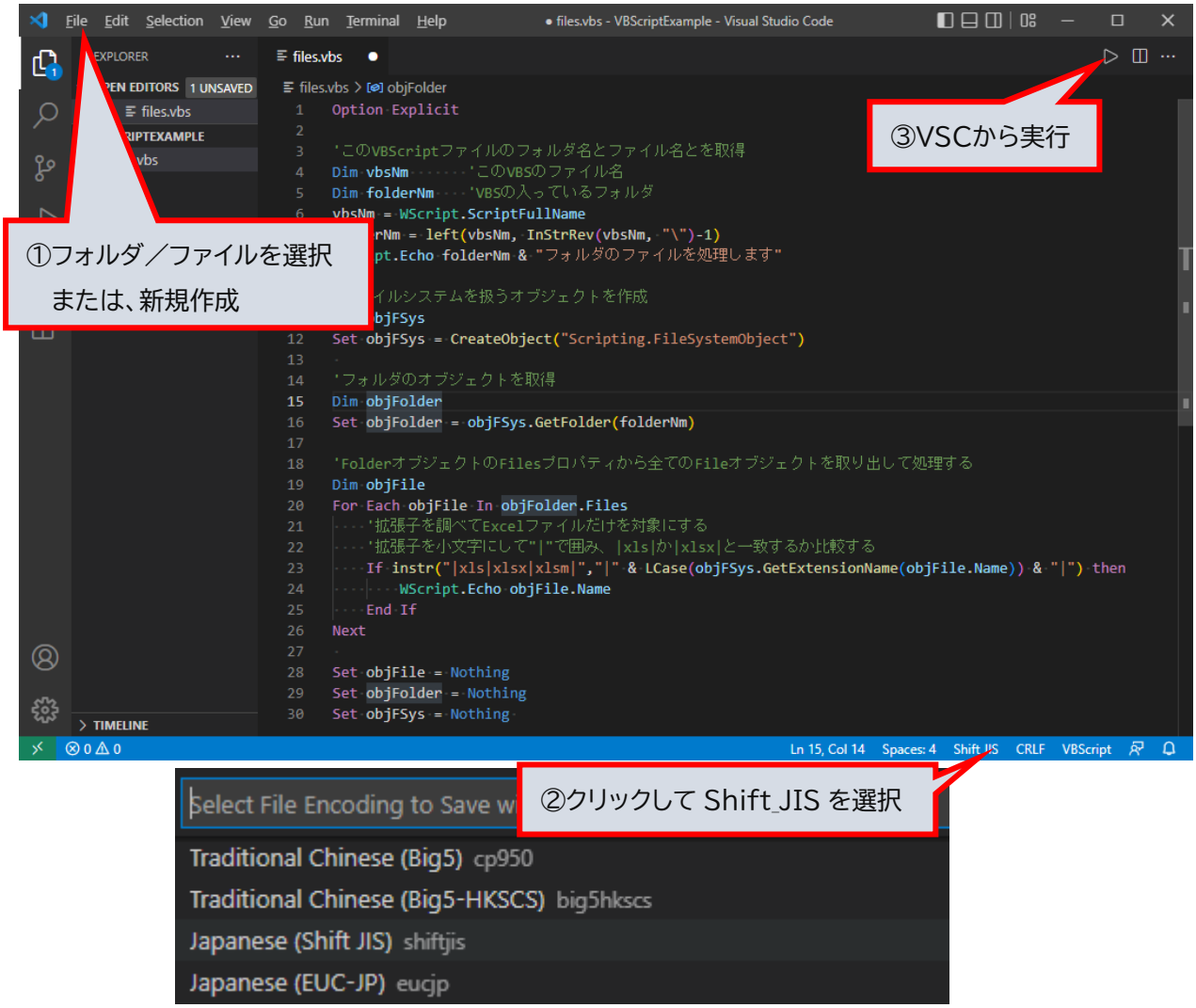
以降は、コード入力補助（候補表示 他）に VBS とコード実行に vs-code-runner の 2 つの機能拡張（どちらも変更や再配布を認める MIT License です）をインストールして実行したサンプルを挙げます。

---

[2] com 言語 VBScript への翻訳
https://docs.microsoft.com/ja-jp/windows/win32/com/translating-to-vbscript

（2）文字コード

VBScript は Shift_JIS（CP932：コードページ 932）を使います。VSC を含む殆どのエディターは UTF-8 がデフォルトになっているので、コーディングを開始する前に Shift_JIS に変えてください。



2.2. コード例

以下のコードをコピーし、エディターで拡張子 vbs のファイルを作ってペーストしてください。文字コードはエディタにより表記が異なりますが Shift_JIS、SJIS、ANSI のいずれかを選んで保存します。

保存するとこのようなアイコンが表示されます。 

このファイルをダブルクリックすると実行され、WScript.Echo の部分でポップアップを表示して処理が中断します。

ポップアップの中断なしで処理を行いたい場合は、vbs ファイルを cscript コマンドで実行します。

> cscript "c:¥Users¥User¥Desktop¥VBScriptExample¥files.vbs"

2.2.1. ファイル選択と処理

VBScript はファイル選択ダイアログが使えないので、この例では vbs ファイルを処理対象のファイルと同じフォルダに入れる前提で作ってあります。主要な部分は以下の 3 カ所です。

