Blog

We believe there is something unique at every business that will ignite the fuse of innovation.

June 5, 2017

Spring Batch Multi-Threaded-Step

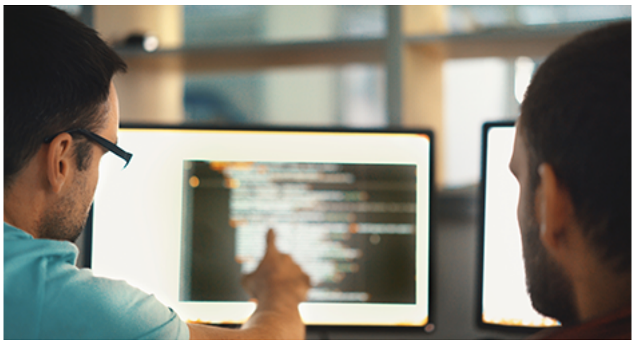
Written By:
RYAN TARRANT (/search#q=Ryan Tarrant)

TAGS

software engineering, spring, java, technology solutions, software development, programming, performance, patterns, maven, development

SHARE (http url=

Have you written or thinking about writing a Spring Batch (http://projects.spring.io/spring-batch/) application? If so, have you considered multi-threading or parallel processing? If you answered YES, then you should find this post helpful.



Purpose of this Post:

- Solve the following problem: *Improve performance and processing time for an application that processes a large quantity of files in bulk*.
- Show an implementation of Spring Batch using the multi-threaded-step approach.
- Describe and analyze the multi-threaded-step approach behavior.

Resources

- Spring Batch (http://projects.spring.io/spring-batch/) project
- Spring Batch's multi-threaded-step approach (http://docs.spring.io/spring-batch/trunk/reference/html/scalability.html)
- My github multi-threaded-step example (https://github.com/tarrantrj/spring-batch-examples/tree/master/multi-threaded-step) repository

Implementation

For simplicity, the multi-threaded-step consists of a Reader, Processor, and Writer. The project consists of a single job and a single step. Refer to my github repository in the resources for the complete source code.

This implementation uses a Java configuration rather than an XML configuration. The complete configuration can be found at Application.java (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/Application.java). To implement a multi-threaded-step follow these steps in your java configuration file.

- Create a Task Executor bean. In this case I used SimpleAsyncTaskExecutor (http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/core/task/SimpleAsyncTaskExecutor.html).

```
@Bean
public TaskExecutor taskExecutor() {
        SimpleAsyncTaskExecutor taskExecutor = new SimpleAsyncTaskExecutor();
        taskExecutor.setConcurrencyLimit(maxThreads);
        return taskExecutor;
}
```

- Add Task Executor to Step bean by calling `taskExecutor()` . Also add a throttle limit below.

```
@Bean
public Step step() {
        return stepBuilderFactory.get("step").<Attempt, Attempt>chunk(chunkSize)
                .reader(processAttemptReader())
                .processor(processAttemptProcessor())
                .writer(processAttemptWriter())
                .taskExecutor(taskExecutor())
                .listener(stepExecutionListener())
                .listener(chunkListener())
                .throttleLimit(maxThreads).build();
}
```

- The `maxThreads` variable is set in the application.properties (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/resources/application.properties). file and then is added to the configuration file using the `@Value` annotation as follows:

```
@Value("${max-threads}")
private int maxThreads;
```

- When implementing a multi-threaded solution, it is important to use the keyword `synchronized` on a shared resource or shared method. In the example provided here, I `synchronized` the `read()` method to prevent thread collision on the queue. Here is the excerpt from the AttemptReader (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/AttemptReader.java):

```
public synchronized Attempt read() throws Exception {
        Attempt attempt = null;
        logger.info("Attempt Queue size {}.", attemptQueue.size());
        if (attemptQueue.size() > 0) {
                attempt = attemptQueue.remove();
        }
        return attempt;
}
```

The following is focused on the Spring Batch steps for this implementation for those that are newer to Spring Batch and want a little more explanation.

- The reader and writer are necessary to create the step while the processor and listeners are optional. As the reader, processor, and writer are typical code for Spring Batch, I will not go into details of implementation in this post, but you can reference the code here: AttemptReader (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/AttemptReader.java), AttemptProcessor (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/AttemptProcessor.java), and AttemptWriter (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/AttemptWriter.java).
- Adding the Job bean using `JobBuilderFactory` and adding the step to the flow:

```
@Bean
public Job processAttemptJob() {
        return jobBuilderFactory.get("process-attempt-job")
                .incrementer(new RunIdIncrementer())
                .listener(jobExecutionListener())
                .flow(step()).end().build();
}
```

- OPTIONAL: For logging and analysis, the following listeners were implemented in this project: StepExecutionNotificationListener (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/StepExecutionNotificationListener.java), Chu (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/ChunkExecutionListener.java), and JobCompletionNotificationListener (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/src/main/java/com/captechconsulting/springbatchexamples/multithreadedstep/JobCompletionNotificationListener.java). Simply implement the classes, add the listeners in the step bean or job bean (above), and add the following beans to the configuration file:

