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

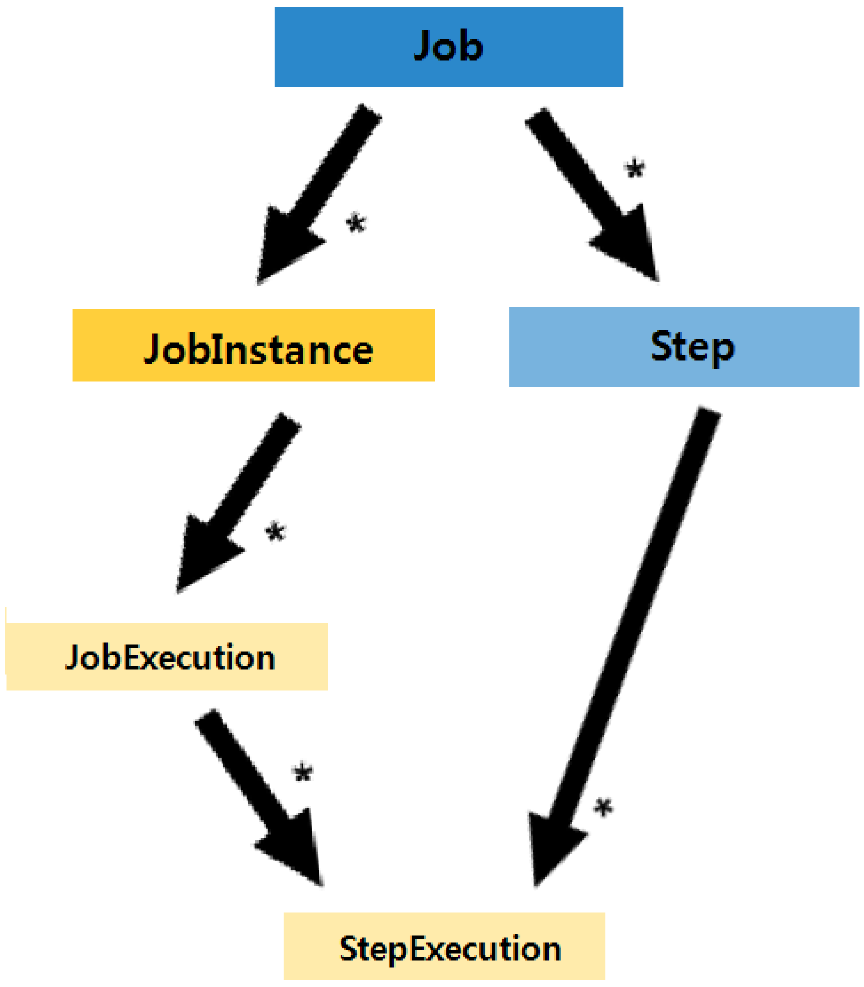
Step

Outline

Defining and controlling the tasks for actual batch processes in a job, Step configures the input resources, defines the method of processing and generates output resources.

Description

Encapsulating isolated, sequential steps of a job, Step is a domain object each of which comprises at least one Step. In teh Step you have a variety of information that is required to process and control batches. A series of steps comprise a flow that is sequentially executed. A step comprises StepExecution serving the similar function to JobExecution described above.

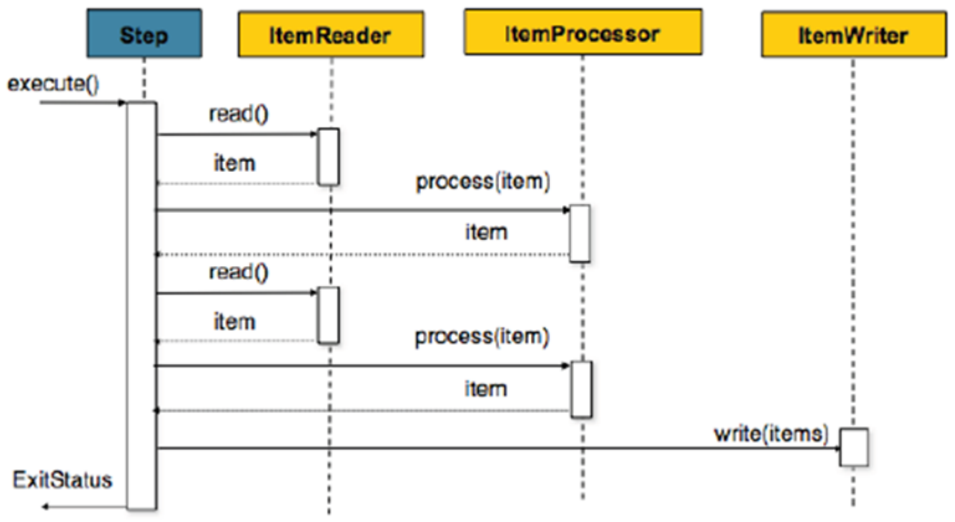


Type of Steps

Chunk-Oriented Processing

Being one of the most typical type of step, Chunk-Oriented Processing reads the data in the form of ‘chunk’, generating ‘chunk’ within the range of transaction for writing of data once and for all.
In Chunk-Oriented Processing, a single item is read via ItemReader, with a chunk of items transferred to ItemWriter.

