指定ディレクトリにあるプログラム群を並列実行するコマンド

2016年06月22日
バッチファイル名に " & " が含まれていると実行ができなかったのを修正。

※このプログラムは .NET Framework 4.5.2の環境下で動作を確認しています。

練習がてらここのソースをGitHubにもあげてみました。

https://github.com/k896951/Parallelrun

よくわからなくて泣きそうになった。

PARARUN.EXE ダウンロード

pararun.zip 2016-06-22版　.NET Framework 4 以降で実行　.NET Framework 4.5.2で実行できることを確認。

PARARUNコマンドの簡単な説明

指定のディレクトリにある *.bat　*.cmd　*.exe をスレッドで並列実行するコマンドです。
指定ディレクトリにある実行ファイルのパスはキュ　(FIFO)に格納され、指定数生成された各スレッドがこのキューから実行ファイルのパスを取り出して実行していきます。
各スレッドがキューから実行ファイルのパスを読み出せなくなったら終了となります。

```
H:\JOB>pararun
pararun [-ut] [-nr] -qs count folder [folder ...] [-qf count folder [folder
...]] [-qh count folder [folder ...]]

    -ut : Use "Task Class"
    -nr : Not reuse free threads.
    -qs : Use "s" queue.
    -qf : Use "f" queue.
    -qh : Use "h" queue.
  count : Number of the used thread.
 folder : Batch job stock folder.

H:\JOB>
```

以下は実行例です。 ディレクトリ H:\JOBにPARARUN.EXEがあり　H:\JOB\job1　H:\JOB\job2　H:\JOB\job3　のディレクトリがあります。

```
H:\JOB>pararun -qs 2 .\job1 -qf 2 .\job2 -qh 1 .\job3
```

```
2016/06/22 5:30:32,----, Thread reuse mode: True
2016/06/22 5:30:32,----, Use Task Class: False
2016/06/22 5:30:32,----, s queue is use 2 threads
2016/06/22 5:30:32,----, f queue is use 2 threads
2016/06/22 5:30:32,----, h queue is use 1 threads
2016/06/22 5:30:32,----, total 5 threads use, 9 jobs enqueued.
2016/06/22 5:30:32,----, Start    threads
2016/06/22 5:30:32,  s0, Start    .\job1\a1.bat
2016/06/22 5:30:32,  s1, Start    .\job1\b1.bat
2016/06/22 5:30:32,  f1, Start    .\job2\a2.bat
2016/06/22 5:30:32,  h0, Start    .\job3\a3.bat
2016/06/22 5:30:32,  f0, Start    .\job2\b2.bat
2016/06/22 5:30:35,  f0, Finish   .\job2\b2.bat, 00:00:03.0739841, rcd=13
2016/06/22 5:30:35,  f0, Start    .\job2\c2.bat
2016/06/22 5:30:35,  f1, Finish   .\job2\a2.bat, 00:00:03.0994884, rcd=0
2016/06/22 5:30:35,  f1, Start    .\job1\c1.bat
2016/06/22 5:30:37,  s0, Finish   .\job1\a1.bat, 00:00:05.1208972, rcd=0
2016/06/22 5:30:37,  s0, Start    .\job3\b3.bat
2016/06/22 5:30:38,  f0, Finish   .\job2\c2.bat, 00:00:03.0000954, rcd=0
2016/06/22 5:30:38,  f0, Start    .\job3\c3.bat
2016/06/22 5:30:40,  f1, Finish   .\job1\c1.bat, 00:00:04.9450942, rcd=0
2016/06/22 5:30:40,  f1, Queue is empty.
2016/06/22 5:31:02,  s1, Finish   .\job1\b1.bat, 00:00:30.1207724, rcd=0
2016/06/22 5:31:02,  s1, Queue is empty.
2016/06/22 5:31:22,  h0, Finish   .\job3\a3.bat, 00:00:50.0874377, rcd=0
2016/06/22 5:31:22,  h0, Queue is empty.
2016/06/22 5:31:33,  f0, Finish   .\job3\c3.bat, 00:00:55.0596714, rcd=0
2016/06/22 5:31:33,  f0, Queue is empty.
2016/06/22 5:35:37,  s0, LongRun  .\job3\b3.bat
2016/06/22 5:36:07,  s0, Finish   .\job3\b3.bat, 00:05:29.9256878, rcd=0
2016/06/22 5:36:07,  s0, Queue is empty.
2016/06/22 5:36:07,----, End      threads, 00:05:35.1250946
2016/06/22 5:36:07,----, Retcode    0 :     8 jobs.
2016/06/22 5:36:07,----, Retcode   13 :     1 jobs.
H:\JOB>
```

