Java 並行処理

[Java]

非同期に grep を実施する例 1

内容

・Thread ではなく、Executor を使って、Runnable を非同期に実行する
・共有データにスレッドセーフな単一の変数 (java.util.concurrent.atomic) を使用する

■ java.util.concurrent.atomic

・単一の変数に対するロックフリーでスレッドセーフなプログラミングをサポートするクラスの小規模なツールキット

■ Executor

・java.util.concurrent
・送信された Runnable タスクを実行するオブジェクト
・通常、executor は、明示的にスレッドを作成する代わりに使用

■ソース

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Executor;
import java.util.concurrent.atomic.AtomicInteger;

/**
 *
 */
public class JGrep {

    // 同期された変数
    //volatile int workingThreadCount;
    static AtomicInteger workingThreadCount = null;

    private static final int THREAD_SIZE = 10;
    public static void main(String[] args) throws Exception {

        if (args.length <=2) {
            System.out.println("Usage: java JGrep 対象ディレクトリ  正規表現  [ ファイルフィルタ ]");
            System.exit(-1);
        }

        String fileFilter = "";
        if (args.length > 2) {
            fileFilter = args[2];
        }
        (new JGrep()).grep(args[0], args[1], fileFilter);
    }

    public void grep(String dirName, String expr, String filter) throws Exception {

        // 対象ファイルを取得
        File dir = new File(dirName);
        final String fltr = filter;
        File[] files = dir.listFiles(
                new FilenameFilter(){
```

```
                public boolean accept(File dir, String name) {
                    return name.matches(fltr);
                }
            });

        // 出力先ディレクトリ
        File outDir = new File(dir.getAbsolutePath() + File.separator + "out");
        if (!outDir.exists()) {
            if (!outDir.mkdirs()) {
                throw new IllegalStateException(" 出力ディレクトリが作成できません ");
            }
        }

        // スレッドで処理させる単位 ( バケツ ) のサイズを決定
        int buketSize = files.length / THREAD_SIZE;
        buketSize++;

        int filesInBuketCnt = 0;
        List<List<File>> buketBuket = new ArrayList<List<File>>();
        List<File> fileBuket = null;

        for (File file : files) {
            if (filesInBuketCnt == 0) {
                fileBuket = new ArrayList<File>();
            }
            fileBuket.add(file);
            filesInBuketCnt++;
            if (filesInBuketCnt > buketSize) {
                buketBuket.add(fileBuket);
                filesInBuketCnt = 0;
                fileBuket = null;
            }
        }
        if (files !=null && filesInBuketCnt > 0) {
            buketBuket.add(fileBuket);
        }

        // スレッド数を登録
        workingThreadCount = new AtomicInteger(buketBuket.size());

        // 非同期処理を行う
        Executor executer = new Executor() {
            public void execute(Runnable command) {
                new Thread(command).start();
            }
        };

        int threadId = 1;
        for (List<File> lstFiles : buketBuket) {
            executer.execute(
                new GrepCommand(threadId++,
                        (File[])lstFiles.toArray(new File[0]),
                        outDir,
                        expr));
        }
    }

    /**
     * grep を実行する
     */
    public static class GrepCommand implements Runnable {
        private int threadId;
        private File[] files;
        private File outDir;
        private String expr;

        /**
         * 対象ファイルから、正規表現に一致する行を、出力ディレクトリにスレッド識別用 ID ファイル名にて
結果出力する
         * @param threadId スレッド識別用 ID
         * @param files grep 対象ファイル
         * @param outDir 結果出力ディレクトリ
         * @param expr 正規表現 (Java)
         */
        public GrepCommand(int threadId, File[] files, File outDir, String expr) {
            if (outDir == null || !outDir.isDirectory()) {
                throw new IllegalArgumentException(" 不正な出力先 ");
            }
            this.threadId = threadId;
            this.files = files;
            this.outDir = outDir;
            this.expr = expr;
```