① 行 6､7> vbs ファイルのパスを取得し、最後の”¥”の前迄をフォルダ名として取り出し

② 行 16 行> フォルダ名をオブジェクト化

③ 行 20> ②で作ったフォルダオブジェクトの中のファイルを 1 件づつ objFile に取り出し、23 行めで拡張子が Excel の拡張子に含まれていたら処理（この例では表示）の対象にする。
23 行めは If 文を 2 行に分けています。(行末の“_”が継続のしるし)

```
1  Option Explicit '宣言(Dim)無しの変数は使用新規
2
3  'この VBScript ファイルのフォルダ名とファイル名とを取得
4  Dim vbsNm        'この VBS のファイル名
5  Dim folderNm     'VBS の入っているフォルダ
6  vbsNm = WScript.ScriptFullName
7  folderNm = left(vbsNm, InStrRev(vbsNm, "¥")-1)
8  WScript.Echo folderNm & "フォルダのファイルを処理します"
9
10 'ファイルシステムを扱うオブジェクトを作成
11 Dim objFSys
12 Set objFSys = CreateObject("Scripting.FileSystemObject")
13
14 'フォルダのオブジェクトを取得
15 Dim objFolder
16 Set objFolder = objFSys.GetFolder(folderNm)
17
18 'Folder オブジェクトの Files プロパティから全ての File オブジェクトを取り出して処理する
19 Dim objFile
20 For Each objFile In objFolder.Files
21     '拡張子を調べて Excel ファイルだけを対象にする
22     '拡張子を小文字にして"|"で囲み、|xls|、|xlsx|、|xlsm|のどれかに一致するか比較する
23     If instr("|xls|xlsx|xlsm|"  _
24         ,    "|" & LCase(objFSys.GetExtensionName(objFile.Name)) & "|") Then
25         WScript.Echo objFile.Name
26     End If
27 Next
28
29 Set objFile = Nothing
30 Set objFolder = Nothing
31 Set objFSys = Nothing
```


＜選択条件の例＞

23-24 行目の選択条件は、他に以下のようなものがあります。

・ファイル更新日が今日から 30 日以上前

```
If DateDiff("d", objFile.DateLastModified, Date) >= 30 then
```

・ファイルの大きさが 0 バイト

```
If objFile.Size = 0 Then
```

＜処理の例＞

25 行目の処理は、以下のようなものがあります。

① ファイルの移動…引数はフォルダ名です。最後に"¥"を付けなかった場合はファイル名とみなしてファイル名の変更になります。以下はカレントフォルダにある backup フォルダに移動する例

```
objFile.Move  ".¥backup¥"
```

② ファイルの削除

```
objFile.Delete
```

③ ファイルの更新日時変更

FileSystemObject では更新日は変更できず、エクスプローラを制御する Shell.Application からフォルダとファイルの情報（NameSpace → ParseName →フォルダーアイテム）を使って設定します（以下のコードは元の更新日時から 30 日前を設定しています）。

この例ではフォルダの繰り返し処理に入る前に共通で使える変数の宣言とオブジェクトの取得を行うようにして、オブジェクトの生成回数が増えないようにしています。

```
'Folder オブジェクトの Files プロパティから全ての File オブジェクトを取り出して処理する
Dim objFile, objShell, objNameSpace, objFolderItem
Set objShell = CreateObject("Shell.Application")
Set objNameSpace = objShell.NameSpace(objFolder.Name)
For Each objFile In objFolder.Files
    If objFile.Size = 0 Then
        WScript.Echo objFile.Name
        Set objFolderItem = objNameSpace.ParseName(objFile.Name)
        WScript.Echo "Before :" & objFolderItem.ModifyDate
        objFolderItem.ModifyDate = DateAdd("d", -30, objFolderItem.ModifyDate)
        WScript.Echo "After  :" & objFile.DateLastModified
    End If
Next
```


2.2.2. Zip ファイルに圧縮する

以下は Windows のエクスプローラーの機能を使って zip（書庫）ファイルを作るサンプルをです。フォルダから作ることもできますが、この例では 1 ファイル毎に選別して書庫に追加しています。

【注意】マイクロソフト社のドキュメントに、「正しく処理ができない場合があり」「エクスプローラーから、ユーザー操作以外の方法で ZIP ファイルを扱うことは想定されていません。」と記されています。書庫へのコピーがスクリプトと非同期に動作するため、スクリプトが先に終わってしまうことで書庫が壊れるようです。マイクロソフト社は DotNetZip(Microsoft Public License)を使うことを推奨しています。

```
1  Option Explicit '変数の宣言(Dim)は必須
2
3  'この VBScript ファイルのフォルダ名とファイル名とを取得
4  Dim vbsNm       'この VBS のファイル名
5  Dim folderNm    'VBS の入っているフォルダ
6  vbsNm = WScript.ScriptFullName
7  folderNm = left(vbsNm, InStrRev(vbsNm, "\")-1)
8  WScript.Echo folderNm & "フォルダのファイルを処理します"
9
10 'ファイルシステムを扱うオブジェクトを作成
11 Dim objFSys
12 Set objFSys = CreateObject("Scripting.FileSystemObject")
13
14 'フォルダのオブジェクトを取得
15 Dim objFolder
16 Set objFolder = objFSys.GetFolder(folderNm)
17
18 Dim objFile, objShell, objZip, zipFNm
19 'アーカイブ先のファイルを準備
20 zipFNm = Replace(Now, "/", "")
21 zipFNm = Replace(zipFNm, ":", "")
22 zipFNm = Replace(zipFNm, " ", "-")
23 zipFNm = folderNm & "\bkup_" & zipFNm & ".zip"
24 WScript.Echo "アーカイブ to>" & zipFNm
25 'zip ファイルのガラ作成＜CreateTextFile(filename[, overwrite[, unicode]])＞
26 Set objZip = objFSys.CreateTextFile(zipFNm, True)
27 'zip ヘッダ書き込み(このヘッダが書かれることで ZIP と認識されます)
28 objZip.Write("PK" & Chr(5) & Chr(6) & String(18, 0))
29 objZip.Close
30
31 'Folder オブジェクトの Files プロパティから全ての File オブジェクトを取り出して処理する
32 Set objShell = CreateObject("Shell.Application")
33 For Each objFile In objFolder.Files
34     '拡張子を調べて Excel ファイルだけを対象にする。※先頭が"~"の作業ファイルは除外する
35     '拡張子を小文字にして"|"で囲み、|xls|か|xlsx|と一致するか比較する
36     If instr("|xls|xlsx|xlsm|"  _
37         ,    "|" & LCase(objFSys.GetExtensionName(objFile.Name)) & "|") _
38     AND Left(objFile.Name, 1) <> "~"     Then
39         'zip に追加
```



```
        WScript.Echo objFile.Path
        objShell.NameSpace(zipFNm).CopyHere(objFile.Path)
        'ファイルのコピーは非同期で行われ、1ファイル毎に完了の待ち合わせが必要です。
        'Microsoft は、DotNetZip(Microsoft Public License)を使うことを推奨しています（為念）
        WScript.Sleep 1000
    End If
Next

Set objFile = Nothing
Set objFolder = Nothing
Set objFSys = Nothing
Set objZip = Nothing
Set objShell = Nothing
```

