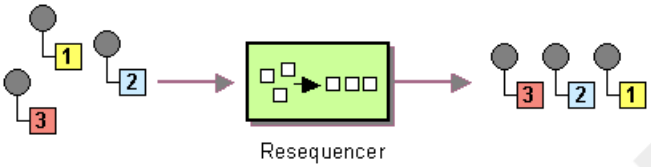
Resequencer

Overview

The *resequencer* pattern enables you to resequence messages according to a sequencing expression. Messages that generate a low value for the sequencing expression are moved to the front of the batch and messages that generate a high value are moved to the back.

Figure 20. Resequencer Pattern



Camel supports two resequencing algorithms:

- *Batch resequencing* collects messages into a batch, sorts the messages and sends them to their output.
- *Stream resequencing* re-orders (continuous) message streams based on the detection of gaps between messages.

Batch resequencing

The batch resequencing algorithm is enabled by default. For example, to resequence a batch of incoming messages based on the value of a timestamp contained in the `TimeStamp` header, you could define the following route in Java DSL:

```
from("direct:start").resequencer(head
er("TimeStamp")).to("mock:result");
```

By default, the batch is obtained by collecting all of the incoming messages that arrive in a time interval of 1000 milliseconds (default *batch timeout*), up to a maximum of 100 messages (default *batch size*). You can customize the values of the batch timeout and the batch size by appending a `batch()` DSL command, which takes a `BatchResequencerConfig` instance as its sole argument. For example, to modify the preceding route so that the batch consists of messages collected in a 4000 millisecond time window, up to a maximum of 300 messages, you could define the Java DSL route as follows:

```
import org.apache.camel.model.config.BatchResequencerConfig;

RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("direct:start").resequencer(head
er("TimeStamp")).batch(new BatchResequencerCon
fig(300,4000L)).to("mock:result");
    }
};
```

You can also use multiple expressions to sort messages in a batch. For example, if you want to sort incoming messages, first of all according to their JMS priority (as recorded in the `JMSPriority` header) and second, according to the value of their time stamp (as recorded in the `TimeStamp` header), you could define a route like the following:

```
from("direct:start").resequencer(header("JMSPriority"), head
er("TimeStamp")).to("mock:result");
```

In this case, messages with the highest priority (that is, low JMS priority number) would be moved to the front of the batch. If more than one message has the highest priority, the highest priority messages would be ordered amongst themselves according to the value of the `TimeStamp` header.

You can also specify a batch resequencer pattern using XML configuration. For example, to define a batch resequencer with a batch size of 300 and a batch timeout of 4000 milliseconds:

```
<camelContext id="resequencerBatch" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start" />
    <resequencer>
      <simple>header.TimeStamp</simple>
      <to uri="mock:result" />
      <!--
        batch-config can be omitted for default (batch)
resequencer settings
      -->
      <batch-config batchSize="300" batchTimeout="4000" />
    </resequencer>
```

```
  </route>
</camelContext>
```

Stream resequencing

To enable the stream resequencing algorithm, you need to append `stream()` to the `resequencer()` DSL command. For example, to resequence incoming messages based on the value of a sequence number in the `seqnum` header, you could define a DSL route as follows:

```
from("direct:start").resequencer(header("se
qnum")).stream().to("mock:result");
```

The stream-processing resequencer algorithm is based on the detection of gaps in a message stream, rather than on a fixed batch size. Gap detection in combination with timeouts removes the constraint of having to know the number of messages of a sequence (that is, the batch size) in advance. Messages must contain a unique sequence number for which a predecessor and a successor is known. For example a message with the sequence number `3` has a predecessor message with the sequence number `2` and a successor message with the sequence number `4`. The message sequence `2,3,5` has a gap because the successor of `3` is missing. The resequencer therefore has to retain message `5` until message `4` arrives (or a timeout occurs).

By default, the stream resequencer is configured with a timeout 1000 milliseconds and a maximum message capacity of 100. To customize the stream's timeout and capacity, you can pass a `StreamResequencerConfig` object as an argument to `stream()`. For example, to configure a stream resequencer with a capacity of 5000 and a timeout of 4000 milliseconds, you could define a route as follows:

```
// Java
import org.apache.camel.model.config.StreamResequencerConfig;

RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("direct:start").resequencer(header("seqnum")).
            stream(new StreamResequencerConfig(5000, 4000L)).

             to("mock:result");
    }
};
```

If the maximum time delay between successive messages (that is, messages with adjacent sequence numbers) in a message stream is known, the resequencer's timeout parameter should be set to this value. In this case, you can guarantee that all messages in the stream are delivered in the correct order to the next processor. The lower the timeout value is compared to the out-of-sequence time difference, the more likely it is that the resequencer will deliver messages out of sequence. Large timeout values should be supported by sufficiently high capacity values, where the capacity parameter is used to prevent the resequencer from running out of memory.

If you want to use sequence numbers of some type other than `long`, you would need to define a custom comparator, as follows:

```
// Java
ExpressionResultComparator<Exchange> comparator = new MyCompar
ator();
StreamResequencerConfig config = new StreamResequencerCon
fig(5000, 4000L, comparator);
from("direct:start").resequencer(header("seqnum")).stream(con
fig).to("mock:result");
```

You can also specify a stream resequencer pattern using XML configuration. For example, to define a stream resequencer with a capacity of 5000 and a timeout of 4000 milliseconds:

```
<camelContext id="resequencerStream" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <resequencer>
      <simple>header.seqnum</simple>
      <to uri="mock:result" />
      <stream-config capacity="5000" timeout="4000"/>
    </resequencer>
  </route>
</camelContext>
```