Simple ETL with Spring Batch

TK Java, Spring framework April 23, 2019

Spring Batch (https://spring.io/projects/spring-batch) is literally a batch framework based on Spring Framework. I usually use it to develop a simple ETL(Extraction, Transaformation and Loading) program.

In this post, I'll show you how to write a simple ETL program. (This sample is tested on Spring Batch 3.0.10)

Prerequisites

- Database (MySQL or Oracle)
- Spring batch context database
- Spring batch libraries (spring-batch-core, spring framework, spring-jdbc)
- Database JDBC driver
- DBCP (Optional)

Spring batch context database

Spring batch context database must be created to run Spring batch job. Table creation DDL can be found on spring-batch-core-version.jar (org.springframework.batch.core package contains DDL for several databases)

It contains the following tables.

- BATCH_JOB_INSTANCE
- BATCH_JOB_EXECUTION
- BATCH_JOB_EXECUTION_PARAMS
- BATCH_STEP_EXECUTION
- BATCH_STEP_EXECUTION_CONTEXT
- BATCH_JOB_EXECUTION_CONTEXT

Test scenario

In this post, I use two tables : TB_SOURCE, TB_TARGET. The sample program reads from TB_SOURCE and writes all data into TB_TARGET.

```
1  CREATE TABLE TB_SOURCE(
2  col1 VARCHAR2(10),
3  col2 VARCHAR2(20),
4  col3 VARCHAR2(20)
5  );
6
7  CREATE TABLE TB_TARGET(
8  new_col1 VARCHAR2(10),
9  new_col2 VARCHAR2(20),
10 new_col3 VARCHAR2(20)
11 );
```

view raw test db schema hosted with ❤ by **GitHub**

Writing base Spring context

To run a Spring batch program, some beans need to be declared.

- Spring Batch Context DataSource
- DataSource for source and target
- TransactionManager for Spring Batch Context
- TransactionManager for source and target
- Job Repository bean
- Job Launcher (which is the starting point of a job)

The following is a snippet of context.xml.

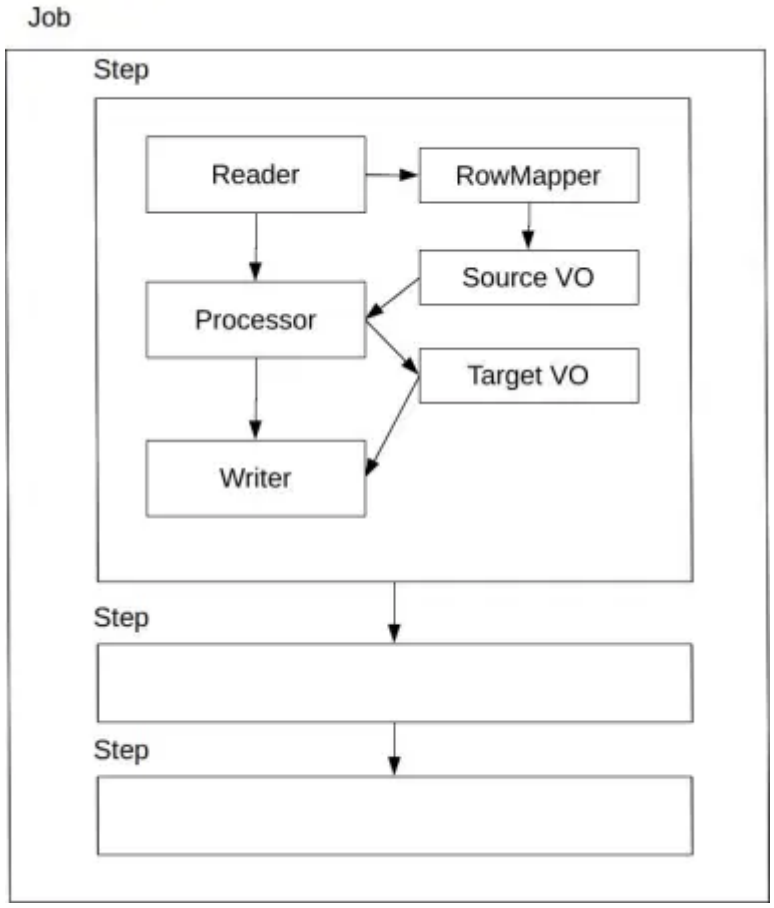
```
1   <bean id="repositoryDataSource"
2       class="org.apache.commons.dbcp.BasicDataSource"
3       destroy-method="close">
4       <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
5       <property name="url" value="jdbc:oracle:thin:@guest01:1521:myora" />
6       <property name="username" value="scott" />
7       <property name="password" value="tiger" />
8   <property name="initialSize" value="2" />
9   <property name="minIdle" value="2" />
10  <property name="maxActive" value="10" />
11  <property name="maxIdle" value="5" />
12  </bean>
13
14  <bean id="testDataSource"
15      class="org.apache.commons.dbcp.BasicDataSource"
16      destroy-method="close">
17      <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
```

```
18          <property name="url" value="jdbc:oracle:thin:@guest01:1521:myora" />
19          <property name="username" value="scott" />
20          <property name="password" value="tiger" />
21      <property name="initialSize" value="2" />
22      <property name="minIdle" value="2" />
23      <property name="maxActive" value="10" />
24      <property name="maxIdle" value="5" />
25      </bean>
26
27      <bean id="repositoryTransactionManager" lazy-init="true"
28          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
29          <property name="dataSource" ref="repositoryDataSource" />
30      </bean>
31
32      <bean id="testTransactionManager" lazy-init="true"
33          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
34          <property name="dataSource" ref="testDataSource" />
35      </bean>
36
37      <batch:job-repository id="jobRepository"
38          data-source="repositoryDataSource"
39          transaction-manager="repositoryTransactionManager"
40          isolation-level-for-create="READ_COMMITTED"
41          table-prefix="BATCH_"
42      />
43
44      <bean id="jobLauncher" class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
45          <property name="jobRepository" ref="jobRepository" />
46      </bean>
```

view raw spring_batch_test.basic_spring_context hosted with ❤ by **GitHub**

Writing Job flow

Now, it's ready to write a ETL program. A job is also declared in spring context.xml. The following is the basic structure of a Job.



