SpringBatch

Spring Batch Component

The **spring-batch:** component and support classes provide integration bridge between Camel and Spring Batch infrastructure.

Maven users will need to add the following dependency to their `pom.xml` for this component:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spring-batch</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

URI format

```
spring-batch:jobName[?options]
```

Where **jobName** represents the name of the Spring Batch job located in the Camel registry.

This component can only be used to define producer endpoints, which means that you cannot use the Spring Batch component in a `from()` statement.

Options

Name	Default Value	Description
`jobLauncherRef`	`null`	**Deprecated and will be removed in Camel 3.0! Camel 2.10:** Use jobLauncher=#theName instead.
`jobLauncher`	`null`	**Camel 2.11.1:** Explicitly specifies a `JobLauncher` to be used from the Camel Registry.
`jobFromHeader`	`false`	**Camel 2.18:** Explicitly defines if the jobName shouls be taken from the headers instead of the URI. The header has name: `CamelSpringBatchJobName`

Usage

When Spring Batch component receives the message, it triggers the job execution. The job will be executed using the `org.springframework.batch.core.launch.JobLaucher` instance resolved according to the following algorithm:

- if `JobLauncher` is manually set on the component, then use it.
- if `jobLauncherRef` option is set on the component, then search Camel Registry for the `JobLauncher` with the given name. **Deprecated and will be removed in Camel 3.0!**
- if there is `JobLauncher` registered in the Camel Registry under **jobLauncher** name, then use it.
- if none of the steps above allow to resolve the `JobLauncher` and there is exactly one `JobLauncher` instance in the Camel Registry, then use it.

All headers found in the message are passed to the `JobLauncher` as job parameters. `String`, `Long`, `Double` and `java.util.Date` values are copied to the `org.springframework.batch.core.JobParametersBuilder` - other data types are converted to Strings.

Examples

Triggering the Spring Batch job execution:

```
from("direct:startBatch").to("spring-batch:myJob");
```

Triggering the Spring Batch job execution with the `JobLauncher` set explicitly.

```
from("direct:startBatch").to("spring-batch:myJob?jobLauncherRef=myJobLauncher");
```

Starting from the Camel **2.11.1** `JobExecution` instance returned by the `JobLauncher` is forwarded by the `SpringBatchProducer` as the output message. You can use the `JobExecution` instance to perform some operations using the Spring Batch API directly.

```
from("direct:startBatch").to("spring-batch:myJob").to("mock:JobExecutions");
...
MockEndpoint mockEndpoint = ...;
JobExecution jobExecution = mockEndpoint.getExchanges().get(0).getIn().getBody(JobExecution.class);
BatchStatus currentJobStatus = jobExecution.getStatus();
```

Support classes

Apart from the Component, Camel Spring Batch provides also support classes, which can be used to hook into Spring Batch infrastructure.

CamelItemReader

`CamelItemReader` can be used to read batch data directly from the Camel infrastructure.

For example the snippet below configures Spring Batch to read data from JMS queue.

```
<bean id="camelReader" class="org.apache.camel.component.spring.batch.support.CamelItemReader">
  <constructor-arg ref="consumerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="camelReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

CamelItemWriter

`CamelItemWriter` has similar purpose as `CamelItemReader`, but it is dedicated to write chunk of the processed data.

For example the snippet below configures Spring Batch to read data from JMS queue.

```
<bean id="camelwriter" class="org.apache.camel.component.spring.batch.support.CamelItemWriter">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="camelwriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

CamelItemProcessor

`CamelItemProcessor` is the implementation of Spring Batch `org.springframework.batch.item.ItemProcessor` interface. The latter implementation relays on Request Reply pattern to delegate the processing of the batch item to the Camel infrastructure. The item to process is sent to the Camel endpoint as the body of the message.

For example the snippet below performs simple processing of the batch item using the Direct endpoint and the Simple expression language .

```
<camel:camelContext>
  <camel:route>
    <camel:from uri="direct:processor"/>
    <camel:setExchangePattern pattern="InOut"/>
    <camel:setBody>
      <camel:simple>Processed ${body}</camel:simple>
    </camel:setBody>
  </camel:route>
</camel:camelContext>

<bean id="camelProcessor" class="org.apache.camel.component.spring.batch.support.CamelItemProcessor">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="direct:processor"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" processor="camelProcessor" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

CamelJobExecutionListener

`CamelJobExecutionListener` is the implementation of the `org.springframework.batch.core.JobExecutionListener` interface sending job execution events to the Camel endpoint.

The `org.springframework.batch.core.JobExecution` instance produced by the Spring Batch is sent as a body of the message. To distinguish between before- and after-callbacks `SPRING_BATCH_JOB_EVENT_TYPE` header is set to the `BEFORE` or `AFTER` value.

The example snippet below sends Spring Batch job execution events to the JMS queue.

```
<bean id="camelJobExecutionListener" class="org.apache.camel.component.spring.batch.support.
CamelJobExecutionListener">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:batchEventsBus"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
  <batch:listeners>
    <batch:listener ref="camelJobExecutionListener"/>
  </batch:listeners>
</batch:job>
```