Management Example

Management Example

Available as of Camel 2.1

This is a new example we added in Camel 2.1 as a little example of the new and overhauled JMX Management we did.

The example is included in the distribution at `examples/camel-example-management`. It contains a `README.txt` file with details how to use and run it.

The routes

This example have 3 routes:
1. A route that produces a file with 100 stock quotes every 5th second. This is done using a timer endpoint.
2. A route that uses a file consumer to read files produced from route 1. This route then splits the file and extract each stock quote and send every quote to a JMS queue for further processing. However to avoid exhausting the JMS broker Camel uses a throttler to limit how fast it send the JMS messages. By default its limited to the very low value of 10 msg/second.
3. The last route consumes stock quotes from the JMS queue and simulate some CPU processing (by delaying 100 millis). Camel then transforms the payload to another format before the route ends by a logger which reports the progress. The logger will log the progress by logging how long time it takes to process 100 messages.

The idea and a plan

Now the idea is to use the Camel JMX management to be able to adjust this during runtime. What it allows you to do is to improve the performance of this example.

At first there is a throttler that will throttle how fast Camel sends message to the JMS queue.
For starters you can change this at runtime from the default 10 msg/sec to 500 msg/sec etc. This is done by changing the JMX attribute maximumRequestsPerPeriod on the throttler in the /producer group.

The next issue is that the JMS consumer now cannot catch up and you should see that the number of messages on the JMS queue grows a little by little. You can find the queue from the ActiveMQ mbean and drill down under /queues.

If this goes a bit to slow you can increase the first route in Camel to produce files faster. This is done by changing the period in the timer endpoint from 5000 to let say 2000. Before this takes effect you have to restart the timer consumer. So find the timer consumer and invoke the stop and start JMX operation.

Now you should see the messages start to pile up in the JMS queue. What we do next is to increase the number of concurrent consumers. To do that you have to set this on the JMS endpoint. Set the concrrentConsumers from 1 to 20. And just as the timer consumer this only takes effect when the JMS consumer is restarted. So do a stop and start operation.

What you should see is that Camel should be able to process the files much faster now and the logger should output a higher throughput.

Usage

You can use maven to run the example using

```
mvn camel:run
```

You can then use jconsole to manage the Camel application at runtime. You start jconsole by

```
jconsole
```

It should list a process in the local tab. It should be something with `camel:run` as its our Camel application.
If it does not list any processes in this tab, then click the *Advanced* tab and enter the following in the JMX URI:

```
service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
```

You should now be able to access both ActiveMQ and Camel from jconsole.

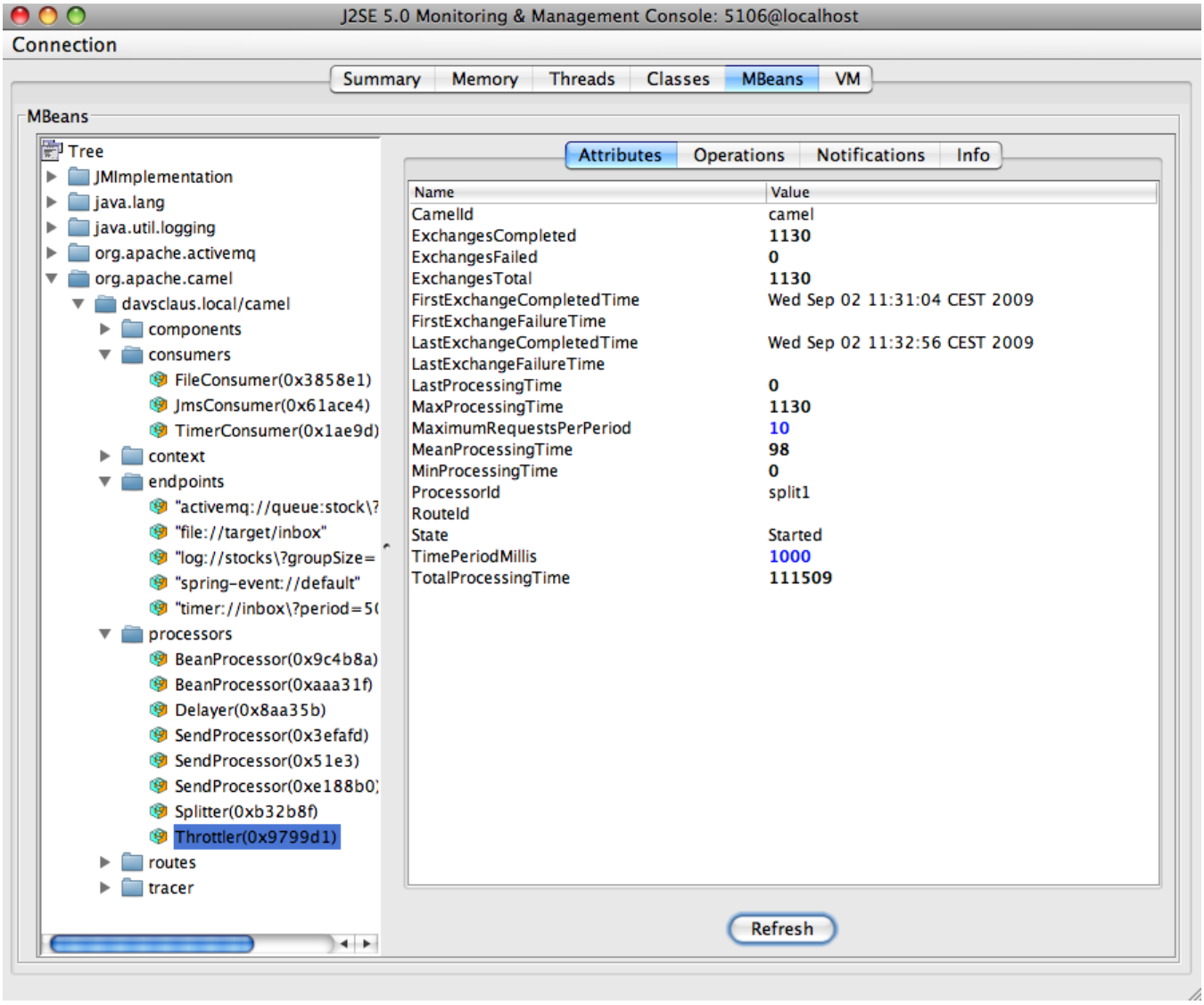
It takes about 10 second before Camel logs to the screen the first time. Its should be like this

```
2009-09-02 11:31:15,393 [enerContainer-1] INFO  stocks                          - Received: 100 messages so far.
Last group took: 10354 millis which is: 9.658 messages per second. average: 9.658
```

Now the goal is to get this faster using the jconsole, as we stated above in the goal section.

JConsole screenshot

Here is a screenshot of the jconsole in action. We have selected the Throttler which is the first one you should manage and change the maximumRequestsPerPeriod from 10 to 500 or even more. You simple click on the number and change it.



See Also

- Examples
- Camel JMX