時刻はローカルタイムです。タイムゾーンが日本に設定されているOSで実行したのであれば、2016/06/22 5:30:32 は素直に日本時間の 2016/06/22 5:30:32 になります。
s0□1がsキュ□□f0□1がfキュ□□h0がhキューの処理に使われているスレッドのスレッド名です。
各スレッドで実行中の処理時間が5分以上かかっている場合□"LongRun"のメッセージが出ます。
処理終了時のリターンコードを rcd=xx の形でログに出力しています。

全部のスレッドで " Queue is empty."のメッセージが出ると全ての実行ファイルが実行されたことになります。

単一キューでの実行

sキューはデフォルトのキューで必ず使用されます。オプションが全く指定されていない場合□sキューに実行ファイルのパスが格納され4スレッドで実行が行われます。

例えば以下の指定

```
pararun .\job1 .\job2 .\job3
```

は、

- ディレクトリ .\job1、.\job2、.\job3 にある *.CMD、*.BAT、*.EXE を sキュー に格納
- sキューの実行ファイルを4並列(4スレッド)で実行

を意味します。指定するディレクトリ名の最後に " ¥ " をつけないでください。

明示的にsキュー処理用のスレッド数を指定する場合、例えば、

```
pararun -qs 6 .\job1 .\job2 .\job3
```

の指定は、

- ディレクトリ .\job1、.\job2、.\job2 にある *.CMD、*.BAT、*.EXE を sキュー に格納
- sキューの実行ファイルを6並列(6スレッド)で実行

を意味します。

複数キューでの実行

キューは3つまで利用できます。例えば、

```
pararun .\job2 .\job1 -qf 3 .\job3 -qh 1 .\job4
```

の指定は、

- ディレクトリ .\job2、.\job1 にある *.CMD、*.BAT、*.EXE を sキュー に格納
- ディレクトリ .\job3 にある *.CMD、*.BAT、*.EXE を fキュー に格納
- ディレクトリ .\job4 にある *.CMD、*.BAT、*.EXE を hキュー に格納
- sキューを4並列(4スレッド)で実行
- fキューを3並列(3スレッド)で実行
- hキューを1並列(1スレッド)で実行

を意味しますし、以下の指定、

```
pararun -qs 8 .\job2 .\job1 -qf 3 .\job3 -qh 1 .\job4
```

の場合は、

- ディレクトリ .\job2、.\job1 にある *.CMD、*.BAT、*.EXE を sキュー に格納
- ディレクトリ .\job3 にある *.CMD、*.BAT、*.EXE を fキュー に格納
- ディレクトリ .\job4 にある *.CMD、*.BAT、*.EXE を hキュー に格納
- sキューを8並列(8スレッド)で実行
- fキューを3並列(3スレッド)で実行
- hキューを1並列(1スレッド)で実行

を意味します。また各キューで使ったスレッドはすぐに破棄せず、他のキューの処理に使用されます。

- sキューの内容が全て実行されたのち、
  - まだfキューに処理されていない実行ファイルがあるとsキューで使っていたスレッドをfキューの処理に使用します。
  - まだhキューに処理されていない実行ファイルがあるとsキューで使っていたスレッドをhキューの処理に使用します。
- fキューの内容が全て実行されたのち、
  - まだsキューに処理されていない実行ファイルがあるとfキューで使っていたスレッドをsキューの処理に使用します。
  - まだhキューに処理されていない実行ファイルがあるとfキューで使っていたスレッドをhキューの処理に使用します。
- hキューの内容が全て実行されたのち、
  - まだsキューに処理されていない実行ファイルがあるとhキューで使っていたスレッドをsキューの処理に使用します。
  - まだfキューに処理されていない実行ファイルがあるとhキューで使っていたスレッドをfキューの処理に使用します。

スレッドの再使用をさせたくない場合はオプション -nr を指定してください。

ParallelクラスとTaskクラス

Actionクラスで定義したラムダ式の処理を、デフォルトではParallel.Invoke()で実行します。-ut オプションを指定すると個々のスレッド処理をTask.Start()で起動しTask.WaitAll()で終了待ちします。

