Spring Batch : Framework for processing the application data in chunk processing manner

Provide infrastructure component that Supports hanlding the application failure/transaction/scheduling.

===================================================================================================================

Component Description

===================================================================================================================

Job repository An infrastructure component that persists job execution metadata

Job launcher An infrastructure component that starts job executions

Job An application component that represents a batch process

Step A phase in a job; a job is a sequence of steps

Tasklet A transactional, potentially repeatable process occurring in a step

Item A record read from or written to a data source

Chunk A list of items of a given size

Item reader A component responsible for reading items from a data source

Item processor A component responsible for processing (transforming, validating, or filtering) a read item before it’s written

Item writer A component responsible for writing a chunk to a data source

Job instance(not component) A specific run of a job

Job execution(not component) The execution of a job instance (with success or failure)

Job repository::::

The job repository maintains all metadata related to job executions in meta data table. (all these table will get automatically created, if configured. or controlled by configuration)

Tables (Meta-Data Schema) :

BATCH\_JOB\_INSTANCE - The BATCH\_JOB\_INSTANCE table holds all information relevant to a JobInstance, and serves as the top of the overall hierarchy.

(jon instance id, job name, etc)

BATCH\_JOB\_EXECUTION - table holds all information relevant to the JobExecution object.Every time a Job is run

there will always be a new and a new row in this table

(Start time, end time, status, exit code, exit message)

BATCH\_JOB\_EXECUTION\_PARAMS - holds all information relevant to the JobParameters object.

(Start time, end time, status, exit code, exit message)

BATCH\_JOB\_EXECUTION\_CONTEXT - table holds all information relevant to an Job's ExecutionContext.

There is exactly one Job ExecutionContext per JobExecution, and

it contains all of the job-level data that is needed for a particular job execution.

BATCH\_STEP\_EXECUTION\_CONTEXT - table holds all information relevant to the StepExecution object. This

Job Launcher::::

Used to launch in 2 modes

1. Sync

2. Async (using task executor)

Types of Launcher :

CommandLineJobRunner

JobLauncer - web launcher

Job ::::

Job is a collection of steps & tasklets.

Listener : can run before & after completion

restartable or not.

==================================================== SAMPLE JOB CONFIGURATION ========================================================

<task:executor id="executor" pool-size="10" />

<bean id="jobLauncher" class="org.springframework.batch.core.launch.support.SimpleJobLauncher">

<property name="jobRepository" ref="jobRepository" />

<property name="taskExecutor" ref="executor" />

</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="${batch.jdbc.driver}" />

<property name="url" value="${batch.jdbc.url}" />

<property name="username" value="${batch.jdbc.user}" />

<property name="password" value="${batch.jdbc.password}" />

</bean>

<bean id="transactionManager" lazy-init="true" class="org.springframework.jdbc.datasourceDataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

<batch:job-repository id="jobRepository" data-source="dataSource" transaction-manager="transactionManager" />

<batch:job id="importProductsJob" job-repository="jobRepository">

(...)

<batch:step id="readWrite">

<batch:tasklet>

<batch:chunk

(...)

skip-limit="20"

retry-limit="3"

cache-capacity="100"

chunk-completion-policy="timeoutCompletionPolicy"/>

</batch:tasklet>

</batch:step>

</batch:job>

=============================================================================================================================================

Step:::

A step is a phase in a job;

Steps define the sequence of actions a job will perform, one at a time.

TASKLET:::

A tasklet corresponds to a transactional, potentially repeatable process occurring in a step.

Custom tasklet : (implements Tasklet , public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception) )

<job id="importProductsJob">

<step id="decompress" next="readWrite">

<tasklet ref="decompressTasklet" />

</step>

</job>

CHUNK-ORIENTED TASKLET:::

Process data in chunks: the ChunkOriented-Tasklet.

reader Bean identifier used to read data from a chunk. The bean must implement the Spring Batch ItemReader interface.

processor Bean identifier used to process data from a chunk. The bean must implement the Spring Batch ItemProcessor interface.

writer Bean identifier used to write data from a chunk. The bean must implement the Spring Batch ItemWriter interface.

commit-interval Number of items to process before issuing a commit. When the number of items read reaches the commit interval number, the entire corresponding chunk is written out through the item writer and the transaction is committed.

skip-limit Maximum number of skips during processing of the step.