A Chunk-Oriented Processing comprises ‘Reading Chunk Items’ → ‘Processing/Conversion’ → ‘Writing’ as follows:



Refer to the following codes that represents the foregoing example:

```
List items = new Arraylist();
for(int i = 0; i < commitInterval; i++){
       Object item = itemReader.read()

       Object processedItem = itemProcessor.process(item);

       items.add(processedItem);
}

itemWriter.write(items);
```

- Composition: ItemReader, ItemWriter, PlatformTransactionManager, JobRepository (and, optionally, ItemProcessor)
- Significant Attribute: Commit-interval(Processing count per transaction), startLimit(execution limit of Step)

TaskletStep

Structures based upon ItemReader and ItemWriter may not fit the batch-oriented environment. Where you have a batch processing completed by procedure call from DB, you might want to implement this using a simple method. TaskletStep is intended for such a case.
Being a simple interface having a single method of ‘execute()’ until either RepeatStatus.FINISHED or error is thrown, Tasklet calls repository procedure, script or simple SQL update statement.

For you to comprise TaskletStep you need to refer to th eobject “Tasklet” using the attribute ‘ref’ out of the tag <tasklet>. Note that the tag <chunk> is not used in <tasklet>. (Chunk-Oriented Processing uses the tag <Chunk>.)

```
<step id="step1">
    <tasklet ref="myTasklet"/>
</step>
```

How to implement the method execute() of Tasklet

Refer to the following example for how to execute the command “echo hello” using the class SystemCommandTasklet after 5 seconds’ timeout:

```
<bean id="myTasklet”>
       <property name="tasklet">
            <bean class="org.springframework.batch.sample.tasklet.SystemCommandTasklet">
                   <property name="command" value="echo hello" />
                   <property name="timeout" value="5000" />
```

```
        </bean>
    </property>
</bean>
```

Example

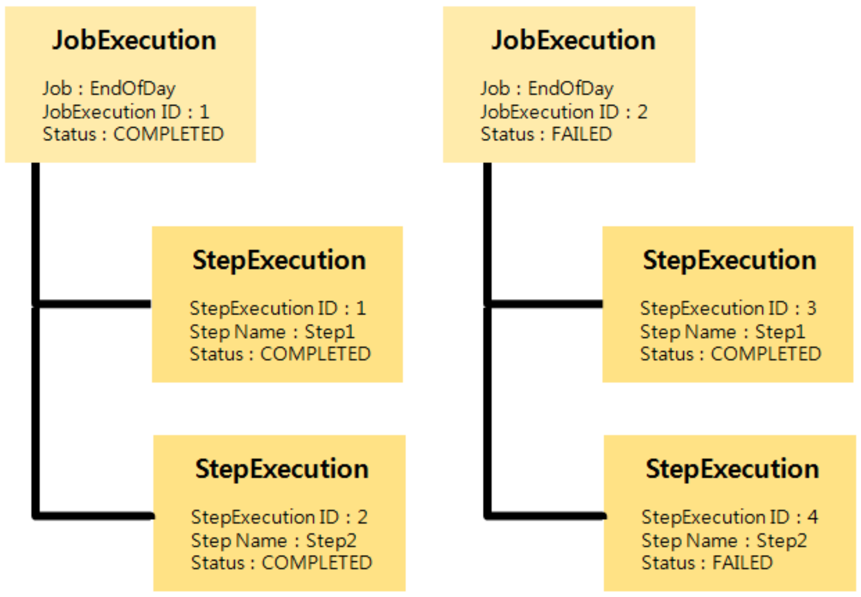
Simple Processing (Tasklet) Practice

StepExecution

Eqivalent to JobExecution, StepExecution signifies an attempt to execute step and is generated for every attempt over and over again. Serving as a repository mechanism, StepExecution reposits properties and status information available while execution is underway, such as commit count, rollback count, start time and end time. (Check out details of StepExecution properties)

Assuming ‘EndOfDay’ jobs were executed a couple times for 'Step1’ and 'Step2’ (a pair of JobExecution would be generated accordingly), whereby a total of four StepExecutions are to be generated.

