Idempotent Consumer

Overview

The *idempotent consumer* pattern is used to filter out duplicate messages. For example, consider a scenario where the connection between a messaging system and a consumer endpoint is abruptly lost due to some fault in the system. If the messaging system was in the middle of transmitting a message, it might be unclear whether or not the consumer received the last message. To improve delivery reliability, the messaging system might decide to redeliver such messages as soon as the connection is re-established. Unfortunately, this entails the risk that the consumer might receive duplicate messages and, in some cases, the effect of duplicating a message may have undesirable consequences (such as debiting a sum of money twice from your account). In this scenario, an idempotent consumer could be used to weed out undesired duplicates from the message stream.

Idempotent consumer with in-memory cache

In Mediation Router, the idempotent consumer pattern is implemented by the `idempotentConsumer()` processor, which takes two arguments:

- `messageIdExpression`—an expression that returns a message ID string for the current message; and
- `messageIdRepository`—a reference to a message ID repository, which stores the IDs of the messages received so far.

As each message comes in, the idempotent consumer processor looks up the current message ID in the repository to see if this message has been seen before. If yes, the message is discarded; if no, the message is allowed to pass and its ID is added to the repository.

For example, the following example uses the `TransactionID` header to filter out duplicates:

```
import static org.apache.camel.processor.idempotent.MemoryMes
sageIdRepository.memoryMessageIdRepository;
...
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("seda:a")
          .idempotentConsumer(
             header("TransactionID"),
             memoryMessageIdRepository(200)
          ).to("seda:b");
```

```
    }
};
```

Where the call to `memoryMessageIdRepository(200)` creates an in-memory cache that can hold up to 200 message IDs.

You can also define an idempotent consumer using XML configuration. For example, you can define the preceding route in XML, as follows:

```
<camelContext id="buildIdempotentConsumer" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="seda:a"/>
    <idempotentConsumer messageIdRepositoryRef="MsgIDRepos">
      <simple>header.TransactionID</simple>
      <to uri="seda:b"/>
    </idempotentConsumer>
  </route>
</camelContext>

<bean id="MsgIDRepos" class="org.apache.camel.processor.idem
potent.MemoryMessageIdRepository">
    <!-- Specify the in-memory cache size. -->
    <constructor-arg type="int" value="200"/>
</bean>
```

Idempotent consumer with JPA repository

The in-memory cache suffers from the disadvantage that it can easily run out of memory and it does not work in a clustered environment. To avoid these shortcomings, you could use a Java Persistent API (JPA) based repository instead. The JPA message ID repository uses an object-oriented database to store the message IDs. For example, you can define a route that uses a JPA repository for the idempotent consumer, as follows:

```
import org.springframework.orm.jpa.JpaTemplate;

import org.apache.camel.spring.SpringRouteBuilder;
import static org.apache.camel.processor.idempotent.jpa.JpaMes
sageIdRepository.jpaMessageIdRepository;
...
RouteBuilder builder = new SpringRouteBuilder() {
    public void configure() {
        from("seda:a").idempotentConsumer(
          header("TransactionID"),
          jpaMessageIdRepository(bean(JpaTemplate.class),
"myProcessorName")
        ).to("seda:b");
```

```
    }
};
```

Where the JPA message ID repository is initialized with two arguments: a `JpaTemplate` instance, which provides the handle for the JPA database, and a processor name, which uniquely identifies the current idempotent consumer processor. The `SpringRouteBuilder.bean()` method is a shortcut that references a bean defined in the Spring XML file. The `JpaTemplate` bean provides a handle to the underlying JPA database. See the JPA documentation for details of how to configure this bean.

For more details about setting up a JPA repository, see JPA Component [http://activemq.apache.org/camel/jpa.html] documentation, the Spring JPA [http://static.springframework.org/spring/docs/2.5.x/reference/orm.html#orm-jpa] documentation, and the sample code in the Camel JPA unit test [https://svn.apache.org/repos/asf/activemq/camel/trunk/components/camel-jpa/src/test].

REVISIT - NB: The doc for JPA is woefully inadequate. I need to beef this up, either here or in the JPA component doc page.