Producteur Consommateur

La création d'un thread consomme une quantité importante de mémoire. Dans une application où il y a beaucoup de programmes clients, la création d'un thread par client n'évoluera pas. Ainsi, Java a proposé un framework d'exécuteur pour fournir un pool de threads pour l'exécution limitant le nombre de threads servant la demande du client à tout moment. Cela améliore les performances et réduit les besoins en mémoire.

Java 5 fournit également des implémentations de file d'attente de blocage et nous n'avons plus besoin de contrôler les applications producteur/consommateur à l'aide de wait/notify. Ceci est automatiquement pris en charge par les implémentations de BlockingQueue.

Un exemple de producteur/consommateur utilisant une implémentation de file d'attente bloquante et un framework d'exécuteur est le suivant :

```
public class ProducerConsumer {

    private static final int NUM_OF_MSGS = 20;
    private static final BlockingQueue<String> queue
                                            = new ArrayBlockingQueue<String>(5);
    private static ExecutorService producerPool = Executors.newFixedThreadPool(3);
    private static ExecutorService consumerPool = Executors.newFixedThreadPool(1);

    private static Logger logger =  Logger.getLogger(ProducerConsumer.class.getName());

    public static void main(String[] args) {
        Runnable producerTask = () -> {
            try {
                queue.put("test Message");
                System.out.println(Thread.currentThread().getName()
                                    + " put message queue.size() " + queue.size());
            } catch (InterruptedException e) {
                logger.log(Level.SEVERE, e.getMessage(), e);
            }
        };
        Runnable consumerTask = () -> {
            try {
                System.out.println(Thread.currentThread().getName()
```

```
                                    + " received msg " + queue.take());

            } catch (InterruptedException e) {
                logger.log(Level.SEVERE, e.getMessage(), e);
            }
        };
        try {
            for (int i = 0; i < NUM_OF_MSGS; i++) {
                producerPool.submit(producerTask);
            }
            for (int i = 0; i < NUM_OF_MSGS; i++) {
                consumerPool.submit(consumerTask);
            }
        } finally {
            if (producerPool != null) {
                producerPool.shutdown();
            }
            if (consumerPool != null) {
                consumerPool.shutdown();
            }
        }
    }

}
```

Le but de cela est de faire des opérations longues en mode asynchrone.

Ici nous avons une double queue avec une conversion HTML->PDF

```
public class ProducerConsumer2 {

    private static final int NUM_OF_MSGS = 20;
    private static final BlockingQueue<String> queueHtml
                                            = new ArrayBlockingQueue<String>(5);
    private static final BlockingQueue<File> queueFile= new ArrayBlockingQueue<File>(5);

    private static ExecutorService consumerHtmlPool = Executors.newFixedThreadPool(4);
    private static ExecutorService consumerFilePool = Executors.newFixedThreadPool(4);
```

```
    private static Logger logger =  Logger.getLogger(ProducerConsumer2.class.getName());

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

        Runnable consumerTask = () -> {
            try {

            String html=queueHtml.take();
            UUID uuid=UUID.randomUUID();
            File outputFile=new File(uuid.toString()+".pdf");
            HtmlConverter.convertToPdf(html, new FileOutputStream(outputFile));
            queueFile.put(outputFile);
            } catch (InterruptedException e) {
                logger.log(Level.SEVERE, e.getMessage(), e);
            } catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
        };
        Runnable consumerFileTask = () -> {
            try {

            File pdfFile=queueFile.take();
            System.err.println(pdfFile.getAbsolutePath());
            } catch (InterruptedException e) {
                logger.log(Level.SEVERE, e.getMessage(), e);
            }
        };
        try {
            for (int i = 0; i < NUM_OF_MSGS; i++) {
            consumerHtmlPool.submit(consumerFileTask);
            }
            for (int i = 0; i < NUM_OF_MSGS; i++) {
            consumerFilePool.submit(consumerTask);
            }
            queueHtml.add("\"<h1>Hello</h1>\"\r\n"
```

```
            □□+ "□□□+ \"<p>This was created using iText</p>\"\r\n"
            □□+ "□□□");
            System.in.read();
        } finally {
            if (consumerHtmlPool != null) {
            □consumerHtmlPool.shutdown();
            }
            if (consumerFilePool != null) {
            □consumerFilePool.shutdown();
            }
        }
    }

}
```

Le pom doit être modifié ainsi:

```
□  <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>itext7-core</artifactId>
        <version>7.1.9</version>
        <type>pom</type>
    </dependency>

    <!-- iText pdfHTML add-on -->
    <dependency>
□    <groupId>com.itextpdf</groupId>
□    <artifactId>html2pdf</artifactId>
□    <version>2.1.6</version>
□</dependency>
```

Revision #2
Created 1 June 2022 18:21:27 by ggpilou2
Updated 19 June 2022 10:52:23 by ggpilou2