マイクロソフト社推奨の DotNetZip[3]をインストールした環境では Ionic.Zip.ZipFile を CreateObject して利用します。以下のようになります。

```
'zip オブジェクト生成
set objZip = CreateObject("Ionic.Zip.ZipFile")
'暗号化オプション 3=AES256, 2=AES128, 1=PKZIP, 0=none
objZip.Encryption = 0
'パスワード設定
objZip.Password = "This is the Password."
'対象ファイル選択(AddFile はファイルパスで指定、AddSelectedFiles は条件選択)
objZip. AddFile objFIle.Path
objZip.AddSelectedFiles "*.xlsx"
'zip ファイルパス¥ファイル名設定・・・Save(ファイルパス)と同
objZip.Name = "<ファイルパス>¥ファイル名.zip"
'保存・・・Name を設定しなかった場合は、ここでファイルパスを指定
objZip.Save
'zip オブジェクト廃棄
objZip.Dispose
```

[3] DotNetZip http://freshmeat.sourceforge.net/projects/dotnetzip

2.2.3. 複数の Excel ファイルから集計

Ruby や Python、PHP 等、スクリプト言語の多くで Excel ファイルを扱えるライブラリの開発が行われていますが、VBScript は Excel のアプリケーションを呼び出して機能を使うことができます。

Excel の VBA マクロで行うような処理も VBScript を使って同様のことが可能です。

＜コード例＞

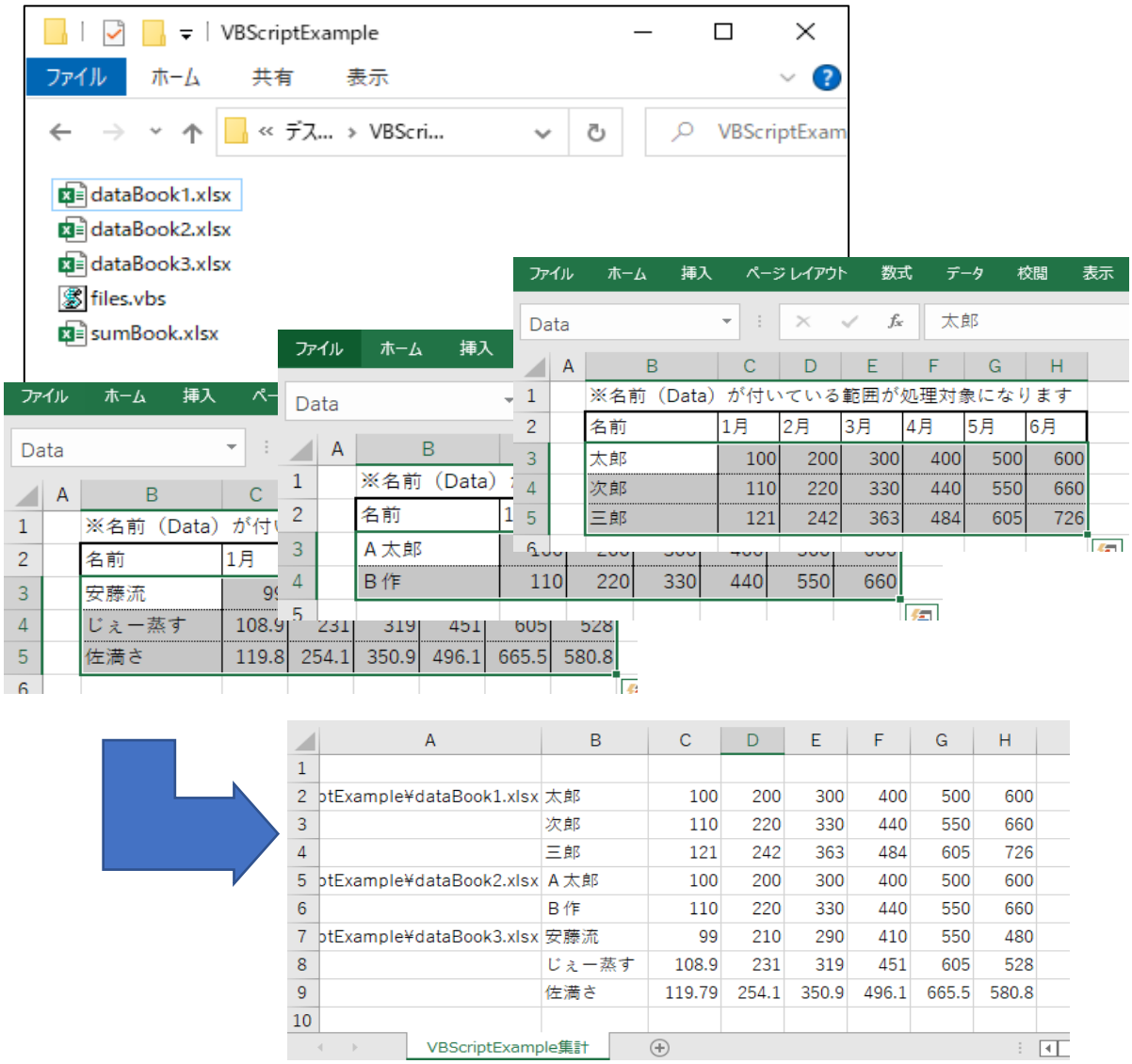
以下にあるコードを拡張子.vbs（文字コード Shift_JIS）で保存し実行すると以下のように動作します。

