Reti e Laboratorio III
Modulo Laboratorio III
AA. 2025-26

docente: Laura Ricci

laura.ricci@unipi.it

Correzione Assignment 5

”Analisi di un weblog”
14/11/2025

ANALISI DI UN WEBLOG

Il log file di un web server contiene un insieme di linee, con il seguente formato:

```
150.108.64.57 - - [15/Feb/2001:09:40:58 -0500] "GET / HTTP 1.0" 200 2511
```

in cui:

```
- 150.108.64.57 indica l'host remoto, in genere secondo la dotted quad form
- [data]
- "HTTP request"
- status
- bytes sent
- eventuale tipo del client "Mozilla/4.0......."
```

- scrivere un'applicazione Weblog che prende in input il nome del log file (che sarà fornito) e ne stampa ogni linea, in cui ogni indirizzo IP è sostituito con l'hostname
- sviluppare due versioni del programma, la prima single-threaded, la seconda invece utilizza un thread pool, in cui il task assegnato ad ogni thread riguarda la traduzione di un insieme di linee del file. Confrontare i tempi delle due versioni.

VERSIONE SEMPLIFICATA

```
public class Weblog {
    // Tempo di permanenza in cache.
    public static final int cachingTime = 600;
    public static final String stringCachingTime = String.format("%s", cachingTime);
    public static void main(String[] args) {
        // Lettura dei parametri da riga di comando.
        if (args.length < 2) {
            System.err.println("Usage: Weblog <inputFile> <outputFile>");
            System.exit(1);
        }
        String inputFile = args[0],
        outputFile = args[1];
        // Imposto il tempo di permamenza in cache degli indirizzi tradotti.
        Security.setProperty("networkaddress.cache.ttl", stringCachingTime));
        long start = System.currentTimeMillis();
        int count = 0;
```

VERSIONE SEMPLIFICATA

```
try (
    BufferedReader in = new BufferedReader(new FileReader(inputFile));
    PrintWriter out = new PrintWriter(outputFile);
) {
    String line = null;
    // per ogni riga estraggo l'indirizzo IP e lo traduco.
    while ((line = in.readLine()) != null) {
        String[] parts = line.split("-", 2);
        String address = parts[0].trim();
        String hostname = InetAddress.getByName(address).getHostName();
        // Scrivo la riga tradotta sul file di output.
        out.printf("%s -%s\n", hostname, parts[1]);
        count++;
    }
}
```

VERSIONE SEMPLIFICATA

```
        catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            System.exit(1);
        }
        long end = System.currentTimeMillis();
        // Stampo il numero di righe lette e il tempo impiegato (in ms).
        System.out.printf(
            "N. of lines\t: %d\nElapsed time\t: %d ms\n", count, end-start
        );
    }
}
```

ELEMENTO DEL LOG

```
/**
 *Reti e laboratorio III - A.A. 2022/2023
 *Soluzione del sesto ASSIGNMENT
 *
 */

public class Element implements Comparable<Element> {
    public final int id;              // Numero della riga.
    public final String line;         // Contenuto della riga.


    public Element(int id, String line) {
        this.id = id;
        this.line = line;
    }
```

CONFRONTARE GLI ELEMENTI

```
/**
* Confronta l'elemento corrente con il parametro.
* @param o elemento da confrontare con quello corrente
* @return il valore 0 se <code>this.id == o.id</code>,
* un valore < 0 se <code>this.id < o.id</code>
* e un valore > 0 se <code>this.id > o.id</code>.
*/

@Override
public int compareTo(Element o) {
    return Integer.compare(this.id, o.id);
}
```

CONSUMER TASK

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.List;
import java.util.concurrent.BlockingQueue;

/**
 *La classe rappresenta il generico task Consumer,
 *che si occupa di effettuare la traduzione di un insieme
 *di righe del file originale. Le righe tradotte vengono
 *inserite in una coda con priorita'.
 */

public class Consumer implements Runnable {
    private List<Element> batch;
    private BlockingQueue<Element> outputQueue;
```

CONSUMER TASK