- A job can have several Steps (for simplicity, I use one Step for this sample)
- Each Step has a Reader, a Processor and a Writer
- Reader reads data from database, file or some data store
- Reader invokes RowMapper to convert raw data into Source VO

- Processor reads a Source VO and convert it into Target VO
- Writer writes Target VO into database, file or some data store

The following is a sample ETL job definition.

```
1   <batch:job id="TestJob01" job-repository="jobRepository">
2       <batch:step id="SimpleStep">
3           <batch:tasklet transaction-manager="testTransactionManager" allow-start-if-complete="true">
4               <batch:chunk
5                   reader="TestReader"
6                   processor="TestProcessor"
7                   writer="TestWriter"
8                   commit-interval="1"
9                   reader-transactional-queue="false"
10              />
11          </batch:tasklet>
12      </batch:step>
13  </batch:job>
14
15  <bean id="SourceMapper"
16      class="test.rowmapper.TbSourceMapper"/>
17
18  <bean id="TestReader"
19      class="org.springframework.batch.item.database.JdbcCursorItemReader"
20  scope="step">
21      <property name="dataSource" ref="testDataSource"/>
22      <property name="sql"
23          value="SELECT COL1, COL2, COL3
24              FROM TB_SOURCE"/>
25      <property name="rowMapper"
26          ref="SourceMapper"/>
27      <property name="fetchSize" value="100" />
```

```
28          <property name="maxRows" value="0" />
29      </bean>
30
31      <bean id="TestProcessor"
32      class="test.processor.TestProcessor"
33      scope="step">
34      </bean>
35
36      <bean id="TestWriter"
37          class="org.springframework.batch.item.database.JdbcBatchItemWriter"
38      scope="step">
39          <property name="assertUpdates" value="true" />
40          <property name="itemSqlParameterSourceProvider">
41              <bean class="org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourceProvider" />
42          </property>
43          <property name="sql"
44              value="INSERT INTO TB_TARGET(NEW_COL1, NEW_COL2, NEW_COL3)
45              VALUES(:newCol1, :newCol2, :newCol3)"
46           />
47          <property name="dataSource" ref="testDataSource" />
48      </bean>
```

view raw spring_batch_test.sample_job hosted with ❤ by **GitHub**

As the above sample shows, if a Reader or a Writer's target is database, SQL can be used directly. (It's the strength of Spring batch)

Writing RowMapper

RowMapper is a component which converts raw source data into Source VO. It must implement org.springframework.jdbc.core.RowMapper. The sample RowMapper is as follows.

```
1  import java.sql.ResultSet;
2  import java.sql.SQLException;
3  import org.springframework.jdbc.core.RowMapper;
4  import test.vo.SourceVO;
5
6  public class TbSourceMapper implements RowMapper<SourceVO> {
7
8      @Override
9      public SourceVO mapRow(ResultSet rs, int rowNum) throws SQLException {
10         SourceVO vo = new SourceVO();
11         vo.col1 = rs.getString(1);
12         vo.col2 = rs.getString(2);
13         vo.col3 = rs.getString(3);
14         return vo;
15     }
16
17 }
```

view raw spring_batch_test.sample_row_mapper hosted with ❤ by **GitHub**

Writing Processor

Processor is the core of Spring Batch. It can transform source data, verify it or execute any additional logic. In this sample, it simply maps source to another column name. The sample Processor is as follows.

```
1  import org.springframework.batch.item.ItemProcessor;
2  import test.vo.SourceVO;
3  import test.vo.TargetVO;
4
5  public class TestProcessor implements ItemProcessor<SourceVO, TargetVO> {
```

```
6
7       @Override
8       public TargetVO process(SourceVO item) throws Exception {
9           TargetVO targetVO = new TargetVO();
10          targetVO.new_col1 = item.col1;
11          targetVO.new_col2 = item.col2;
12          targetVO.new_col3 = item.col3;
13          return targetVO;
14      }
15  }
```

view raw spring_batch_test.sample_processor hosted with ❤ by **GitHub**

Writing Source VO

```
1  public class SourceVO {
2      public String col1;
3      public String col2;
4      public String col3;
5  }
```

view raw spring_batch_test.sample_source_vo hosted with ❤ by **GitHub**

Writing Target VO

Target VO must have getter method in this sample.

```
1  public class TargetVO {
2      public String new_col1;
```

```
3       public String new_col2;
4       public String new_col3;
5
6       public String getNewCol1() {
7           return this.new_col1;
8       }
9
10      public String getNewCol2() {
11          return this.new_col2;
12      }
13
14      public String getNewCol3() {
15          return this.new_col3;
16      }
17  }
```

view raw spring_batch_test.sample_target_vo hosted with ❤ by **GitHub**

And there is no more components to write except a program which invokes this sample Job.

You can download the full sources from github (https://github.com/tkstone/spring_batch_sample01.git).

Tagged:
spring batch,
Spring framework

Published by TK

View all posts by TK

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Blog at WordPress.com.