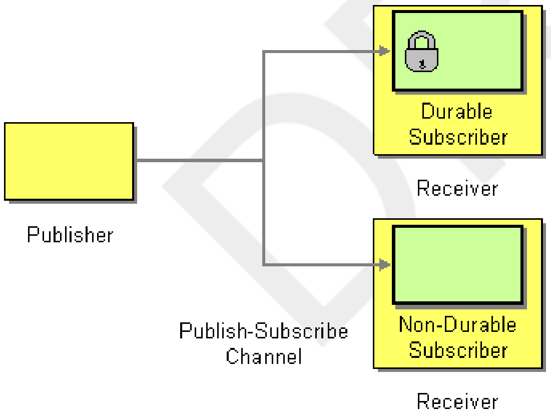
Durable Subscriber

Overview

A *durable subscriber* is a consumer that wants to receive all of the messages sent over a particular publish-subscribe on page 38 channel, including messages sent while the consumer is disconnected from the messaging system. This requires the messaging system to store messages for later replay to the disconnected consumer. There also has to be a mechanism for a consumer to indicate that it wants to establish a durable subscription. Generally, a publish-subscribe channel (or topic) can have both durable and non-durable subscribers, which behave as follows:

- A *non-durable subscriber* can have two states: *connected* and *disconnected*. While a non-durable subscriber is connected to a topic, it receives all of the topic's messages in real time. While a non-durable subscriber is disconnected from a topic, however, it misses all of the message sent during the period of disconnection.
- A *durable subscriber* can have the following states: connected and *inactive*. The inactive state means that the durable subscriber is disconnected from the topic, but wants to receive the messages that arrive in the interim. When the durable subscriber reconnects to the topic, it receives a replay of all the messages sent while it was inactive.

Figure 31. Durable Subscriber Pattern



JMS durable subscriber

The JMS component implements the durable subscriber pattern. In order to set up a durable subscription on a JMS endpoint, you need to specify a *client*

ID, which identifies this particular connection, and a *durable subscription name*, which identifies the durable subscriber. For example, the following route sets up a durable subscription to the JMS topic, `news`, with a client ID of `conn01` and a durable subscription name of `John.Doe`:

```
from("jms:topic:news?clientId=conn01&durableSubscription
Name=John.Doe").
    to("cxf:bean:newsprocessor");
```

You can also set up a durable subscription using the ActiveMQ endpoint:

```
from("activemq:topic:news?clientId=conn01&durableSubscription
Name=John.Doe").
    to("cxf:bean:newsprocessor");
```

If you want to process the incoming messages concurrently, you could use a SEDA endpoint to fan out the route into multiple, parallel segments, as follows:

```
from("jms:topic:news?clientId=conn01&durableSubscription
Name=John.Doe").
    to("seda:fanout");

from("seda:fanout").to("cxf:bean:newsproc01");
from("seda:fanout").to("cxf:bean:newsproc02");
from("seda:fanout").to("cxf:bean:newsproc03");
```

Where each message is processed only once, because the SEDA component supports the competing consumers pattern.