```
    public Consumer(List<Element> batch, BlockingQueue<Element> outputQueue) {
        this.batch = batch;
        this.outputQueue = outputQueue;
    }

    public void run() {
        for (Element element : batch) {
            String[] parts = element.line.split("-", 2);
            String address = parts[0].trim();
            String hostname = null;
            try {hostname = InetAddress.getByName(address).getHostName();}
            catch (UnknownHostException e) {}
            String translated = String.format("%s -%s\n", hostname, parts[1]);;
            outputQueue.add(new Element(element.id, translated));
        }
    }
}
```

VERSIONE CON THREADPOOL

```
public class WeblogM {
    // Tempo di permanenza in cache.
    public static final int cachingTime = 600;
    public static final String stringCachingTime = String.format("%s", cachingTime);
    // Coda con priorita' in cui vengono memorizzate le righe tradotte.
    public static BlockingQueue<Element> outputQueue = new PriorityBlockingQueue<>();
    // Pool di thread.
    public static ExecutorService pool = Executors.newCachedThreadPool();
    // Tempo massimo di attesa per la terminazione del pool.
    public static final int maxDelay = 60000;

    public static void main(String[] args) {
        // Controllo dei parametri da riga di comando.
        if (args.length < 3) {
            System.err.println("Usage: <inputFile> <outputFile> <maxBatchSize>");
            System.exit(1);
        }
```

CALCOLO BATCH

```
int maxBatchSize = Integer.parseInt(args[2]), numLines = 0;
// Imposto il tempo di permanenza in cache degli indirizzi tradotti.
Security.setProperty("networkaddress.cache.ttl", stringCachingTime);
// Apro i file di input e di output.
long start = System.currentTimeMillis();
try (
    BufferedReader in = new BufferedReader(new FileReader(args[0]));
    PrintWriter out = new PrintWriter(args[1]);
) {
    // Leggo il file di input riga per riga.
    // Creo gruppi di `batchSize` righe e per ciascun gruppo
    // attivo un Consumer che effettua la traduzione.
    String line = null;
    List<Element> batch = new ArrayList<>();
```

CALCOLO BATCH

```
while ((line = in.readLine()) != null) {
    // Se la lista ha raggiunto la capacita' massima,
    // la passo al Consumer e la (re-)inizializzo.
    if (batch.size() == maxBatchSize) {
        pool.execute(new Consumer(batch, outputQueue));
        batch = new ArrayList<>();
    }
    // Aggiungo la nuova riga alla lista.
    batch.add(new Element(numLines, line));
    numLines++;
}
// A questo punto, attendo la terminazione del pool.
awaitPoolTermination();
```

SCRITTURA RISULTATI

```
            // Scrivo sul file di output le righe
            // Per ottenere le righe nell'ordine corretto, chiamo ripetutamente
            // il metodo take() che aspetta finché non c'è un elemento in testa
            // da prendere, a differenza del poll() che ha un timeout
            while (outputQueue.size() > 0) {
                Element element = outputQueue.take();
                out.print(element.line);
            }
        }
        catch (Exception e) {
            System.err.printf("Errore: %s\n", e.getMessage());
        }
        long end = System.currentTimeMillis();
        // Stampo il numero di righe lette e il tempo impiegato (in ms).
        System.out.printf(
            "N. of lines\t: %d\nElapsed time\t: %d ms\n", numLines, end-start
        );
    }
```

TERMINAZIONE THREADPOOL

```
/**
 * Avvia la procedura di terminazione del pool.
 */
public static void awaitPoolTermination() {
    pool.shutdown();
    try {
        if (!pool.awaitTermination(maxDelay, TimeUnit.MILLISECONDS))
            pool.shutdownNow();
    }
    catch (InterruptedException e) {pool.shutdownNow();}
}
```