✔ With a total of four StepExecutions concluded FAILED, a pair of JobExecutions are ended up FAILED accordingly (you’ll need to get all the Steps complete for conclusion of jobs comprising steps concluded).



See the following table that tabulates visualized information above:

StepExecution ID	Step Name	JobExecution ID	Status
1	Step1	1	COMPLETED
2	Step2	1	COMPLETED
3	Step1	2	COMPLETED
4	Step2	2	FAILED

Step Configuration

It's a developer's discretion to have the step configuration either simple or complex. In order to simplify the configuration, developers can take advantage of Spring's batch Namespace.

Configurations for Chunk-oriented Step

```
<job id="sampleJob" job-repository="jobRepository">
	<step id="step1">
		<tasklet transaction-manager="transactionManager">
			<chunk reader="itemReader" writer="itemWriter" commit-interval="10"/>
		<tasklet>
	</step>
</job>
```

- reader : ItemReader for batch processing
- writer : ItemWriter for the items read
- transaction-manager : Commences and commits transactions while batch processing is underway via Spring's PlatformTransactionManager ("transactionManger" is default. Default configuration can be omitted)
- job-repository : Reposition of StepExecution and ExecutionContext during batch operation on a regular basis (default configuration can be omitted for 'jobRepository')
- commit-interval : Number of items to be processed before commitment of transaction

✔ The attribute ItemProcessor is optional. Direct transfer from Reader to Writer takes place when ItemProcessor is deactivated.

Configuration using Inheritance

parent

You can wisely use the attribute 'parent' when a multitude of steps are available with similar configuration. Like class inheritance in Java, a child step combines its properties with those of the parent step and from time to time overrides properties of the parent step.
"concreteStep1" inherited "parentStep" is configured 'itemReader', 'itemProcessor', 'itemWriter', startLimit=5 and allowStartIfComplete=true, with commit-interval overridden by "5".

```
<step id="parentStep">
	<tasklet allow-start-if-complete="true">
		<chunk reader="itemReader" writer="itemWriter" commit-interval="10"/>
	</tasklet>
</step>

<step id="concreteStep1" parent="parentStep">
	<tasklet start-limit="5">
		<chunk processor="itemProcessor" commit-interval="5"/>
	</tasklet>
</step>
```

abstract

Abstract in eGovFramework is defined the totally same with that of Java, by which you'll from time to time need to define parent steps that does not comprise a complete step. In the attribute 'abstract' you can define whether the configuration Step falls under abstract level or not.
See the following example to see "abstractParentStep" is declared 'abstract' and the child step defining 'itemReader', 'itemWriter' and 'commitInterval':

```
<step id="abstractParentStep" abstract="true">
	<tasklet>
		<chunk commit-interval="10"/>
	</tasklet>
</step>

<step id="concreteStep2" parent="abstractParentStep">
	<tasklet>
		<chunk reader="itemReader" writer="itemWriter"/>
```

```
        </tasklet>
</step>
```

merge

An attribute defined in a child step the same way done for the parent step is to be overridden, save that you can use the attribute 'merge' for a listener defined by the parent step to be severally defined for the child step (you can use the child step in the any attribute regarding 'list', inclusive of <listeners>).
Refer to the following example for how the step "concreteStep3" uses both "listenerTwo" and "listenerOne".

```
<step id="listenersParentStep" abstract="true">
        <listeners>
                <listener ref="listenerOne"/>
        <listeners>
</step>

<step id="concreteStep3" parent="listenersParentStep">
        <tasklet>
                <chunk reader="itemReader" writer="itemWriter" commit-interval="5"/>
        </tasklet>
        <listeners merge="true">
        <listener ref="listenerTwo"/>
        <listeners>
</step>
```

Configurations for Restart Step

start-limit

You can manually configure execution of steps. Default execution is Integer.MAX_VALUE, as configured in the class SimpleStepFactoryBean.

allow-start-if-complete

You can manually configure execution of step "COMPLETED" when job is restarted. When configured "true", you can get you completed step executed over again to override the previous attempt. (Configure "false" to skip the completed steps)

See the following example for 10-time repetition of "step1" execution. When the job is restarted, it gets repeted regardless of the result of the previous attempt:

```
<step id="step1">
        <tasklet allow-start-if-complete="true" start-limit="10">
                <chunk reader="itemReader" writer="itemWriter" commit-interval="10"/>
        </tasklet>
</step>
```

Configuring Skip/Retry/Repeat

Skip

You can take a deeper look at configurations for Skip here

Retry

You can take a deeper look at configurations for Retry here

Step Flow Control

You can take a deeper look at configurations for Step Flow Control here

References

- http://static.springsource.org/spring-batch/reference/html/domain.html#domainStep
- http://static.springsource.org/spring-batch/reference/html/configureStep.html