ソース

表示等が気に入らない場合はソースを修正し再コンパイルを行ってください。

とりあえずのコンパイルコマンドラインをコピペしておきます。.NET Frameworkのバージョンによってcsc.exeのパスが違うと思うので適宜書き換えを。

make.bat

```
@echo off
set CMD=C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe
set OPT1=/noconfig /nowarn:1701,1702,2008 /nostdlib+ /platform:anycpu
/warn:4 /filealign:512 /optimize+ /target:exe /utf8output
set
OPT2=/reference:"C:\Windows\Microsoft.NET\Framework\v4.0.30319\Microsoft.CSharp.dll"
/reference:"C:\Windows\Microsoft.NET\Framework\v4.0.30319\mscorlib.dll"
/reference:"C:\Windows\Microsoft.NET\Framework\v4.0.30319\System.Core.dll"
/reference:"C:\Windows\Microsoft.NET\Framework\v4.0.30319\System.Data.dll"
/reference:"C:\Windows\Microsoft.NET\Framework\v4.0.30319\System.Data.DataSetExtensions.dll"
/reference:"C:\Windows\Microsoft.NET\Framework\v4.0.30319\System.dll"
```

```
%CMD% %OPT1% %OPT2% /out:pararun.exe  Pararun.cs
```

以下はPARARUN.EXEのソースコードです。

Pararun.cs

```
// Pararun
//
// Copyright (C) 2015,2016 Hideki Gotoh ( k896951 )
//
// This software is released under the MIT License.
// http://opensource.org/licenses/mit-license.php
//

using System;
using System.Collections.Generic;
using System.Collections;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics;
using System.Text.RegularExpressions;
using System.Linq;

namespace pararun
{
    class Program
    {
        static int longexecspan = 5 * 60 * 1000;

        static void Main(string[] args)
        {
            DateTime pSt = DateTime.Now;
            DateTime pEt;
            Action[] ta;
            Task[] tary = null;

            jobCollector cj = new jobCollector(args);

            if (0 == cj.getJobsTotal)
            {
                help();
                Environment.Exit(0);
            }

            ta = new Action[cj.getThreadsTotal];
            if( true == cj.getUserTaskClassFlag)
            {
                tary = new Task[cj.getThreadsTotal];
            }
```

```
            Console.WriteLine("{0},{1,4}, Thread reuse mode: {2}",
DateTime.Now, "----", cj.getReuseThreadFlag);
            Console.WriteLine("{0},{1,4}, Use Task Class: {2}",
DateTime.Now, "----", cj.getUserTaskClassFlag);

            Console.WriteLine("{0},{1,4}, s queue is use {2} threads",
DateTime.Now, "----", cj.getSlimCount);
            Console.WriteLine("{0},{1,4}, f queue is use {2} threads",
DateTime.Now, "----", cj.getFatCount);
            Console.WriteLine("{0},{1,4}, h queue is use {2} threads",
DateTime.Now, "----", cj.getHeavyCount);
            Console.WriteLine("{0},{1,4}, total {2} threads use, {3}
jobs enqueued.", DateTime.Now, "----", cj.getThreadsTotal,
cj.getJobsTotal);

            if (true == cj.getUserTaskClassFlag)
            {
                Console.WriteLine("{0},{1,4}, Start    threads",
DateTime.Now, "----");
            }

            for (int i = cj.getThreadsTotal - 1; i > -1; i--)
            {
                int ii = i;

                jobCollector.useQueue uq = cj.threadIdxToQueue(ii);

                ta[ii] = new Action(() =>
                {
                    Process pa = new Process();

                    while (true)
                    {
                        try
                        {
                            DateTime et;
                            DateTime st;
                            String cmdline = cj.dequeue(uq);

                            if (null == cmdline)
                            {
                                Console.WriteLine("{0},{1,4}, Queue is
empty.", DateTime.Now, cj.getThreadName(ii));
                                break;
                            }
                            pa.StartInfo.FileName = "CMD.EXE";
                            pa.StartInfo.Arguments = "/C;" + "\"" +
cmdline + "\"";
                            pa.StartInfo.UseShellExecute = false;
                            pa.StartInfo.CreateNoWindow = true;
                            pa.StartInfo.ErrorDialog = false;
```

```
                            st = DateTime.Now;
                            Console.WriteLine("{0},{1,4}, Start
{2}", st, cj.getThreadName(ii), cmdline);
                            pa.Start();

                            pa.WaitForExit(longexecspan);
                            if (false == pa.HasExited)
                            {
                                Console.WriteLine("{0},{1,4}, LongRun
{2}", DateTime.Now, cj.getThreadName(ii), cmdline);
                                pa.WaitForExit();
                            }
                            et = DateTime.Now;
                            cj.setRetCodeCount(pa.ExitCode);

                            Console.WriteLine("{0},{1,4}, Finish   {2},
{3}, rcd={4}", et, cj.getThreadName(ii), cmdline, et - st,
pa.ExitCode);
                        }
                        catch(Exception e)
                        {
                            Console.WriteLine("{0},{1,4}, Error
{2}", DateTime.Now, cj.getThreadName(ii), e.Message + " " +
e.StackTrace);
                            break;
                        }
                    }

                    pa.Close();
                });

                if (true == cj.getUserTaskClassFlag)
                {