① 同一のフォルダにある Excel ファイルを全て参照し

② Data と名前が付けられた領域をコピー（Data という名前がなければ処理対象外）

③ 新しい Excel Book を作って①の Excel ファイルの②のデータを 1 シートに写します

…このとき、①の各 Book のファイル名をシートの左端に記録します



※実行中に vbs の処理を打ち切ると Excel のプロセスが残ってしまう場合があります。その場合はタスクマネージャから Excel のタスクを終了してください


```vbscript
Option Explicit '変数の宣言(Dim)は必須

'この VBScript ファイルのフォルダ名とファイル名とを取得
Dim vbsNm       'この VBS のファイル名
Dim folderNm    'VBS の入っているフォルダ
vbsNm = WScript.ScriptFullName
folderNm = left(vbsNm, InStrRev(vbsNm, "\")-1)
WScript.Echo folderNm & "フォルダのファイルを処理します"

'ファイルシステムを扱うオブジェクトを作成
Dim objFSys
Set objFSys = CreateObject("Scripting.FileSystemObject")

'フォルダのオブジェクトを取得
Dim objFolder
Set objFolder = objFSys.GetFolder(folderNm)

'Folder オブジェクトの Files プロパティから全ての File オブジェクトを取り出して処理する
Dim objFile, objXls, sumBook, sumSheet
Set objXls = CreateObject("Excel.Application")
objXls.Visible=False '処理中の Excel を表示する場合は True
Set sumBook = objXls.Workbooks.Add() '集計用 ExcelBook を作成
Set sumSheet = sumBook.sheets(1)
sumSheet.Name = objFolder.Name & "集計"

For Each objFile In objFolder.Files
    '拡張子を調べて Excel ファイルだけを対象にする。※先頭が"~"の作業ファイルは除外する
    '拡張子を小文字にして"|"で囲み、|xls|か|xlsx|と一致するか比較する
    If instr("|xls|xlsx|xlsm|"  _
        ,    "|" & LCase(objFSys.GetExtensionName(objFile.Name)) & "|") _
    AND Left(objFile.Name, 1) <> "~"    Then
        'WScript.Echo  TypeName(sumXls(objXls, sumSheet, objFile))
        sumXls objXls, sumSheet, objFile
    End If
Next
'Excel の SavaAs で使われるデフォルトのパスはユーザの Documents フォルダになります
sumBook.SaveAs folderNm & "\sumBook.xlsx"
sumBook.Close
objXls.Quit
Set sumSheet = Nothing
Set sumBook = Nothing
Set objXls = Nothing

Set objFile = Nothing
Set objFolder = Nothing
Set objFSys = Nothing

Function sumXls(objXls, sumSheet, anyBook)
    Const xlCellTypeLastCell = 11
```



```
    Dim lastRow
    Dim dataBook
    Dim dataRange
    WScript.Echo "Book=" & anyBook
    Set dataBook = objXls.WorkBooks.open(anyBook)

    On Error Resume Next 'エラー発生を Err.Number で感知できるようにする
    Set dataRange = dataBook.Names("data").RefersToRange
    If Err.Number <> 0 Then
        On Error Goto 0 'エラー発生時の対処を中断に戻す
        WScript.Echo "  ...×"
    Else
        On Error Goto 0 'エラー発生時の対処を中断に戻す
        '既存データの最終行の次行を求める
        lastRow = sumSheet.Cells.SpecialCells(xlCellTypeLastCell).Row + 1
        'コピー元の Book 名を2列目にセット
        sumSheet.Range("A" & lastRow) = anyBook
        '2列目からコピー元の領域と同じサイズの領域を確保してデータを代入する
        sumSheet.Range(sumSheet.Cells(lastRow , 2), _
                       sumSheet.Cells(lastRow -1 + dataRange.Rows.Count,
dataRange.Columns.Count + 1)) _
                        = dataRange.Value
        sumXls = dataRange.Value
        WScript.Echo "  ...OK "
    End If
    dataBook.Close
    Set dataRange = Nothing
    Set dataBook = Nothing
End Function
```

上記の処理は Excel ブック単位（シート単位ではなく）に付けられた data（大小文字の別なし）という名前の領域を同一のサイズで新しいワークブック／シートの最後にコピーしているだけですが、VBA と同様の方法で最終行に集計関数等を設定することも可能です。

2.2.4. WMI を使ったシステム情報の収集（イベントログ）

Windows には WMI (Windows Management Instrumentation)[4]というシステム管理の仕組みがあり、VBScript を使うと簡単に Windows システムで管理している情報が取り出せます。例えばシステムログから出力日等を条件に抽出する場合は以下のようにします。

```
'＜WMI を使ってイベントログを参照します＞
'ローカル／リモート環境どちらも接続が可能ですが、リモートの場合は管理者権限が必要です。
'_______
'WMI に接続する方法は、GetObject で WMI モニカー "winmgmts:"を指定するか、
'SWbemLocator を CreateObject するかですが、資格情報を指定するには SWbemLocator を使います

Option Explicit '変数の宣言(Dim)は必須
Dim strComputer, objSWbemLocator, objSWbemServices
strComputer = "." 'リモートの場合はコンピュータ名を指定
Set objSWbemLocator = CreateObject("WbemScripting.SWbemLocator")
Set objSWbemServices = objSWbemLocator.ConnectServer _
    (strComputer, "root\cimv2")  'リモートの場合 (… "ユーザ", "パスワード")を追加

Dim fromDate, queryStr, logs, theLog
'ログの取得開始日
fromDate = DateAdd("ww", -1, Now) '1週間前～
'イベントログ(Application、System)問合せ文字列
Const qStr1 = "Select * From Win32_NTLogEvent"
Const qStr2 = " Where (logfile='System' or Logfile='application') And TimeGenerated > '"
queryStr = qStr1 & qStr2 & fromDate & "'"
WScript.Echo queryStr
Set logs = objSWbemServices.ExecQuery(queryStr)
For Each theLog In logs
    WScript.Echo _
        "-------------------------------------------------" _
        & vbCrLf & " ComputerName=" & theLog.ComputerName & "," _
        & vbCrLf & "      Logfile=" & theLog.Logfile & "," _
        & vbCrLf & "TimeGenerated=" & theLog.TimeGenerated & "," _
        & vbCrLf & "    EventType=" & theLog.EventType & "," _
        & vbCrLf & "         Type=" & theLog.Type & "," _
        & vbCrLf & "    EventCode=" & theLog.EventCode & "," _
        & vbCrLf & "         User=" & theLog.User & "," _
        & vbCrLf & "   SourceName=" & theLog.SourceName & "," _
        & vbCrLf & "      Message=" & theLog.Message
Next

Set logs = Nothing
Set objSWbemServices = Nothing
Set objSWbemLocator = Nothing
```

