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

Job

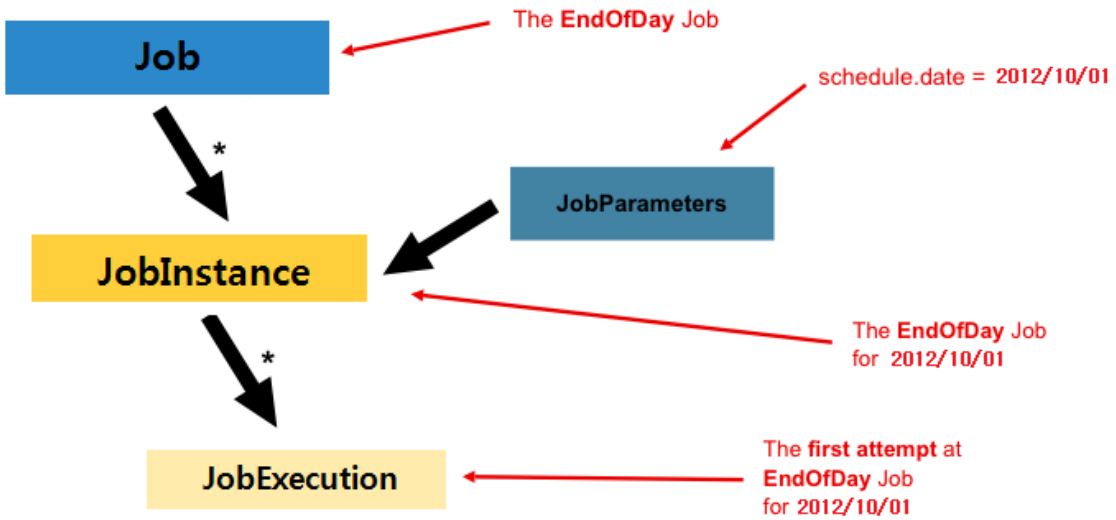
Outline

Referring to “batch job”, job is on the highest tier of the process steps. Execution of jobs thus constitutes execution of batch processing.

Description

- Again, jobs refers to “capsulated” batch jobs on the highest tier of the process steps.
- A JobInstance is generated and assigned to the corresponding job as per JobParameters. Further generated in JobExecution per job attempt.
- A job must comprise one or more Steps.

In the following figure you can find the job entitled ‘EndOfDay’, where the JobParameter of ‘2012/10/01’is assigned to geenrate a JobParameter. Also generated is JobExecution that signifies Attempt 1 of ‘EndOfDay’.

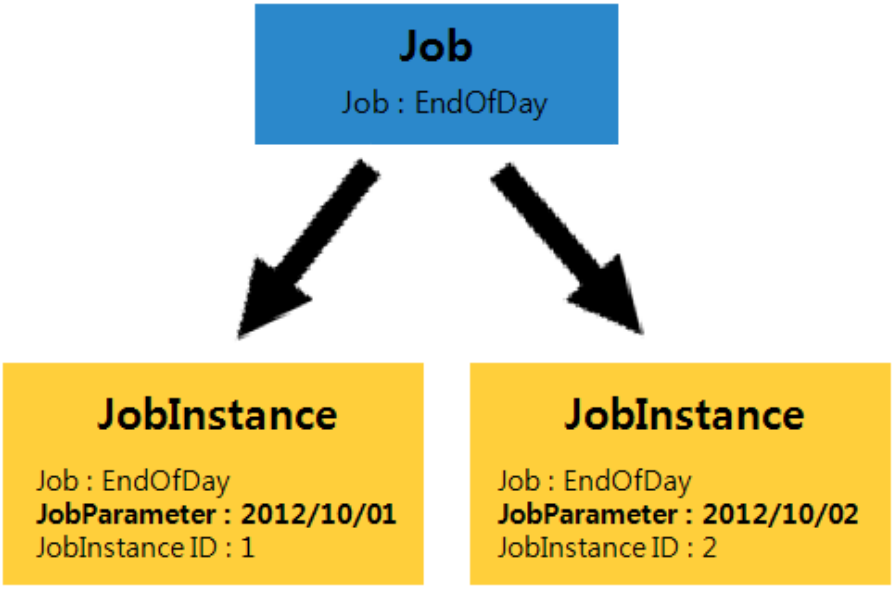


You can implement the job interface using the class SimpleJob capable of a variety of standard functions working in all sorts of jobs. One another way, you can also use the tag <job> as follows:

```
<job id="footballJob">
    <step id="playerload" next="gameLoad"/>
    <step id="gameLoad" next="playerSummarization"/>
    <step id="playerSummarization"/>
</job>
```

JobInstance

Being a notion of logical execution of jobs, **JobInstance = Job + JobParameters** is an execution unit of jobs that were executed by several jobParameters (A JobInstance has the identical Job and Job Parameter).