                    tary[i] = new Task(ta[i]);
                    tary[i].Start();
                }

            }

            if (true == cj.getUserTaskClassFlag)
            {
                Task.WaitAll(tary);
            }
            else
            {
                Console.WriteLine("{0},{1,4}, Start    threads",
DateTime.Now, "----");
                Parallel.Invoke(ta);
            }
```

```
            pEt = DateTime.Now;
            Console.WriteLine("{0},{1,4}, End      threads, {2}", pEt,
"----", pEt - pSt);

            SortedDictionary<int, int> retlist = cj.getRetcdDic;
            if (retlist.Count!=0)
            {
                foreach(int retcd in retlist.Keys)
                {
                    Console.WriteLine("{0},{1,4}, Retcode {2,4} : {3,5}
jobs.", pEt, "----", retcd, retlist[retcd]);
                }
            }

        }

        static public void help()
        {
            Console.WriteLine("pararun [-ut] [-nr] -qs count folder
[folder ...] [-qf count folder [folder ...]] [-qh count folder [folder
...]]");
            Console.WriteLine("");
            Console.WriteLine("    -ut : Use \"Task Class\"");
            Console.WriteLine("    -nr : Not reuse free threads.");
            Console.WriteLine("    -qs : Use \"s\" queue.");
            Console.WriteLine("    -qf : Use \"f\" queue.");
            Console.WriteLine("    -qh : Use \"h\" queue.");
            Console.WriteLine("  count : Number of the used thread.");
            Console.WriteLine(" folder : Batch job stock folder.");
        }
    }

    class jobCollector
    {
        Regex regJobs =  new
Regex(@".*\.(:?[Cc][Mm][Dd]|[Bb][Aa][Tt]|[Ee][Xx][Ee])$");
        Regex regOThreads = new Regex(@"\d+");

        Int32 threadCountSlim  = 4;
        Int32 threadCountFat   = 0;
        Int32 threadCountHeavy = 0;
        Boolean notReuseThreadFlas = false;
        Boolean useTaskClassFlag = false;

        Queue jobQueueSlim;
        Queue jobQueueFat;
        Queue jobQueueHeavy;
        Queue[] jqAry;

        SortedDictionary<int, int> retcodeCollecter = new
SortedDictionary<int, int>();
```

```
        public enum useQueue
        {
            slim,
            fat,
            heavy
        }

        public Int32 getSlimCount
        {
            get
            {
                return threadCountSlim;
            }
        }
        public Int32 getFatCount
        {
            get
            {
                return threadCountFat;
            }
        }
        public Int32 getHeavyCount
        {
            get
            {
                return threadCountHeavy;
            }
        }

        public jobCollector(String[] paramStrs)
        {
            jobQueueSlim  = Queue.Synchronized(new Queue());
            jobQueueFat   = Queue.Synchronized(new Queue());
            jobQueueHeavy = Queue.Synchronized(new Queue());

            jqAry  = new Queue[] { jobQueueSlim,  jobQueueFat,
jobQueueHeavy,
                                   jobQueueFat,   jobQueueSlim,
jobQueueHeavy,
                                   jobQueueHeavy, jobQueueSlim,
jobQueueFat   };

            Int32 pLen = paramStrs.Length;
            Queue refQueue = jobQueueSlim;

            for ( Int32 idx=0; idx< pLen; idx++)
            {
                switch (paramStrs[idx])
                {
                    case @"-ut":
```

```
                        useTaskClassFlag = true;
                        continue;

                    case @"-nr":
                        notReuseThreadFlas = true;
                        continue;

                    case @"-qs":
                        if (idx < (pLen - 1))
                        {
                            threadCountSlim = setThreads(paramStrs[idx
+ 1], threadCountSlim);
                            refQueue = jobQueueSlim;
                            idx++;
                            continue;
                        }
                        break;

                    case @"-qf":
                        if (idx < (pLen - 1))
                        {
                            threadCountFat = setThreads(paramStrs[idx +
1], threadCountFat);
                            if (0!= threadCountFat) refQueue =
jobQueueFat;
                            idx++;
                            continue;
                        }
                        break;

                    case @"-qh":
                        if (idx < (pLen - 1))
                        {
                            threadCountHeavy = setThreads(paramStrs[idx
+ 1], threadCountHeavy);
                            if (0 != threadCountHeavy) refQueue =
jobQueueHeavy;
                            idx++;
                            continue;
                        }
                        break;
                }

                try
                {
                    foreach (String item in
Directory.GetFiles(paramStrs[idx], "*",
SearchOption.AllDirectories).Where(f => regJobs.IsMatch(f)).ToArray())
                    {
                        refQueue.Enqueue(item);
                    }
```

```
                }
                catch(Exception)
                {
                    //Console.WriteLine(e.Message);