[4] Microsoft ドキュメント-WMI の使用
https://docs.microsoft.com/ja-jp/windows/win32/wmisdk/using-wmi

このスクリプトを実行すると、以下のような出力が得られます。

```
  ----------------------------------------------------
   ComputerName=WIN-host,
        Logfile=System,
  TimeGenerated=20220608234335.715411-000,
      EventType=3,
           Type=情報,
      EventCode=30,
           User=,
     SourceName=Microsoft-Windows-Kernel-Boot,
        Message=ファームウェアからブート メトリックが報告されました。
  ----------------------------------------------------
   ComputerName=WIN-host,
        Logfile=System,
  TimeGenerated=20220608234334.500316-000,
      EventType=3,
           Type=情報,
      EventCode=1,
           User=,
     SourceName=Microsoft-Windows-Kernel-General,
        Message=システム時刻は　2022 - 06 - 08T07:08:03.636295700Z　から　2022 - 06 -
08T23:43:34.500000
  000Z に変更されました。

  変更の理由: System time synchronized with the hardware clock。
  プロセス: '' (PID 4)。
  ----------------------------------------------------
  （以下略）
```


3. PowerShell

PowerShell はマイクロソフト社が開発し、Windows 以外の OS 向けにもオープンソースで公開[5]されています。Ver. 5.1 までは Windows に標準搭載されているのはマイクロソフト社製の Windows PowerShell（以下、PowerShell）です。

3.1. 開発・動作環境

PowerShell Ver. 5.1 までは Windows PowerShell ISE という開発ツールが標準で添付されていますが、もはや新規機能の開発対象ではなくなっています[6]。PowerShell のスクリプトはテキストで記述するのでメモ帳等の一般のエディターで作ることができますが、マイクロソフト社は関連サイトに「PowerShell 拡張機能と共に Visual Studio Code を使用することをお勧めします。」と書いて言います。

PoweShell ISE は日本語（複数バイト文字）を扱う場合、デフォルトで Shift_JIS(CP932)になっているのでそのままだとインターネット接続で一般に使われる UTF-8 は文字化けします。また、これとは別に Windows10 以前の PowerShell コンソールはバグ[7]により文字化けが生じることがあります。こちらの対処としてはコマンドプロンプトから PowerShell を起動（コマンドプロンプトが文字化けを回避します）して開発で使う文字コードを Shift_JIS に統一してください。

また、Ver 6 以降では UTF-8(BOM 無)等に統一して OutputEncoding を設定[8]する必要がありますが、いずれにしろ日本語等の複数バイト文字を使わなければ問題は発生しません。

3.2. PowerShell の特徴／注意点

PowerShell は名前に Shell が付いているうえに、条件文の書き方やパイプ（ライン）を使った書き方、コマンドの別名に Unix/Linux（以下、Linux）と同じコマンド名を付ける等シェルに似せて作られました。しかし、データをオブジェクト／配列という単位で扱うことと、出力が標準出力／標準エラー出力だけではない等 PowerShell のパイプラインは Linux のパイプ（単純なテキストが流れる FIFO の装置）とは異なります。また、Linux のコマンドと PowerShell のコマンドレットのオプション及び結果（形式、出力方法）も異なります。今後の Vup の方向も含めて全く別言語です。

---

[5] マルチプラットフォーム／オープンソース版は PowerShell Core と呼称。動作基盤である.NET Core も一緒にオープンソースになり、どちらの Core にも Windows 版があります

[6] Windows PowerShell ISE https://docs.microsoft.com/ja-jp/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-7.2

[7] PowerShell コンソールの文字が中国語、日本語、韓国語で文字化け https://docs.microsoft.com/en-us/troubleshoot/windows-server/system-management-components/powershell-console-characters-garbled-for-cjk-languages

[8] 文字エンコーディング https://docs.microsoft.com/ja-jp/powershell/module/microsoft.powershell.core/about/about_character_encoding?view=powershell-5.1

3.3. コード例

以降のコードは VSC に PowerShell 拡張機能をインストールし、Shift_JIS で開発したものです。
また、PowerShell のバージョンは 5.1.19041.1682（$PSVersionTable を表示した結果）です。

3.3.1. 記法

#：行コメント、<# ブロックコメント #>、$：変数（代入前の宣言は必要ありません）、二重引用符（”）で囲まれた文字列に含まれる変数は展開され、単引用符（’）は展開されません。エスケープ文字はバッククオート（`）で、改行もエスケープすることができます。式の区切りはセミコロンか改行で、独立した式を同一行に続けて書きたい時はセミコロンで分かちます。

PowerShell の組み込みコマンドはコマンドレットと呼び、名称は Get、Set、Write、ForEach 等の操作に操作対象が繋がっています。スクリプトは主にこのコマンドレットをパイプラインで繋いだもので、多少のデータ加工や演算、条件分岐に文や式を使う場合があります。

3.3.2. ダイアログを使ったファイル選択

Add-Type コマンドレットを使うと.NET のクラスや dll の機能も使うことができます（7 行目）
10 行目の@{}は連想配列（ハッシュテーブル）で、“キー＝値”のペアで配列を作ります。

```
1  <# ------------------------------------------------
2   ファイル選択ダイアログを表示してファイルを選択し、
3     ・選択がなければ打ち切り
4     ・選択されたらコンソールにファイル名を出力します
5  ------------------------------------------------#>
6  # Windows の GUI（各種のフォーム／コントロール／関連情報）の諸クラスを使えるようにします
7  Add-Type -AssemblyName System.Windows.Forms
8
9  # ファイルを選択（複数可）します
10 $fileDialog = New-Object System.Windows.Forms.OpenFileDialog -Property @{
11     InitialDirectory = [Environment]::GetFolderPath('Desktop')
12     Filter = 'ログファイル (*.log,*.txt)|*.log;*.txt'
13     Title = '対象のファイルを選択してください。'
14     Multiselect = $true
15 }
16
17 # ファイル選択で OK が推されなかった場合は、Exit します（ファイルが選択されないと OK は押下できません）
18 if($fileDialog.ShowDialog() -eq [System.Windows.Forms.DialogResult]::OK){
19 }else{
20     [System.Windows.Forms.MessageBox]::Show('処理を打ち切りました')
21     Exit-PSHostProcess
22 }
23
24 # 選択された各ファイルの処理を行います
25 foreach($file in $fileDialog.FileNames){
26     Write-Host $file
27 }
28
```


