# Multi-threaded Simultaneous Processing for Massive Processing

## Outline

This Section describes simultaneous processing of jobs for massive processing, by way of multi-threaded step, parallel step or partitioning step.

## Description

### Multi-threaded Step Example

Simultaneous Multi-threaded Processing requires a step to be processed multi-threaded. You need to process a thread by unit of chunk.

#### Job Configurations

##### Check out parallelJob.xml, the job configuration file for simultaneous multi-thread example.

Apply asynchronous configuration in tasklet of the step that you need to proceed with multi-thread step as follows:

```
<job id="parallelJob" xmlns="http://www.springframework.org/schema/batch">
	<step id="staging" next="loading">
		<tasklet>
			<chunk reader="fileItemReader" processor="validatingProcessor"
writer="stagingItemWriter" commit-interval="2"/>
		</tasklet>
	</step>
	<step id="loading">
		<tasklet task-executor="taskExecutor">
			<chunk reader="stagingReader" processor="stagingProcessor" writer="tradeWriter"
commit-interval="3"/>
		</tasklet>
	</step>
</job>

<bean id="taskExecutor" class="org.springframework.core.task.SimpleAsyncTaskExecutor" />
```

#### Composition and Implementation of JunitTest

##### Composition of JunitTest

Implementation of Junit Test using concurrent multi-threaded configuration and associated class is required, whereby the batch is implemented and the information are described.

✔ See Junit Test Description for Batch Execution Environment for structure of the Class JunitTest.

✔ assertEquals(BatchStatus.COMPLETED, execution.getStatus()) : Make sure you check the batch execution result is COMPLETED.

✔ assertEquals(after - before, execution.getStepExecutions().iterator().next().getReadCount()) : Check out the result of data and stepexecution in BATCH_STAGING to check the result of step for staging.

```
@RunWith(SpringJUnit4ClassRunner.class)
```

```
@ContextConfiguration(locations = { "/egovframework/batch/simple-job-launcher-context.xml",
"/egovframework/batch/jobs/parallelJob.xml",
                "/egovframework/batch/job-runner-context.xml"    })
public class EgovParallelJobFunctionalTests {

        // JobLauncherTestUtils to test the batches.
        @Autowired
        private JobLauncherTestUtils jobLauncherTestUtils;

        // SimpleJdbcTemplate for using DB
        private SimpleJdbcTemplate jdbcTemplate;

        ...

        @Test
        public void testLaunchJob() throws Exception {
                int before = SimpleJdbcTestUtils.countRowsInTable(jdbcTemplate, "BATCH_STAGING");
                JobExecution execution = jobLauncherTestUtils.launchJob();
                int after = SimpleJdbcTestUtils.countRowsInTable(jdbcTemplate, "BATCH_STAGING");
                assertEquals(BatchStatus.COMPLETED, execution.getStatus());
                assertEquals(after - before, execution.getStepExecutions().iterator().next().getReadCount());
        }

}
```

**Implementation of JunitTest**

See Implementation of JunitTest for more information.

### Verify Result

**Multi-threaded steps do not work simultaneously due to difference in processing. You can check out these in TRADE data representing the result of step for loading.**

Status | Result1

| | ID | VERSION | ISIN | QUANTITY | PRICE | CUSTOMER |
|---|---|---|---|---|---|---|
| 1 | 6 | 0 | UK21341... | 213 | 33.11 | customer3 |
| 2 | 7 | 0 | UK21341... | 212 | 32.11 | customer2 |
| 3 | 8 | 0 | UK21341... | 214 | 34.11 | customer4 |
| 4 | 9 | 0 | UK21341... | 211 | 31.11 | customer1 |
| 5 | 10 | 0 | UK21341... | 215 | 35.11 | customer5 |

## Parallel Examples

parallelStep implements parallel processing of a pair of split flows. In parallelStep, Flow 1 and Flow 2 are processed in parallel in each thread.

### Job Configurations

**Check out parallelStep.xml, the job configuration file for ParallelStep example.**

✔ You can work on parallel processing using asynchronous configuration in the tag split.

```
<job id="parallelStep" xmlns="http://www.springframework.org/schema/batch">
        <split id="split1" task-executor="taskExecutor" next="step4">
                <flow>
                        <step id="step1" next="step2">
                                <tasklet>
```

```
                                        <chunk reader="itemReader" writer="itemWriter1" commit-
interval="1" />
                                    </tasklet>
                                </step>
                                <step id="step2">
                                    <tasklet>
                                        <chunk reader="itemReader" writer="itemWriter2" commit-
interval="2" />
                                    </tasklet>
                                </step>
                            </flow>
                            <flow>
                                <step id="step3">
                                    <tasklet>
                                        <chunk reader="itemReader" writer="itemWriter3" commit-
interval="2" />
                                    </tasklet>
                                </step>
                            </flow>
                        </split>
                        <step id="step4">
                            <tasklet>
                                <chunk reader="itemReader" writer="itemWriter4" commit-interval="2" />
                            </tasklet>
                        </step>
</job>

<bean id="taskExecutor" class="org.springframework.core.task.SimpleAsyncTaskExecutor" />
```

## Composition and Implementation of JunitTest

### Composition of JunitTest

Implementation of Junit Test using parallelStepconfiguration and associated class is required, whereby the batch is implemented and the information are described.

✔ See Junit Test Description for Batch Execution Environment for structure of the Class JunitTest.

✔ assertEquals(BatchStatus.COMPLETED, jobExecution.getStatus()) : Make sure you check the batch execution result is COMPLETED.

