Load Balancer

Overview

The *load balancer* pattern allows you to delegate to one of a number of endpoints using a variety of different load-balancing policies.

Java DSL example

The following route distributes incoming messages amongst the target endpoints, `mock:x`, `mock:y`, `mock:z`, using a round robin load-balancing policy:

```
from("direct:start").loadBalance().roundRobin().to("mock:x",
 "mock:y", "mock:z");
```

XML configuration example

The following example shows how to configure the same route in XML:

```
<bean id = "roundRobinRef" class="org.apache.camel.pro
cessor.loadbalancer.RoundRobinLoadBalancer" />

<camelContext id="camel" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <loadBalance ref="roundRobinRef">
        <to uri="mock:x"/>
        <to uri="mock:y"/>
        <to uri="mock:z"/>
    </loadBalance>
  </route>
</camelContext>
```

Load-balancing policies

The Mediation Router load balancer supports the following load-balancing policies:

- Round robin on page 75
- Random on page 76
- Sticky on page 77
- Topic on page 77

Round robin

The round robin load-balancing policy cycles through all of the target endpoints, sending each incoming message to the next endpoint in the cycle.

For example, if the list of target endpoints is, `mock:x`, `mock:y`, `mock:z`, the round robin, incoming messages would be sent to the following sequence of endpoints: `mock:x`, `mock:y`, `mock:z`, `mock:x`, `mock:y`, `mock:z`, and so on.

You can specify the round robin load-balancing policy in Java DSL, as follows:

```
from("direct:start").loadBalance().roundRobin().to("mock:x",
 "mock:y", "mock:z");
```

Alternatively, you can configure the same route in XML, as follows:

```
<bean id = "roundRobinRef" class="org.apache.camel.pro
cessor.loadbalancer.RoundRobinLoadBalancer" />

<camelContext id="camel" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <loadBalance ref="roundRobinRef">
        <to uri="mock:x"/>
        <to uri="mock:y"/>
        <to uri="mock:z"/>
    </loadBalance>
  </route>
</camelContext>
```

Random

The random load-balancing policy chooses the target endpoint at random from the specified list.

You can specify the random load-balancing policy in Java DSL, as follows:

```
from("direct:start").loadBalance().random().to("mock:x",
"mock:y", "mock:z");
```

Alternatively, you can configure the same route in XML, as follows:

```
<camelContext id="camel" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <loadBalance>
        <random/>
        <to uri="mock:x"/>
        <to uri="mock:y"/>
        <to uri="mock:z"/>
    </loadBalance>
```

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

Sticky

The sticky load-balancing policy directs the *In* message to an endpoint that is chosen by calculating a hash value from a specified expression. The advantage of this load-balancing policy is that expressions of the same value always get sent to the same server. For example, by calculating the hash value from a header that contains a username, you can ensure that messages from a particular user always get sent to the same target endpoint. Another useful approach is to specify an expression that extracts the session ID from an incoming message. In that way, you can ensure that all messages belonging to the same session get sent to the same target endpoint.

You can specify the sticky load-balancing policy in Java DSL, as follows:

```
from("direct:start").loadBalance().sticky(header("user
name")).to("mock:x", "mock:y", "mock:z");
```

Alternatively, you can configure the same route in XML, as follows:

```
<camelContext id="camel" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <loadBalance>
      <sticky>
        <expression>
          <simple>header.username</simple>
        </expression>
      </sticky>
      <to uri="mock:x"/>
      <to uri="mock:y"/>
      <to uri="mock:z"/>
    </loadBalance>
  </route>
</camelContext>
```

Topic

The topic load-balancing policy sends a copy of each *In* message to *all* of the listed destination endpoints (effectively broadcasting the message to all of the destinations, rather like a JMS topic).

You can use the Java DSL to specify the topic load-balancing policy, as follows:

```
from("direct:start").loadBalance().topic().to("mock:x",
"mock:y", "mock:z");
```

Alternatively, you can configure the same route in XML, as follows:

```
<camelContext id="camel" xmlns="http://act
ivemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <loadBalance>
        <topic/>
        <to uri="mock:x"/>
        <to uri="mock:y"/>
        <to uri="mock:z"/>
    </loadBalance>
  </route>
</camelContext>
```