3.3.3. 単純な関数

PowerShell は関数が使えます。単純な例を以下に示します。

```
<# ------------------------------------------
 ファイル選択ダイアログを表示してファイルを選択し、
   ・選択がなければ打ち切り
   ・選択されたファイル名を出力します
------------------------------------------ #>
function selectFiles ($dirName){
    If ( ($null -eq $dirName) -or !(Test-Path -Path $dirName) ){
        # パラメータに有効なパスが指定されていなかったら特別なディレクトリパスを使用する
        $dirName = [Environment]::GetFolderPath('Recent') #最近使ったフォルダを使う
    }
    # Windows の GUI(各種のフォーム／コントロール／関連情報)の諸クラスを使えるようにします
    Add-Type -AssemblyName System.Windows.Forms

    # ファイルを選択(複数可)します
    $fileDialog = New-Object System.Windows.Forms.OpenFileDialog -Property @{
        InitialDirectory = $dirName
        Filter = 'ログファイル (*.log,*.txt)|*.log;*.txt'
        Title = '対象のファイルを選択してください。'
        Multiselect = $true
    }

  #ファイル選択で OK が推されなかった場合は、Exit します(ファイルが選択されないと OK は押下できません)
    if($fileDialog.ShowDialog() -eq [System.Windows.Forms.DialogResult]::OK){
        # 選択されたファイルのパスを出力します
        return $fileDialog.FileNames
    }else{
        #[System.Windows.Forms.MessageBox]::Show('処理を打ち切りました')
        #Exit-PSHostProcess
        return
    }
}
```

上のスクリプトをカレントフォルダに PSFunctions¥SimpleSelectFiles.ps1 として保存すると、以下のようにして呼び出すことができます。

・32 行目の 1 桁目“.”は source コマンドで機能は Linux の同コマンドと同じく、実行時にこの場所に指定したファイルの内容を展開します。

・関数からの戻り値は関数内で実行した Write-Output を使った出力の全てと、return のパラメータに書いた内容です。関数がパイプラインに書かれていれば、後続のプロセスに全てが渡されます。

```
. .\PSFunctions\SimpleSelectFiles.ps1
selectFiles 'C:\Tools\apache-tomee-microprofile-8.0.4\logs'
```


3.3.4. パイプラインからパラメータを貰う関数

パイプラインに組み込んで関数を使うためには、パラメータの受け取り方と動作タイミングを知っておく必要があります。

```
<# ------------------------------------------------------------
 ファイル選択ダイアログを表示してファイルを選択し、
   ・選択がなければ打ち切り
   ・選択されたらコンソールにファイル名を出力します
   ★最初に開くディレクトリのパス（複数可）をパイプで受け取ります
------------------------------------------------------------ #>
function selectFiles (){
   [CmdletBinding()]
    Param (
        [Parameter(ValueFromPipeline)]
        [String[]]$dirName
        )
    Begin{Write-Host "function Begin"} #処理ブロックの中は空でも構いません

    Process{Write-Host "function Process"
        If ( ($null -eq $dirName) -or !(Test-Path -Path $dirName) ){
            # パラメータに有効なパスが指定されていなかったら特別なディレクトリパスを使用する
            $dirName = [Environment]::GetFolderPath('Recent') #最近使ったフォルダを使う
        }
        # Windows の GUI（各種のフォーム／コントロール／関連情報）の諸クラスを使えるようにします
        Add-Type -AssemblyName System.Windows.Forms

        # ファイルを選択（複数可）します
        $fileDialog = New-Object System.Windows.Forms.OpenFileDialog -Property @{
            InitialDirectory = $dirName
            Filter = 'ログファイル (*.log,*.txt)|*.log;*.txt'
            Title = '対象のファイルを選択してください。'
            Multiselect = $true
        }

  #ファイル選択で OK が推されなかった場合は、Exit します（ファイルが選択されないと OK は押下できません）
        if($fileDialog.ShowDialog() -eq [System.Windows.Forms.DialogResult]::OK){
            # 選択されたファイルのパスを出力します
            return $fileDialog.FileNames
        }else{
            #[System.Windows.Forms.MessageBox]::Show('処理を打ち切りました')
            #Exit-PSHostProcess
            return
        }
    }
    End{Write-Host "function End"}
}
```


（1）パラメータ

パイプラインらパラメータを受け取るためには、以下の条件を満たす必要があります。

・関数に CmdletBinding 属性を付ける

・Param ブロックを作り、Parameter 属性に ValueFromPipeline 引数を渡す

11 行目の“[String[]]$dirName”は［型］$パラメータ名で、String[]は文字列の配列を受け取れるようにしています。

スクリプトをカレントフォルダに PipeSelectFiles.ps1 として保存すると、以下のようにして呼び出すことができます。以下の例では、'C:¥Tools¥apache-tomee-microprofile-8.0.4¥logs'と'C:¥tmp'の 2 フォルダに対して selectFiles という関数の Process ブロックが実行されます。

```
. .\PSFunctions\PipeSelectFiles.ps1
'C:\Tools\apache-tomee-microprofile-8.0.4\logs','C:\tmp' | selectFiles
```

（2）実行ブロック

関数は実行内容を書いた 3 つのブロックを書きます。

① Begin :関数の開始時に 1 度だけ実行します

② Process :パイプラインで受け取るオブジェクトの数だけ実行します

③ End :関数の最後に 1 度だけ実行します

※以下の関数を作って（dummy.ps1 ファイル）パイプで繋いだ結果で動作順番を確認します

```
function funcDummy (){
    [CmdletBinding()]
    Param (
      [Parameter(ValueFromPipeline)]
        [String[]]$para
        )
    Begin{Write-Host "function-D Begin"}
    Process{
        Write-Host "function-D Process"
        foreach ($p in $para){
            Write-Host "function-D para>",$p
            Write-Output $p
        }
    }
    End{Write-Host "function-D End"}
}
```

