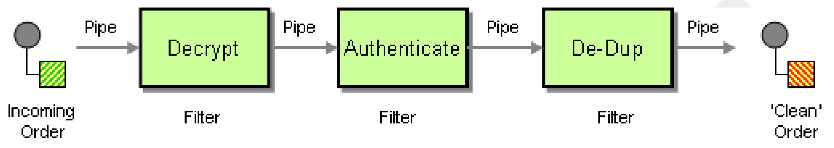
Pipes and Filters

Overview

The *pipes and filters* pattern describes a way of constructiing a route by creating a chain of filters, where the output of one filter is fed into the input of the next filter in the pipeline (analogous to the UNIX `pipe` command). The advantage of the pipeline approach is that it enables you to compose services (some of which can be external to the Mediation Router application) in order to create more complex forms of message processing.

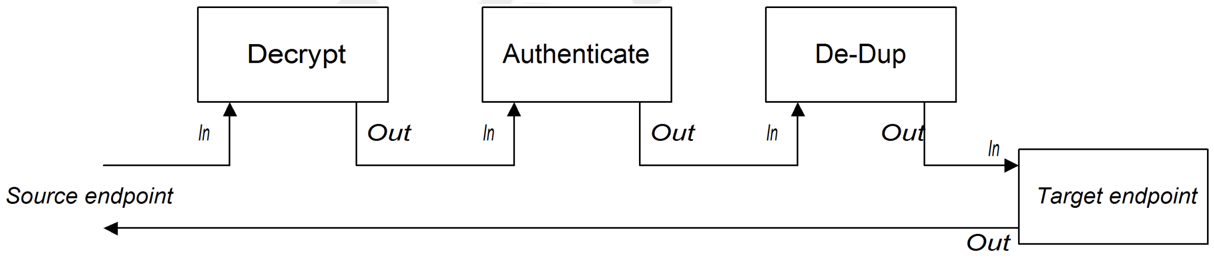
Figure 4. Pipes and Filters Pattern



Pipeline for the InOut exchange pattern

Normally, all of the endpoints in a pipeline would have an input (*In* message) and an output (*Out* message), which implies that they are compatible with the *InOut* message exchange pattern. A typical message flow through an *InOut* pipeline is shown in Figure 5 on page 27.

Figure 5. Pipeline for InOut Exchanges



Where the pipeline connects the output of each endpoint to the input of the next one. The *Out* message from the final endpoint gets sent back to the original caller. You can define a route for this pipeline, as follows:

```
from("jms:RawOrders").pipeline("cxf:bean:decrypt",
"cxf:bean:authenticate", "cxf:bean:dedup", "jms:CleanOrders");
```

The same route can be configured in XML, as follows:

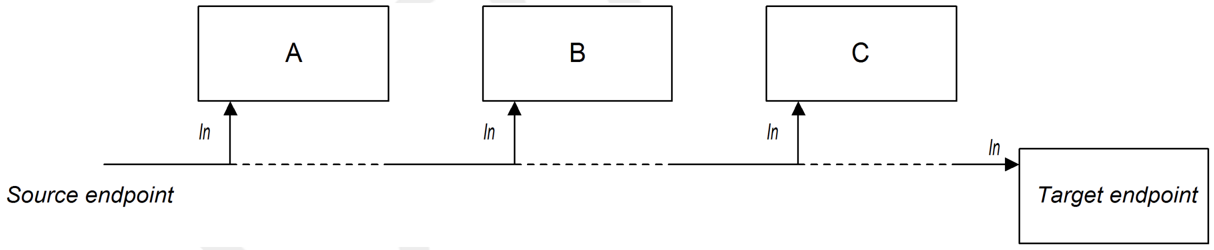
```
<camelContext id="buildPipeline" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="jms:RawOrders"/>
    <to uri="cxf:bean:decrypt"/>
    <to uri="cxf:bean:authenticate"/>
    <to uri="cxf:bean:dedup"/>
    <to uri="jms:CleanOrders"/>
  </route>
</camelContext>
```

There is no dedicated pipeline element in XML: the preceding combination of `from` and `to` elements is semantically equivalent to a pipeline. See Comparison of pipeline() and to() DSL commands on page 28.

Pipeline for the InOnly and RobustInOnly exchange patterns

When there are no *Out* messages available from the endpoints in the pipeline (as is the case for the `InOnly` and `RobustInOnly` exchange patterns), a pipeline cannot be plumbed together in the normal way. In this special case, the pipeline is constructed by passing a copy of the original *In* message to each of the endpoints in the pipeline, as shown in Figure 6 on page 28. This type of pipeline is equivalent to a recipient list with fixed destinations—see Recipient List on page 56.

Figure 6. Pipeline for InOnly Exchanges



The route for this pipeline is defined using the same syntax as an *InOut* pipeline (either in Java DSL or in XML).

Comparison of pipeline() and to() DSL commands

In the Java DSL, you can define a pipeline route using either of the following syntaxes:

- *Using the pipeline() processor command*—use the pipeline processor to construct a pipeline route as follows:

```
from(SourceURI).pipeline(FilterA, FilterB, TargetURI);
```

- *Using the to() command*—use the `to()` command to construct a pipeline route as follows:

  ```
  from(SourceURI).to(FilterA, FilterB, TargetURI);
  ```

  Alternatively, you could use the exactly equivalent syntax:

  ```
  from(SourceURI).to(FilterA).to(FilterB).to(TargetURI);
  ```

You should excercise caution when using the `to()` command syntax, however, because it is *not* always equivalent to a pipeline processor. In the Java DSL, the meaning of `to()` can be modified by the preceding command in the route. For example, when the `multicast()` command precedes the `to()` command, it binds the listed endpoints into a multicast pattern, instead of a pipeline pattern—see Multicast on page 79.