Assuming the job 'EndOfDay' executed on a daily basis, as provided in the foregoing picture, you need to understand that the very same 'EndOfDay' is executed each day, with different JobInstance assigned. For instance, the jobs 'EndOfDay' executed on '2012/10/01' and '2012/10/02' have different JobInstances from each other.

You can use JobInstance to restart jobs, meaning that the execution context of the concerned JobInstance is re-used whereby a new JobInstance is not generated (a brand-new JobExecution is generated.)

Refer to the following table for how a single job entitled 'EndOfDay' is categorized by JobParameter when JobInstances are generated for each job:

JobInstance ID	Job Name	JobParameters
1	EndOfDay	2012/10/01
2	EndOfDay	2012/10/02

JobParameters

Representing a set of Parameters intended to tell a multitude o JobInstances existing in a single Job, JobParameters are from time to time referred to identiy jobs being executed. See the foregoing picture (JobInstance) to find a single job 'EndOfDay' generating a couple of JobInstances generated for separate days ('2012/10/01', '2012/10/02').

Refer to the following table to discover JobParameters assigned separately to Job Instance.

JobInstance ID	JobParameter	Job Name
1	2012/10/01	EndOfDay
2	2012/10/02	EndOfDay

JobParameter Structures

- In JobParameter you've got the actual parameter contents and ParameterType.

public class JobParameter implements Serializable {

private final Object parameter;

```
        private final ParameterType parameterType;

}
```

- ParameterType receives String, Date, Long and Double in the form of enum.

```
public enum ParameterType {

        STRING, DATE, LONG, DOUBLE;
}
```

- Keep in mind JobParameters are manated in the form of Map, as follows:

```
public class JobParameters implements Serializable {

        private final Map<String,JobParameter> parameters;

        public JobParameters() {
                this.parameters = new LinkedHashMap<String, JobParameter>();
        }

        public JobParameters(Map<String,JobParameter> parameters) {
        this.parameters = new LinkedHashMap<String,JobParameter>(parameters);
        }
```

How to generate JobParameter

- Generating JobParemeter using the class JobParameterBuilder:

```
protected JobParameters getUniqueJobParameters() {
                        return new JobParametersBuilder(super.getUniqueJobParameters())
                        .addString("inputFile","data/iosample/input/delimited.csv")
                        .addString("outputFile","file:./target/test-outputs/delimitedOutput.csv").toJobParameters();
}
```

- Generating JobParemeter using the class DefaultJobParametersConverter:

```
new DefaultJobParametersConverter()
                        .getJobParameters(PropertiesConverter
        .stringToProperties("run.id(long)=1,parameter=true,run.date=20121001"));
```

JobParameters generated are utilizable in XML as follows:

- ex) FlatFileItemReader

You can input the Parameter 'inputFile' (key) in JobParameter in the resource as follows:

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" >
        <property name="resource" value="#{jobParameters[inputFile]}" />
                        (skipped...)
</bean>
```

- ex) FlatFileItemWriter

You can input the Parameter 'outputFile' (key) in JobParameter in the resource as follows:

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemWriter" >
        <property name="resource" value="#{jobParameters[outputFile]}" />
                        (skipped...)
```