パイプを使って n 個のファイルパスがパラメータとして渡されます

但し、パイプに渡されるのは Write-Output（または return の引数）だけで Write-Host はコンソール／ターミナルにだけ出力されます

```
. .\PSFunctions\PipeSelectFiles.ps1
. .\PSFunctions\dummy.ps1
'C:\Tools\apache-tomee-8.0.4-microprofile\logs','C:\tmp' |
selectFiles | funcDummy | ForEach-Object{Write-Host "pipe-end" + $_}
```

これで、関数 selectFiles からパスの配列が出力され、次に funcDummy 関数が呼び出されます。

パイプラインの最後の ForEach-Object は、配列の 1 要素が$_に格納されて参照することができるようになり全ての要素に対して {} の中の処理を繰り返します

＜結果出力＞

前掲のスクリプトを実行するとファイル選択ダイアログが 2 回表示され、最初のダイアログで 2 ファイルを次に 1 ファイルを選択しました。結果として以下の内容がターミナルに出力されます。

```
1   function Begin
2   function-D Begin
3   function Process
4   function-D Process
5   function-D para> ＜ダイアログー１のファイルパス-1＞.txt
6   pipe-end + ＜ダイアログー１のファイルパス略-1＞.txt
7   function-D Process
8   function-D para> ＜ダイアログー１のファイルパス-2＞.txt
9   pipe-end + ＜ダイアログー１のファイルパス-2＞.txt
10  function Process
11  function-D Process
12  function-D para> ＜ダイアログー２のファイルパス＞.log
13  pipe-end + ＜ダイアログー２のファイルパス＞.log
14  function End
15  function-D End
```

行番号 1 ～ 9 が 1 回目のファイル選択ダイアログで、selectFiles(以下 F1)と funcDummy(以下 F2)の各ブロックは以下の順で実行されます

F1-Begin → F2-Begin → F1-Process → F2-Process → ForEach[]

return [path1,path2] ↓ ↑ para=path1

→ F2-Process → ForEach[]

↑ para=path2

＜以下、パイプライン 2 巡目＞行番号 10～

→ F1-Process → F2-Process → ForEach[]

＜処理対象 0＞

F1-End → F2-End

【パイプライン／関数を使うポイント】

パイプラインに組み込まれた関数やコマンドレットの出力が配列の場合は要素毎にバラされて後続プロセスにデータが渡され、配列の数だけパイプラインが分岐していきます。したがって、全体の処理を機能単位で細かく分割してパイプラインで繋ぎ、可能になったらすぐに出力して後続処理に渡すことで処理性能の向上やメモリの使用量削減が期待できます。

パイプラインは変数への代入で置き換えることが可能（F1｜F2 ⇒ $ver1=F1; $ver2=F2($ver1)）ですが、これをすると上記と逆に処理効率が低下しメモリ使用量は増大します。大きなファイル全体をメモリに滞留さる等のようなことが無いように注意が必要です。

3.3.5. http ステータスがエラーのログの抽出

Java のシステム、特に Web システムでは GC による長時間化やシステムエラーが人知れず発生している場合があります。以下に特定の日時の範囲から http ステータスが成功（2xx）以外のアクセスログを抜き出すスクリプトの例を記載します。

```
# 関数の取り込み
. .\PSFunctions\PipeSelectFiles.ps1
. .\PSFunctions\dummy.ps1
'C:\Tools\apache-tomee-8.0.4-microprofile\logs','C:\tmp' | selectFiles |
  #① 選択されたファイル毎のループ（http ステータスが 2xx 以外を抽出）
  ForEach-Object { Write-Host '>>>>>>>>' $_
   Select-String -Encoding UTF8 -Pattern '\[(?<times>.+)\].+\"(?<url>.*)\"\s(?<status>[^2]\d{2})\s(\d+|-)$' $_
  } |
    #② マッチしたレコード毎のループ
    ForEach-Object { $_.Matches[0]
    } |
      #③ マッチした文字列グループの処理（①でマッチした文字列は後参照の指定「()」があると Matches[0].Goups[]に納
      ForEach-Object {
        $utc = [DateTime]::ParseExact($_.Groups['times'].Value, "dd/MMM/yyyy:HH:mm:ss zzz", [cultureinfo]'en')
        #使われていた日時が英語表現の日本時間だったため、utc->jst 変換（時差調整）せず出力する
        $utc.ToString("yyyy/MM/dd_HH:mm:ss")+' '+$_.Groups['status'].Value+' '+$_.Groups['url'].Value
      } |
        #④ 取り出すログの出力時間を範囲指定（改行は許してないので、行末のバッククォート[`]で改行コードをエスケープ
        Where-Object {($_.Substring(0, 19) -ge '2021/03/17_00:00:00')  `
                  -and ($_.Substring(0, 19) -le '2021/03/18_24:00:00')
        } > 'C:\tmp\unsuccessful.dat'
```

＜アクセスログ＞　※空白区切りで、後ろから 2 項目目の 200 や 500、302 が http ステータスです

```
0:0:0:0:0:0:0:1 - - [18/Mar/2021:12:02:27 +0900] "OPTIONS /React/login HTTP/1.1" 200 -
0:0:0:0:0:0:0:1 - - [18/Mar/2021:12:02:27 +0900] "POST /React/login HTTP/1.1" 200 126
0:0:0:0:0:0:0:1 - - [18/Mar/2021:12:02:30 +0900] "GET /React/AccGetTantouName.jsp?sno=1010 HTTP/1.1" 500 6809
0:0:0:0:0:0:0:1 - - [18/Mar/2021:12:02:31 +0900] "GET /React/AccGetKamokuName.jsp?kcd=210100 HTTP/1.1" 500 6833
127.0.0.1 - - [18/Mar/2021:12:14:32 +0900] "GET / HTTP/1.1" 200 11447
0:0:0:0:0:0:0:1 - - [19/Mar/2021:10:28:38 +0900] "GET /React HTTP/1.1" 302 -
```