```
        }

        /* (non-Javadoc)
         * @see java.lang.Runnable#run()
         */
        public void run() {
            try {
                File outFile = new File(this.outDir.getAbsolutePath()
                        + File.separator
                        + String.format("%03d.txt", this.threadId));
                BufferedWriter writer = new BufferedWriter(
                        new OutputStreamWriter(new FileOutputStream(outFile)));

                for (File file : files) {
                    System.out.printf("[%03d] 処理中 ... %s\n", this.threadId, file.getName());
                    BufferedReader reader = new BufferedReader(
                            new InputStreamReader(new FileInputStream(file)));

                    long lineno = 0;
                    boolean isFirstMatch = false;
                    String line = null;
                    while((line = reader.readLine()) != null) {
                        lineno++;
                        if (line.matches(this.expr)) {
                            if (!isFirstMatch) {
                                writer.write("\n------------------------------\n");
                                writer.write(file.getAbsolutePath());
                                writer.write("\n------------------------------\n");
                                isFirstMatch = true;
                            }
                            writer.write(line + "\n");
                        }
                    }
                    reader.close();
                }

                writer.close();

            } catch (Exception e) {
                System.out.println(e);
                System.exit(-1);
            } finally {
                // 動いているスレッドが無くなったら、終了メッセージを出力
                int remain = workingThreadCount.addAndGet(-1);
                System.out.println("残りのスレッド数 ... " + remain);
                if ( remain <= 0) {
                    System.out.println("終了しました。");
                }
            }
        }
    }
}
```

実行例 11

```
> java JGrep "C:\\work\\" ".*TEST.*" ".*\.txt$"

[006] 処理中 ... WORK16.txt
[005] 処理中 ... WORK11.txt
[007] 処理中 ... WORK21.txt
[001] 処理中 ... TEST1028.txt
[002] 処理中 ... TEST2085.txt
[002] 処理中 ... TEST2089.txt
[002] 処理中 ... TEST2096.txt
[001] 処理中 ... TEST1135.txt
[006] 処理中 ... WORK17.txt
[007] 処理中 ... WORK23.txt
[008] 処理中 ... WORK25_M25.txt
残りのスレッド数 ... 7
[006] 処理中 ... WORK17.h.txt
[001] 処理中 ... TEST2076.txt
[007] 処理中 ... WORK24.txt
[006] 処理中 ... WORK19.txt
[002] 処理中 ... TEST2097Z.txt
残りのスレッド数 ... 6
[007] 処理中 ... WORK25#.txt
残りのスレッド数 ... 5
[005] 処理中 ... WORK11_M24.txt
```