TASKLET VS CHUNK

Tasklets are meant to perform a single task within a step.

Chunk - performs actions over chunks of data. That is, instead of reading, processing and writing all the lines at once, it’ll read, process and write a fixed amount of records (chunk) at a time.

Chunk size VS commit Interval

Page Size (Chunk Size) - The page-size attribute on the paging ItemReader implementations (JdbcPagingItemReader for example) defines how many records are fetched per read of the underlying resource.

The commit-interval defines how many items are processed within a single chunk. That number of items are read, processed, then written within the scope of a single transaction (skip/retry semantics not withstanding).

Reader : impletemented by ItemReader

FlatFileItemReader (consists of LineMapper, LineTokenizer, FieldSetMapper () )

JdbcCursorItemReaderm / JdbcPagingItemReader

StoredProcedureItemReader

<bean i/d="fixedLengthLineTokenizer" class="org.springframework.batch.io.file.transform.FixedLengthTokenizer">

<property name="names" value="ISIN,Quantity,Price,Customer" />

<property name="columns" value="1-12, 13-15, 16-20, 21-29" />

</bean>

Writer : Implemented by ItemWriter Interface

FlatFileItemWriter

JdbcBatchItemWriter

Process : ItemProcessor Interface

FilteringProductItemProcessor

Custom processor - implpement ItemProcessor

=================================================================================================

Listener type Description

=================================================================================================

Job listener Listens to processing at the job level (beforeJob , afterJob)

Step listeners Listens to processing at the step level (beforeStep , afterStep)

Item listeners Listens to item repeat or retry

=================================================================================================

Master & Slave Concept with partitioner.

<job id="partitionJob" xmlns="http://www.springframework.org/schema/batch">

<!-- master step, 10 threads (grid-size) -->

<step id="masterStep">

<partition step="slave" partitioner="rangePartitioner">

<handler grid-size="10" task-executor="taskExecutor" />

</partition>

</step>

</job>

<!-- each thread will run this job, with different stepExecutionContext values. -->

<step id="slave" xmlns="http://www.springframework.org/schema/batch">

<tasklet>

<chunk reader="pagingItemReader" writer="flatFileItemWriter"

processor="itemProcessor" commit-interval="1" />

</tasklet>

</step>

<bean id="rangePartitioner" class="com.partition.RangePartitioner" /> ( implements Partitioner => Map<String, ExecutionContext> partition(int gridSize))

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

<!-- inject stepExecutionContext -->

<bean id="itemProcessor" class="com.processor.UserProcessor" scope="step">

<property name="threadName" value="#{stepExecutionContext[name]}" />

</bean>

<bean id="pagingItemReader"

class="org.springframework.batch.item.database.JdbcPagingItemReader"

scope="step">

<property name="dataSource" ref="dataSource" />

<property name="queryProvider">

<bean

class="org.springframework.batch.item.database.support.SqlPagingQueryProviderFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="selectClause" value="select id, user\_login, user\_pass, age" />

<property name="fromClause" value="from users" />

<property name="whereClause" value="where id >= :fromId and id <= :toId" />

<property name="sortKey" value="id" />

</bean>

</property>

<!-- Inject via the ExecutionContext in rangePartitioner -->

<property name="parameterValues">

<map>

<entry key="fromId" value="#{stepExecutionContext[fromId]}" />

<entry key="toId" value="#{stepExecutionContext[toId]}" />

</map>

</property>

<property name="pageSize" value="10" />

<property name="rowMapper">

<bean class="com.UserRowMapper" />

</property>

</bean>

<!-- csv file writer -->

<bean id="flatFileItemWriter" class="org.springframework.batch.item.file.FlatFileItemWriter"

scope="step" >

<property name="resource"

value="file:csv/outputs/users.processed#{stepExecutionContext[fromId]}-#{stepExecutionContext[toId]}.csv" />

<property name="appendAllowed" value="false" />

<property name="lineAggregator">

<bean

class="org.springframework.batch.item.file.transform.DelimitedLineAggregator">

<property name="delimiter" value="," />

<property name="fieldExtractor">

<bean

class="org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor">

<property name="names" value="id, username, password, age" />

</bean>

</property>

</bean>

</property>

</bean>

</beans>