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

CompositItem Examples

Outline

Spring provides the user with CompositeItemProcessor and CompositeItemWriter for composite item processing. You may also use CompositeItemWriter to proceed with the different types of writer to cause the job to be processed.

Description

Settings

Configuring Jobs

Check out compositeItemWriterSampleJob.xml, the job configuration file for the CompositeItem example.

CompositeItemWriter causes the item to be transferred to the writer defined as Delegate. See the following example where CompositeItemWriter comprises a total of three writers being tradeDao, fileItemWriter1 and fileItemWriter2.

✔When using CompositeItemWriter, you need to re-define the writers defined as Delegate in Step as Streams.

```
<job id="compositeItemWriterJob" xmlns="http://www.springframework.org/schema/batch" >
	<step id="compositeItemWriterStep1">
		<tasklet>
			<chunk reader="fileItemReader" processor="processor" writer="compositeWriter"
commit-interval="1">
				<streams>
					<stream ref="fileItemReader"/>
					<stream ref="fileItemWriter1"/>
					<stream ref="fileItemWriter2"/>
				</streams>
			</chunk>
		</tasklet>
	</step>
</job>

<bean id="compositeWriter" class="org.springframework.batch.item.support.CompositeItemWriter">
	<property name="delegates">
		<list>
			<bean class="egovframework.brte.sample.common.domain.trade.TradeWriter">
				<property name="dao" ref="tradeDao" />
			</bean>
			<ref bean="fileItemWriter1" />
			<ref bean="fileItemWriter2" />
		</list>
	</property>
	</bean>
<bean id="tradeDao" class="egovframework.brte.sample.common.domain.trade.JdbcTradeDao">
	<property name="dataSource" ref="dataSource" />
	<property name="incrementer">
		<bean parent="incrementerParent">
			<property name="incrementerName" value="TRADE_SEQ" />
		</bean>
	</property>
</bean>
<bean id="fileItemWriter1" class="org.springframework.batch.item.file.FlatFileItemWriter">
```

```
        <property name="name" value="fw1" />
        <property name="resource" value="file:target/test-outputs/CustomerReport1.txt" />
        <property name="lineAggregator">
                <bean class="org.springframework.batch.item.file.transform.PassThroughLineAggregator" />
        </property>
</bean>

<bean id="fileItemWriter2" class="org.springframework.batch.item.file.FlatFileItemWriter">
        <property name="name" value="fw2" />
        <property name="resource" value="file:target/test-outputs/CustomerReport2.txt" />
        <property name="lineAggregator">
                <bean class="org.springframework.batch.item.file.transform.PassThroughLineAggregator" />
        </property>
</bean>
```

Composition and Implementation of JunitTest

Composition of JunitTest

Work Junit Test that comprises CompositeItem configuration and relevant classes, where batch implementation is involved and the associated contents are viewed.

✔ See Junit Test Description for Batch Execution Environment for structure of the Class JunitTest.
✔ assertEquals("COMPLETED", jobExecution.getExitStatus().getExitCode()) : Make sure you check the batch execution result is COMPLETED.
✔ checkOutputFile,checkOutputTable : Check if the writers configured in compositeItemWriter have executed the batches.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/egovframework/batch/simple-job-launcher-context.xml",
"/egovframework/batch/jobs/compositeItemWriterSampleJob.xml",
                                        "/egovframework/batch/job-runner-context.xml" })
public class EgovCompositeItemWriterSampleFunctionalTests {

        // The queries loaded by the date of the table Trade.
        private static final String GET_TRADES = "SELECT isin, quantity, price, customer FROM TRADE order
by isin";

        // Result expected out of the files.
        private static final String EXPECTED_OUTPUT_FILE = "Trade:
[isin=UK21341EAH41,quantity=211,price=31.11,customer=customer1]"
                                + "Trade: [isin=UK21341EAH42,quantity=212,price=32.11,customer=customer2]"
                                + "Trade: [isin=UK21341EAH43,quantity=213,price=33.11,customer=customer3]"
                                + "Trade: [isin=UK21341EAH44,quantity=214,price=34.11,customer=customer4]"
                                + "Trade: [isin=UK21341EAH45,quantity=215,price=35.11,customer=customer5]";
        ...