✔ Leave the location information for input and output resources required for JobParameter in getUniqueJobParameters.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations ={"/egovframework/batch/simple-job-launcher-context.xml",
"/egovframework/batch/jobs/parallelStep.xml","/egovframework/batch/job-runner-context.xml"})
public class EgovParallelStepFunctionalTests{

        // JobLauncherTestUtils to test the batches.
        @Autowired
        private JobLauncherTestUtils jobLauncherTestUtils;

        /**
         * Batch Testing
         */
        @Test
        public void testLaunchJob() throws Exception {
                JobExecution jobExecution = jobLauncherTestUtils.launchJob(this.getUniqueJobParameters());
                assertEquals(BatchStatus.COMPLETED, jobExecution.getStatus());
                // Check out outputs in /target/test-outputs/parallelStep
        }
```

```
        /**
    * Method to configure JobParameter
    * @return jobParameters
    */
        protected JobParameters getUniqueJobParameters() {
                return new
JobParametersBuilder().addString("inputFile","/egovframework/data/input/delimited.csv")
                .addString("outputFile1","file:./target/test-outputs/parallelStep/delimitedOutput1.csv")
                .addString("outputFile2","file:./target/test-outputs/parallelStep/delimitedOutput2.csv")
                .addString("outputFile3","file:./target/test-outputs/parallelStep/delimitedOutput3.csv")
                .addString("outputFile4","file:./target/test-outputs/parallelStep/delimitedOutput4.csv")
                .addParameter("timestamp", new JobParameter(new Date().getTime())).toJobParameters();
        }
}
```

#### Implementation of JunitTest

See Implementation of JunitTest for more information.

### Verify Result

The result of multi-thread implementation for a pair of split flows (SimpleAsyncTaskExecutor-1,SimpleAsyncTaskExecutor-2) can be checked in the log of console. You can check out in the result that Step 1 and Step 3 are implemented in the different thread.

```
Tasks | Properties | Servers | Coverage | Bookmarks | Console | Javadoc | SVN Repositories | SQL Result
<terminated> EgovParallelStepFunctionalTests [JUnit] C:₩eGovFrame-2.0₩bin₩jdk1.5.0_22₩bin₩javaw.exe (2012. 9. 25. 오후 4:55:10)
END_TIME, STATUS, COMMIT_COUNT, READ_COUNT, FILTER_COUNT, WRITE_COUNT, EXIT_CODE, EXIT_MESSA
READ_SKIP_COUNT, WRITE_SKIP_COUNT, PROCESS_SKIP_COUNT, ROLLBACK_COUNT, LAST_UPDATED) values(
0, 'step1', 4, 2012-09-25 16:55:14.54, null, 'STARTING', 0, 0, 0, 0, 'EXECUTING', '', 0, 0,
0, 0, 2012-09-25 16:55:14.543)
16:55:14,565  INFO SimpleAsyncTaskExecutor-2 audit:156 - 3. Connection.commit() returned
16:55:14,565  INFO SimpleAsyncTaskExecutor-1 sqltiming:235 - INSERT into BATCH_STEP_EXECUTION
END_TIME, STATUS, COMMIT_COUNT, READ_COUNT, FILTER_COUNT, WRITE_COUNT, EXIT_CODE, EXIT_MESSA
READ_SKIP_COUNT, WRITE_SKIP_COUNT, PROCESS_SKIP_COUNT, ROLLBACK_COUNT, LAST_UPDATED) values(
0, 'step1', 4, 2012-09-25 16:55:14.54, null, 'STARTING', 0, 0, 0, 0, 'EXECUTING', '', 0, 0,
0, 0, 2012-09-25 16:55:14.543)  {executed in 0 msec}
16:55:14,565  INFO SimpleAsyncTaskExecutor-2 audit:156 - 3. Connection.setAutoCommit(true) re
16:55:14,565  INFO SimpleAsyncTaskExecutor-1 audit:156 - 2. PreparedStatement.executeUpdate(
```

...

```
16:55:14,555  INFO SimpleAsyncTaskExecutor-1 audit:156 - 2. PreparedStatement.setObject(2, 0
16:55:14,558  INFO SimpleAsyncTaskExecutor-2 audit:156 - 3. PreparedStatement.setTimestamp(1
16:55:14,558  INFO SimpleAsyncTaskExecutor-1 audit:156 - 2. PreparedStatement.setString(3, "
16:55:14,558  INFO SimpleAsyncTaskExecutor-2 sqlonly:191 - INSERT into BATCH_STEP_EXECUTION(
END_TIME, STATUS, COMMIT_COUNT, READ_COUNT, FILTER_COUNT, WRITE_COUNT, EXIT_CODE, EXIT_MESSA
READ_SKIP_COUNT, WRITE_SKIP_COUNT, PROCESS_SKIP_COUNT, ROLLBACK_COUNT, LAST_UPDATED) values(
0, 'step3', 4, 2012-09-25 16:55:14.54, null, 'STARTING', 0, 0, 0, 0, 'EXECUTING', '', 0, 0,
0, 0, 2012-09-25 16:55:14.543)
16:55:14,558  INFO SimpleAsyncTaskExecutor-1 audit:156 - 2. PreparedStatement.setObject(4, 4,
16:55:14,558  INFO SimpleAsyncTaskExecutor-1 audit:156 - 2. PreparedStatement.setTimestamp(5,
```

### Partitioning Example

One another way to work on simultaneous processing is that you can partition the files data and DB data for multi-threaded step. Refer to the following example for how to proceed with SingleFile partitioning that involves reading of DB Partitioning and File Partitioning and writing of resources in a single target file:

| Type of Input Resources | Relation | Example |
|---|---|---|
| DB | | DB Partition Example |
| File | N:N | N:N Partition Example |
| File | N:1 | N:1 Partition Example |

## References

- Concurrent Processing