http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte2:brte:batch_core:parallel_process

Parallel Processing

Outline

With parallel processing of massive data, you can get the job done faster than normal as the threads are distributed all across for implementation. Parallel processing can be categorized into multi-threaded step, parallel step and partitioning step.

Description

Multi-threaded Step

You can simply add TaskExecutor to the attribute <tasklet> for parallel processing.

```
<step id="loading">
      <tasklet task-executor="taskExecutor">...</tasklet>
</step>
```

In the TaskExecutor example, you need to define beans to implement the TaskExecutor interface. With TaskExecutor being the Spring's standard of interface, you can refer to the Spring guide for more information. SimpleAsyncTaskExecutor is one of the simplest form of multi-threaded TaskExecutor where reading, processing and writing processes from chunks are processed in a number of threads distributed. This signifies that sequential processing won't be guaranteed and that a chunk of items are not consecutively arranged (provided, however, that the sequences in chunk for each chunk of items can be the same with each other thanks to commit-Interval)

✔ While the default for thread is 4, you can configure for the greater number as follows:

```
<step id="loading">
            <tasklet task-executor="taskExecutor" throttle-limit="20">
                        ...
            </tasklet>
      </step>
```

✔ Like DataSource, you can replace the pools used in Step by resources, whereby you need to configure the pools to the maximum number of threads parallelly processed.

Example

Multi-thread Examples

Parallel Steps

The logic of an execution program requiring parallel steps gets separated with a variety of responsibilities and can thus be processed in parallel, thanks to easy, intuitive implementation and composition. For instance, you can configure for the steps (step1, step2) to be processed in parallel with step3 as follows:

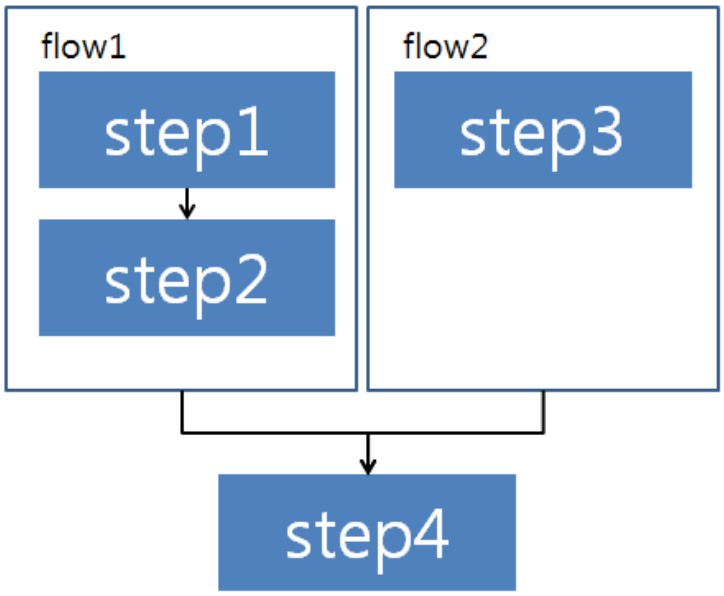
```
<job id="job1">
      <split id="split1" task-executor="taskExecutor" next="step4">
            <flow>
                  <step id="step1" parent="s1" next="step2"/>
                  <step id="step2" parent="s2"/>
            </flow>
            <flow>
                  <step id="step3" parent="s3"/>
```

```
        </flow>
    </split>
    <step id="step4" parent="s4"/>
</job>

<beans:bean id="taskExecutor" class="org.spr...SimpleAsyncTaskExecutor"/>
```

The attribute task-executor is used to designate implementation of TaskExecutor required to execute the flow. With the default being SyncTaskExecutor, you need to change the default configuration into AsyncTaskExecutor if you need asynchronous execution.

✔ The distributed tasks shall be complete before being integrated into one, as follows, and moving onto the ensuing step.