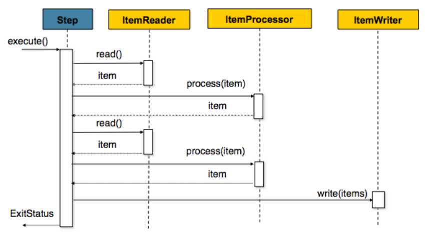
```
@Bean
public StepExecutionNotificationListener stepExecutionListener() {
        return new StepExecutionNotificationListener();
}

@Bean
public ChunkExecutionListener chunkListener() {
        return new ChunkExecutionListener();
}

@Bean
public JobCompletionNotificationListener jobExecutionListener() {
        return new JobCompletionNotificationListener();
}
```

Analysis

Chunk Oriented Processing Review

Before understanding how Spring Batch works using multi-threaded-step, you should understand Chunk Oriented Processing (http://docs.spring.io/spring-batch/trunk/reference/htmlsingle/#chunkOrientedProcessing).

To review quickly for you the following picture shows how a single step would be processed for a single chunk of size 2.



(http://docs.spring.io/spring-batch/trunk/reference/htmlsingle/images/chunk-oriented-processing.png)
For instance, in a single threaded application that uses chunk oriented processing, an item will be read by the `ItemReader`, processed by the `ItemProcessor`, and then queued. Once all of the items in a given chunk are read and processed, the `ItemWriter` will perform a write for an array of items provided in a chunk.

Multi-Threaded-Step Analysis

Now how does this affect the *multi-threaded-step*? The multi-threaded-step will spawn multiple chunks up to the max-threads applied.

Example 1:

- items (files) = 150
- chunk-size = 5
- max-threads = 10

In example 1, 10 threads are spawned at a time. Each thread consists of a single chunk that contains 5 items (files). The chunk reads and processes 1 file at a time and after 5 files have been read/processed, the files are written 5 files at a time.

Since 10 threads are spawned, 10 chunks are kicked off simultaneously meaning 10 files will be processed simultaneously.

For this example of 150 items, 30 chunks are created with 5 items each. I included the log (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/sample-output/multi-threaded-step.files150-chunk5-threads10.log) file for this run in my github repository, and the *Write attempt* log shows the list of the files (items) that are in a single chunk in a single thread.

Example 2:

- items (files) = 50
- chunk-size = 7
- max-threads = 10

In example 2, 10 threads are spawned at a time. Each thread consists of a single chunk. Most of the chunks contain 7 items (files). The last chunk initiated will contain 3 files. Similarly, each chunk reads and processes 1 file at a time and after 5 files have been read / processed, the files are written with all of the files in a chunk at a time.

Again with 10 threads spawned, 10 chunks are kicked off simultaneously meaning 10 files will be processed simultaneously.

For this example of 150 items, 22 chunks are created. 21 of the chunks will have 7 items each, and 1 chunk will have 3 items in it. The chunk with the 3 items in it will be the last chunk created. Again I included the log (https://github.com/tarrantrj/spring-batch-examples/blob/master/multi-threaded-step/sample-output/multi-threaded-step.files150-chunk7-threads10.log) for this run in my github repository, and the *"Write attempt"* text in the log showing the list of files (items) that are in a single chunk in a single thread.

Important note: The last 2 chunks start at approximately the same time. Despite the thread/chunk with 5 items having started first, the last thread/chunk with 3 items will finish first. This happens because the last thread/chunk required less processing time.

Why Use this Approach?

This post should provide you with a simple Spring Batch application using multi-threaded steps at an introductory level. This approach was used at a client for moving thousands of files at a time to a document repository with specific metadata. The solution needed to be flexible to handle automatically uploading both historic files and files currently being created daily. The multi-threaded step approach worked best here because the process was the same single process and the steps were repeated on large scale.

Take Note

A few notes that might help prevent some heartache in the end are:

- Use the keyword `synchronized` on a shared resource or shared method.
- Test and analyze your ideal thread count and chunk size. Run some performance analysis using various inputs. Increasing thread count is not always the answer.
- Ensure the hardware or server can handle your processing
- Start debugging with a single-thread first

About the Author

Ryan Tarrant is a software developer in our Philadelphia office (~/link.aspx?_id=BF553E5CE7B14324BD3752F18FEE082A&_z=z). With over 11 years of experience, Ryan enjoys staying on the cutting edge of technology, and he specializes in Java, System Integration, Analysis and Web Services.

CapTech
@CapTechListens
“Being included in Washington Post’s Top Workplaces award is energizing and helps us pulse check that we are actively engaging in achieving our mission to be the best consulting firm to work for and work with." bit.ly/2prV0rb

Washington Post Ranks CapTech in the Top Wo…
June 22, 2018 – Today, Washington Post recognized CapTech as a winner of its Top Workplaces 2018 award. The Post’s annual rankin...
captechconsulting.com

1h