① 正規表現 '¥[(?<times>.+)¥].+¥"(?<url>.*)¥"¥s(?<status>[^2]¥d{2})¥s(¥d+|-)$' の各意味は、

ⅰ. ¥[(?<times>.+)¥]…[ ]で囲まれた部分を切り出して名前 times を付けます。"¥"はエスケープ

ⅱ. ¥"(?<url>.*)¥"　…” ”で囲まれた部分を切り出して名前 url を付けます

ⅲ. (?<status>[^2]¥d{2}) …数字の 2 以外と後ろに数字 2 桁を切り出して名前 status を付けます

その他… ⅰ、 ⅱ、 ⅲ以外の部分 .+ は任意の文字 1 桁以上、¥s は空白 1 桁、

(¥d+|-)は数字の 1 桁以上または - 、$は行末（改行）

※“￥”と”\”は表示フォントを変えた同一の文字です

上記に合致するレコードがあると、Matches[0].Group[*Name*]に上記の ⅰ、 ⅱ、 ⅲの値が入ります

前掲のコードの 13 行目に以下のコード（下線部）を追加するとターミナルで内容が確認できます。

```
13          ForEach-Object { Write-Host ($_.Groups | Out-String)
```

＜ターミナル出力＞

```
  Groups   : {0, 1, times, url...}
  Success  : True
  Name     : 0
  Captures : {0}
  Index    : 20
  Length   : 89
  Value    : [18/Mar/2021:12:02:30 +0900] "GET /React/AccGetTantouName.jsp?sno=1010 HTTP/1.1" 500 6809

  Success  : True
  Name     : times
  Captures : {times}
  Index    : 21
  Length   : 26
  Value    : 18/Mar/2021:12:02:30 +0900

  Success  : True
  Name     : url
  Captures : {url}
  Index    : 50
  Length   : 49
  Value    : GET /React/AccGetTantouName.jsp?sno=1010 HTTP/1.1

  Success  : True
  Name     : status
  Captures : {status}
  Index    : 101
  Length   : 3
  Value    : 500
```

② 文字列を探した結果として複数の結果が得られることが考えられます。例えば、"abc123xyz123"から"123"や¥d+で探すと 2 カ所が抽出されます。このため Matches と Groups は配列になっていますが、今回の例ではあっても 1 か所／1 行を前提にしているので、Matches[0]から情報を得ます。

③ Matchs[0]の Groups から出力用に編集します。ここはループの必要はなくパイプラインの中に処理を書くためだけに ForEach-Object ブロックを使っています

④ 出力編集後の日時を使って出力対象の範囲を限定し、結果をファイル('C:¥tmp¥unsuccessful.dat')にリダイレクト（21 行目）します

＜出力されたファイルの抜粋＞ *:http ステータス

```
<---タイムスタンプ---> <*> <--------------------URL--------------------------------------
2021/03/19_14:22:33 302 GET /React HTTP/1.1
2021/03/18_12:02:30 500 GET /React/AccGetTantouName.jsp?sno=1010 HTTP/1.1
2021/03/18_12:02:31 500 GET /React/AccGetKamokuName.jsp?kcd=210100 HTTP/1.1
2021/03/18_12:02:31 500 GET /React/AccGetKamokuName.jsp?kcd=210100 HTTP/1.1
2021/03/17_09:03:11 500 GET /React/AccGetTantouName.jsp?sno=1010 HTTP/1.1
```


3.3.6. 入力ファイル（CSV）処理、ネットワークの生存確認、Json 編集

PowerShell を使うと CSV や Json 等のデータ交換用のフォーマットを扱ったりネットワークへのアクセスが容易です。今回挙げる例はパイプラインだけでできる状態確認のみですが、他にも Http リクエストを出す（Invoke-WebRequest/Invoke-RestMethod）等の多種の機能があります。

```
<# ------------------------------------------------------
 CSV ファイルから調査対象のサーバとポート番号を受け取り、
   以下の情報を収集します。
   ・ping が通るか
   ・指定されたポートでサービスが開始されているか
------------------------------------------------------------#>
Import-CSV .\ServersPort.txt |   #調査対処 CSV ファイル（1 行目は"server,port"でなきゃダメ）
    ForEach-Object { [PSCustomObject]@{
          ComputerName = $_.server;
          Reachable = (Test-Connection $_.server -quiet -Count 1) #ping
          ServiceIn = (          #収集情報はポートの活性確認のみ
                      Test-NetConnection $_.server `
                      -Port $_.port `
                      -InformationLevel Quiet)
        }
     } | ConvertTo-Json -depth 1 #Json 形式に変換（保存が必要ならファイルにリダイレクトする）
```

＜ServerPort.txt＞

```
server,port
ubuntu2004,22
localhost,80
```

＜ターミナル出力＞

```
警告: TCP connect to (::1 : 80) failed
警告: TCP connect to (127.0.0.1 : 80) failed
[
   {
      "ComputerName":  "ubuntu2004",
      "Reachable":  true,
      "ServiceIn":  true
   },
   {
      "ComputerName":  "localhost",
      "Reachable":  true,
      "ServiceIn":  false
   }
]
```

※localhost ではポート番号 80 を開いていないので、警告がでています

3.3.7. タスクスケジューラへの登録

PowerShell の適用分野は色々あります。セキュリティ対策のレジストリ書き換えに使うだけでなく、bat ファイルでは制約が多いホスト間のバックアップの取得にコマンドレットは便利です。

しかし、PowerSell のスクリプトをタスクスケジューラに登録してトリガーの条件を満たしてもうまく動作しないことがあります。よくある原因は以下のものです。

① スクリプト（ps1 ファイル）を開くアプリがメモ帳になっている（ログにエラーも出ません）
② 起動するホストにセキュリティポリシーが設定されていない

＜操作の登録例＞

```
powershell.exe -NoProfile -NoLogo -NonInteractive -ExecutionPolicy Bypass -File 実行スクリプト.ps1
```

---オプションの内容--
-NoProfile …起動時に初期化ファイル（$pforile）を読み込みません
-NoLogo …PowerShell の起動メッセージを抑止します（好みの問題）
-NonInteractive …ユーザの応答が必要になった場合は打ち切ります
-ExecutionPolicy Bypass …このスクリプトはブロックされず、警告やプロンプトは表示されません
-File…実行したいスクリプトファイルを指定します

以上