```
残りのスレッド数 ... 4
残りのスレッド数 ... 3
[008] 処理中 ... WORK27.txt
[005] 処理中 ... WORK11_NEW.txt
[008] 処理中 ... コピー ～ TEST2085.txt
[005] 処理中 ... WORK11_W.txt
[007] 処理中 ... WORK25.20090630.txt
残りのスレッド数 ... 2
[007] 処理中 ... WORK25.txt
[005] 処理中 ... WORK11_WW.txt
[005] 処理中 ... WORK12.txt
残りのスレッド数 ... 1
残りのスレッド数 ... 0
終了しました。
```

非同期に grep を実施する例 2

内容

・上記例を書き換え
・ExecutorService を使って複数スレッドを管理
・Runnable の代わりに、Callable を利用して、結果を受け取る
・Future にて結果を表示

■ ExecutorService

・終了を管理するメソッド、および 1 つ以上の非同期タスクの進行状況を追跡する Future を生成できるメソッドを提供

■ Future

・非同期計算の結果を表す
・計算が完了したかどうかのチェック、完了までの待機、計算結果の取得などを行うためのメソッドを用意

■ Callable

・結果を返し、例外をスローすることがあるタスク
・Runnable と似ていて、どちらもインスタンスが別のスレッドによって実行される可能性があるクラス用に設計
・Runnable は結果を返さず、チェック例外をスローすることはできません

■ソース

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

/**
 *
 */
public class JGrep {
```

```
private static final int THREAD_SIZE = 10;
// スレッドプール
private ExecutorService threadPool;

public static void main(String[] args) throws Exception {

    if (args.length <=2) {
        System.out.println("Usage: java JGrep 対象ディレクトリ  正規表現  [ ファイルフィルタ ]");
        System.exit(-1);
    }

    String fileFilter = "";
    if (args.length > 2) {
        fileFilter = args[2];
    }
    (new JGrep()).grep(args[0], args[1], fileFilter);
}

/**
 * @param dirName
 * @param expr
 * @param filter
 * @throws Exception
 */
public void grep(String dirName, String expr, String filter) throws Exception {

    // 対象ファイルを取得
    File dir = new File(dirName);
    final String fltr = filter;
    File[] files = dir.listFiles(
            new FilenameFilter(){
                public boolean accept(File dir, String name) {
                    return name.matches(fltr);
                }
            });

    // 出力先ディレクトリ
    File outDir = new File(dir.getAbsolutePath() + File.separator + "out");
    if (!outDir.exists()) {
        if (!outDir.mkdirs()) {
            throw new IllegalStateException(" 出力ディレクトリが作成できません ");
        }
    }

    // スレッドで処理させる単位 ( バケツ ) のサイズを決定
    int buketSize = files.length / THREAD_SIZE;
    buketSize++;

    int filesInBuketCnt = 0;
    List<List<File>> buketBuket = new ArrayList<List<File>>();
    List<File> fileBuket = null;

    for (File file : files) {
        if (filesInBuketCnt == 0) {
            fileBuket = new ArrayList<File>();
        }
        fileBuket.add(file);
        filesInBuketCnt++;
        if (filesInBuketCnt > buketSize) {
            buketBuket.add(fileBuket);
            filesInBuketCnt = 0;
            fileBuket = null;
        }
    }
    if (files !=null && filesInBuketCnt > 0) {
        buketBuket.add(fileBuket);
    }

    // Callable は、Runnabla と同じような役割を担うが、
    // Runnable と異なり、結果を返し、例外をスローすることができる
    List<Callable<Long>> tasks = new ArrayList<Callable<Long>>();
    threadPool = Executors.newFixedThreadPool(THREAD_SIZE);

    int threadId = 1;
    for (List<File> lstFiles : buketBuket) {
        tasks.add(
            new GrepCommand(threadId++,
                    (File[])lstFiles.toArray(new File[0]),
                    outDir,
                    expr));
    }
```

```
        // 指定されたタスクを実行し、すべて完了すると、ステータスと結果を含む Future のリストを返す
        List<Future<Long>> results = threadPool.invokeAll(tasks);
        for (Future<Long> f : results) {
            System.out.printf("一致件数 %d 件\n", f.get().longValue());
        }
        System.out.println("終了しました。");

    }

    /**
     * grep を実行する
     */
    public static class GrepCommand implements Callable<Long> {
        private int threadId;
        private File[] files;
        private File outDir;
        private String expr;

        /**
         * 対象ファイルから、正規表現に一致する行を、出力ディレクトリにスレッド識別用 ID ファイル名にて結果出力する
         * @param threadId スレッド識別用 ID
         * @param files grep 対象ファイル
         * @param outDir 結果出力ディレクトリ
         * @param expr 正規表現 (Java)
         */
        public GrepCommand(int threadId, File[] files, File outDir, String expr) {
            if (outDir == null || !outDir.isDirectory()) {
                throw new IllegalArgumentException("不正な出力先");
            }
            this.threadId = threadId;
            this.files = files;
            this.outDir = outDir;
            this.expr = expr;
        }

        /* (non-Javadoc)
         * @see java.util.concurrent.Callable#call()
         */
        public Long call() throws Exception {
            File outFile = new File(this.outDir.getAbsolutePath()
                    + File.separator
                    + String.format("%03d.txt", this.threadId));

            BufferedWriter writer = new BufferedWriter(
                    new OutputStreamWriter(new FileOutputStream(outFile)));

            long match = 0L;
            for (File file : files) {
                System.out.printf("[%03d] 処理中 ... %s\n", this.threadId, file.getName());
                BufferedReader reader = new BufferedReader(
                        new InputStreamReader(new FileInputStream(file)));

                long lineno = 0;
                boolean isFirstMatch = false;
                String line = null;
                while((line = reader.readLine()) != null) {
                    lineno++;
                    if (line.matches(this.expr)) {
                        if (!isFirstMatch) {
                            writer.write("\n--------------------------------\n");
                            writer.write(file.getAbsolutePath());
                            writer.write("\n--------------------------------\n");
                            isFirstMatch = true;
                        }
                        writer.write(line + "\n");
                        match++;
                    }
                }
                reader.close();
            }

            writer.close();

            // 結果を返す
            return new Long(match);
        }
    }
}
```