                }
            }
        }

        private Int32 setThreads(String s, Int32 tn)
        {
            Int32 ans = tn;

            if (regOThreads.IsMatch(s))
            {
                ans = Int32.Parse(s);
                if (0 >= ans) ans = tn;
            }
            return ans;
        }

        public String getThreadName(Int32 tNum)
        {
            String ans = "";

            if ((threadCountHeavy != 0) && (tNum >= (threadCountSlim +
threadCountFat)))
            {
                ans = String.Format("h{0}", tNum - (threadCountSlim +
threadCountFat));
            }
            else if ((threadCountFat != 0) && (tNum >=
threadCountSlim))
            {
                ans = String.Format("f{0}", tNum - threadCountSlim);
            }
            else
            {
                ans = String.Format("s{0}", tNum);
            }
            return ans;
        }
        public Int32 getThreadsTotal
        {
            get
            {
                return threadCountSlim + threadCountFat +
threadCountHeavy;
            }
        }
        public Int32 getJobsTotal
        {
```

```
            get
            {
                return jobQueueSlim.Count + jobQueueFat.Count +
jobQueueHeavy.Count;
            }
        }
        public Boolean getReuseThreadFlag
        {
            get
            {
                return !notReuseThreadFlas;
            }
        }
        public Boolean getUserTaskClassFlag
        {
            get
            {
                return useTaskClassFlag;
            }
        }
        public SortedDictionary<int,int> getRetcdDic
        {
            get
            {
                return retcodeCollecter;
            }
        }

        public useQueue threadIdxToQueue(Int32 idx)
        {
            useQueue ans = useQueue.slim;

            if ((0 != threadCountFat) && (0 != threadCountHeavy))
            {
                if ((threadCountSlim <= idx) && (idx < (threadCountSlim
+ threadCountFat)))
                    ans = useQueue.fat;
                else if ((threadCountSlim + threadCountFat) <= idx)
                    ans = useQueue.heavy;
            }
            if ((0 != threadCountFat) && (0 == threadCountHeavy))
            {
                if (threadCountSlim <= idx)
                    ans = useQueue.fat;
            }
            if ((0 == threadCountFat) && (0 != threadCountHeavy))
            {
                if (threadCountSlim <= idx)
                    ans = useQueue.heavy;
            }
```

```
            return ans;
        }
        public String dequeue(useQueue q)
        {
            String ansStr = null;
            Int32 jqAryIdx = 0;

            switch(notReuseThreadFlas)
            {
                case false:
                    switch (q)
                    {
                        default:
                        case useQueue.slim:
                            jqAryIdx = 0;
                            break;

                        case useQueue.fat:
                            jqAryIdx = 3;
                            break;

                        case useQueue.heavy:
                            jqAryIdx = 6;
                            break;
                    }

                    lock(jqAry)
                    {
                        if (0 != jqAry[jqAryIdx + 0].Count)
                            ansStr = jqAry[jqAryIdx + 0].Dequeue() as
String;
                        else if (0 != jqAry[jqAryIdx + 1].Count)
                            ansStr = jqAry[jqAryIdx + 1].Dequeue() as
String;
                        else if (0 != jqAry[jqAryIdx + 2].Count)
                            ansStr = jqAry[jqAryIdx + 2].Dequeue() as
String;
                    }

                    break;

                case true:
                    try
                    {
                        switch (q)
                        {
                            default:
                            case useQueue.slim:
                                ansStr = jobQueueSlim.Dequeue() as
String;
                                break;
```

```
                            case useQueue.fat:
                                ansStr = jobQueueFat.Dequeue() as
String;
                                break;

                            case useQueue.heavy:
                                ansStr = jobQueueHeavy.Dequeue() as
String;
                                break;
                        }
                    }
                    catch(Exception)
                    {
                        ansStr = null;
                    }
                    break;
                }
                return ansStr;
            }
            public void setRetCodeCount(int rcd)
            {
                lock(retcodeCollecter)
                {
                    if (retcodeCollecter.ContainsKey(rcd))
                    {
                        retcodeCollecter[rcd] += 1;
                    }
                    else
                    {
                        retcodeCollecter[rcd] = 1;
                    }
                }
            }
        }
}
```

Windows, batch, tool, Csharp