        @Test
        public void testJobLaunch() throws Exception {


                int before = simpleJdbcTemplate.queryForInt("SELECT COUNT(*) from TRADE");

                JobExecution jobExecution =jobLauncherTestUtils.launchJob();
                assertEquals(BatchStatus.COMPLETED, jobExecution.getStatus());
                checkOutputFile("target/test-outputs/CustomerReport1.txt");
                checkOutputFile("target/test-outputs/CustomerReport2.txt");
                checkOutputTable(before);

        }
```

```
	/**
	 * Make sure you check the data provided in the table Trade after batching.
	 * Number of data in the table Trade before batching in @param.
	 */
	private void checkOutputTable(int before) {
		final List<Trade> trades = new ArrayList<Trade>() {
			{
				add(new Trade("UK21341EAH41", 211, new BigDecimal("31.11"),
"customer1"));
				add(new Trade("UK21341EAH42", 212, new BigDecimal("32.11"),
"customer2"));
				add(new Trade("UK21341EAH43", 213, new BigDecimal("33.11"),
"customer3"));
				add(new Trade("UK21341EAH44", 214, new BigDecimal("34.11"),
"customer4"));
				add(new Trade("UK21341EAH45", 215, new BigDecimal("35.11"),
"customer5"));
			}
		};

		int after = simpleJdbcTemplate.queryForInt("SELECT COUNT(*) from TRADE");

		assertEquals(before + 5, after);


		simpleJdbcTemplate.getJdbcOperations().query(GET_TRADES, new RowCallbackHandler() {
			private int activeRow = 0;
			public void processRow(ResultSet rs) throws SQLException {
				Trade trade = trades.get(activeRow++);

				assertEquals(trade.getIsin(), rs.getString(1));
				assertEquals(trade.getQuantity(), rs.getLong(2));
				assertEquals(trade.getPrice(), rs.getBigDecimal(3));
				assertEquals(trade.getCustomer(), rs.getString(4));
			}
		});

	}
	/**
	 * Make sure you check the data provided in the concerned file after batching.
	 Parallel Examples
	 * @throws IOException
	 */
	private void checkOutputFile(String fileName) throws IOException {
		@SuppressWarnings("unchecked")
		List<String> outputLines = IOUtils.readLines(new FileInputStream(fileName));

		String output = "";
		for (String line : outputLines) {
			output += line;
		}

		assertEquals(EXPECTED_OUTPUT_FILE, output);
	}

}
```

Implementation of JunitTest

See Implementation of JunitTest for more information.

Verification of Result

Make sure the table Trade and files are generated as a result of the foregoing test.

Status | Result1

	ID	VERSION	ISIN	QUANTITY	PRICE	CUSTOMER
1	21	0	UK21341EAH41	211	31.11	customer1
2	22	0	UK21341EAH42	212	32.11	customer2
3	23	0	UK21341EAH43	213	33.11	customer3
4	24	0	UK21341EAH44	214	34.11	customer4
5	25	0	UK21341EAH45	215	35.11	customer5

CustomerReport1.txt | CustomerReport2.txt

```
Trade: [isin=UK21341EAH41,quantity=211,price=31.11,customer=customer1]
Trade: [isin=UK21341EAH42,quantity=212,price=32.11,customer=customer2]
Trade: [isin=UK21341EAH43,quantity=213,price=33.11,customer=customer3]
Trade: [isin=UK21341EAH44,quantity=214,price=34.11,customer=customer4]
Trade: [isin=UK21341EAH45,quantity=215,price=35.11,customer=customer5]
```

References

- multidata_process
- http://static.springsource.org/spring-batch/reference/html/readersAndWriters.html#delegatePatternAndRegistering