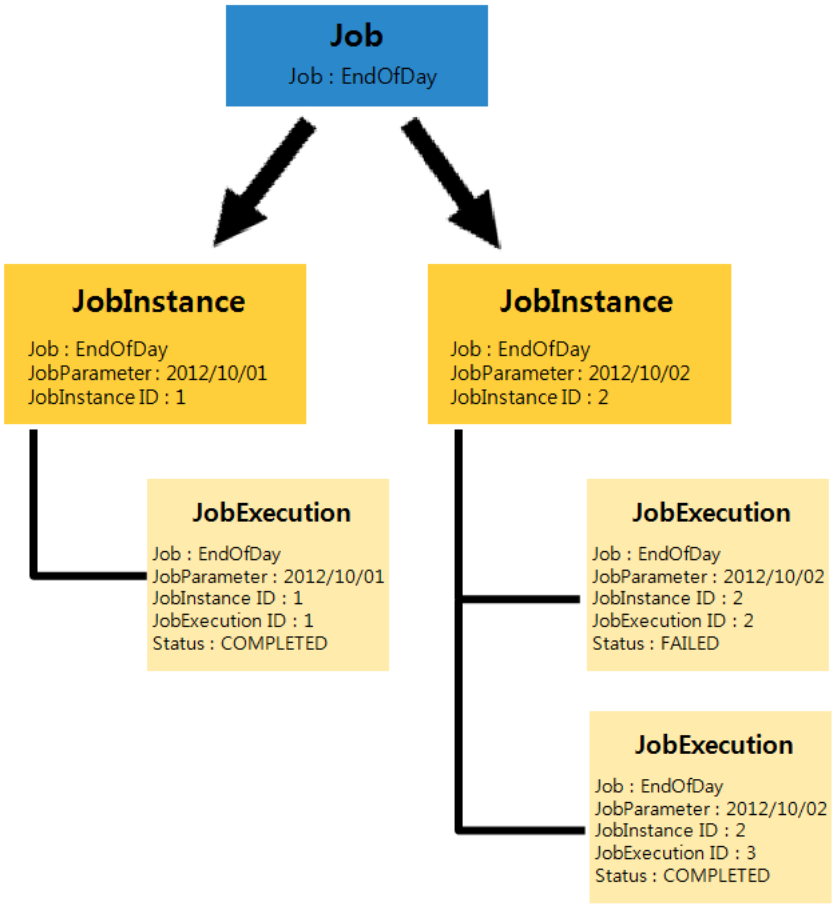
</bean>

JobExecution

JobExecution attempts an execution of job that is resulted either in 'FAILED' or 'COMPLETED' and serves as a reporitory mechanism for properties taken place during the course of job execution. (Click here to see details on JobExecution)

See the following figure to see an 'EndOfDay' Job has a pair of JobInstances with a total of three JobExecutions.
While JobInstances logically categorize jobs executed on a daily basis, JobExecution signifies that a total of three job attempts have completed.
With the job 'EndOfDay' executed in JobParameter entitled '2012/10/02', the JobInstances with the ID entitled '2' is executed. First of all, you'll need to end up FAILED for the Attempt 1 of Job, with the Attempt 2 of Job COMPLETED.
All in all, the Job 'EndOfDay' made a pair of job attempts on '2012/10/02' to generate a pair of JobExecution.

✔ Note that a JobExecution with COMPLETED status may not restart (the concerned JobInstance has completed normal process of batch).



Refer to the following table for a pair of JobInstances generated out of different JobParameters '2012/10/01' and '2012/10/02', of course reporesenting different 'EndOfDay' Job, with a total of three JobExecutions generated by way of three different attempts of job.
Notable here is that JobExecution has been generated every time job is attempted, with JobExecution with the samee JobInstance attempting jobs by way of the very same JobParameter.

JobExecution ID	JobInstance ID	Start Time	End Time	Status
1	1	2012.10.01.12:00	2012.10.01.12:10	COMPLETED

2	2	2012.10.02.12:00	2012.10.02.12:10	FAILED
3	2	2012.10.02.12:30	2012.10.02.12:40	COMPLETED

Job Configuration

In the typical Spring project, a job is represented in the form of and establishes dependency with XML configuration file referred to as “Job configuration”.

Mandatory Job Configuration

```
<job id="footballJob" job-repository="specialRepository">
        <step id="playerload" parent="s1" next="gameLoad"/>
        <step id="gameLoad" parent="s3" next="playerSummarization"/>
        <step id="playerSummarization" parent="s3"/>
</job>
```

- ID : Job identifier
- <step> : Job should define at least one or more steps.
- job-repository : Reposition of JobExecution during batch operation on a regular basis (default configuration can be omitted for 'jobRepository')

Configuration using Inheritance

parent

You can wisely use the attribute 'parent’ when a multitude of jobs are available with similar configuration. Like class inheritance in Java, a child job combines its properties with those of the parent job and from time to time overrides properties of the parent job.

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 jobs that does not comprise a complete job. In the attribute 'abstract’ you can define whether the configuration Job falls under abstract level or not.

Refer to the following example to see “baseJob” is declared to be abstract, with “job1” being a child job defining <step> to configure a job. The foregoing “job1” inherited “listenerOne” from “baseJob”:

```
<job id="baseJob" abstract="true">
        <listeners>
                <listener ref="listenerOne"/>
        <listeners>
</job>

<job id="job1" parent="baseJob">
        <step id="step1" parent="standaloneStep"/>
        <listeners merge="true">
                <listener ref="listenerTwo"/>
        <listeners>
</job>
```

Restartability Configuration

restartable

A job can restart JobInstance that is incomplete. When configuring jobs, you need to configure the attribute ‘restartable’ for you to have JobInstances restarted.

```
<job id="footballJob" restartable="false">
    ...
</job>
```

References

- http://static.springsource.org/spring-batch/reference/html/domain.html#domainJob
- http://static.springsource.org/spring-batch/reference/html/configureJob.html#configuringAJob