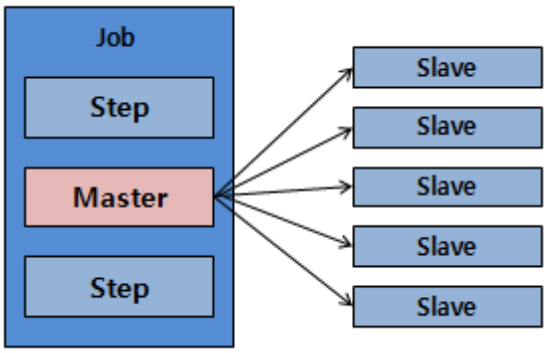


Example

Parallel Examples

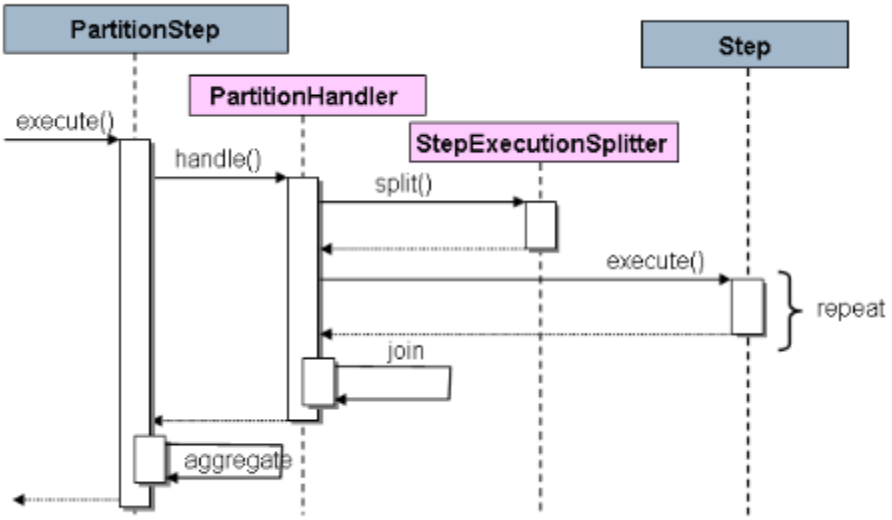
Partitioning

Spring makes feasible remote partitioning of Step using SPI. Refer to the following figure for pattern of implementation:



Note that the Job progresses from left to right as defined in the foregoing figure. One of those steps is defined as the master label and the slave labels are identified as the instances of Step whose results are reverted to the master. Being a typical form of remote service, slave is transmitted to the local thread. In Spring, it is guaranteed that, in metadata of JobRepository, each slave is executed once for each job.

In Spring, SPI comprises PartitionSteps and a pair of interfaces being PartitionHandler and StepExecutionSplitter that serve as follows:



The Step on the right is a remote slave latently having several objects and executing the roles assigned. The Partition Step comprises as follows:

```
<step id="step1.master">
      <partition step="step1" partitioner="partitioner">
            <handler grid-size="10" task-executor="taskExecutor"/>
      </partition>
</step>
```

✔ The attribute Grid-size resembles Throttle-limit for multi-threaded Step that prevents saturation of requests from each Step.

PartitionHandler

PartitionHandler is a component containing structure information of remote and grid environments and covers StepExecution in such format as DTO to remotely send th relevant information to Step. In doing so, you do not need to know how the input data are separated and how multi-threaded step results are aggregated.
With TaskExecutorPartitionHandler defined in an XML Step, you can easily execute steps in a separated thread using TaskExecutor strategy. You can also comprise TaskExecutorPartitionHandler as follows:

```
<step id="step1.master">
      <partition step="step1" handler="handler"/>
</step>

<bean class="org.spr...TaskExecutorPartitionHandler">
      <property name="taskExecutor" ref="taskExecutor"/>
      <property name="step" ref="step1" />
      <property name="gridSize" value="10" />
</bean>
```

✔ Note that gridSize refers to the number of segregated Steps to be generated and matches the number of thread pools in TaskExecutor. You can configure gridSize greater than the number of threads available, but the lowest possible.

Partitioner

Serving a similar role to the input parameter for execution of a brand-new step (you do not need to concern about restart just like that), Partitioner features the following interfaces and a single method of Partition:

```
public interface Partitioner {
    Map<String, ExecutionContext> partition(int gridSize);
}
```

, where the method returns stepExecution, the sole return for both ExecutionContext and String. The title of method matches that of the Step of stepExecution partitioned, shown in the metadata of batch. ExecutionContext is a bag of information for basic key, number of lines, directory of input files, etc. stored in the form of title and value.

Note that a StepExecution should be entitled uniquely, preferable in the form of prefix+suffix in a way the user may recognize. For instance, you can have a prefix represent the title of steps to be executed and a suffix do a counter. You may wisely use SimplePartitioner in doing so.

Binding Input Data to Steps

If you have the Steps feature the uniform composition, you can easily define input parameters to bind input data to Steps, just like the Spring's Step Scope offers.

For instance, if you generate the instance ExecutionContext that contains the key attribute of fileName, you would have the Steps target different files for the following outputs:

Step Execution Name (key)	ExecutionContext (value)
filecopy:partition0	fileName=/home/data/one
filecopy:partition1	fileName=/home/data/two
filecopy:partition2	fileName=/home/data/three

Example

Partitioning Examples

References

http://static.springsource.org/spring-batch/reference/html/scalability.html