RoutePolicy

RoutePolicy

Available as of Camel 2.1

A route policy **`org.apache.camel.spi.RoutePolicy`** is used to control route(s) at runtime. For example you can use it to determine whether a route should be running or not. However the policies can support any kind of use cases.

How it works

You associate a route with a given **`RoutePolicy`** and then during runtime Camel will invoke callbacks on this policy where you can implement your custom logic. Camel provides a support class that is a good base class to extend **`org.apache.camel.impl.RoutePolicySupport`**.

There are these callbacks invoked:

- `onInit` **Camel 2.3**
- `onRemove` **Camel 2.9**
- `onStart` **Camel 2.9**
- `onStop` **Camel 2.9**
- `onSuspend` **Camel 2.9**
- `onResume` **Camel 2.9**
- `onExchangeBegin`
- `onExchangeDone`

See the Javadoc of the **`org.apache.camel.spi.RoutePolicy`** for more details. And also the implementation of the **`org.apache.camel.impl.ThrottlingInflightRoutePolicy`** or **`org.apache.camel.impl.ThrottlingExceptionRoutePolicy`** for a concrete example.

Camel provides the following policies out of the box:

- **`org.apache.camel.impl.ThrottlingInflightRoutePolicy`** - a throttling based policy that automatic suspends/resumes route(s) based on metrics from the current in flight exchanges. You can use this to dynamically throttle e.g. a JMS consumer, to avoid it consuming too fast.
- **`org.apache.camel.impl.ThrottlingExceptionRoutePolicy`** - a policy that implements the circuit breaker EIP. This policy will stop consuming from an endpoint based on the number of exceptions that are thrown on the route. This can be used to avoid scenarios where failures on the route cause the message to be rolled back and then re-consumed without being able to be processed.

As of **Camel 2.5**, Camel also provides an ability to schedule routes to be activated, deactivated, suspended and/or resumed at certain times during the day using a ScheduledRoutePolicy (offered via the camel-quartz component).

SuspendableService

If you want to dynamic suspend/resume routes as the **`org.apache.camel.impl.ThrottlingRoutePolicy`** does then its advised to use **`org.apache.camel.SuspendableService`** as it allows for fine grained **`suspend`** and **`resume`** operations. And use the **`org.apache.camel.util.ServiceHelper`** to aid when invoking these operations as it support fallback for regular **`org.apache.camel.Service`** instances.

`ThrottlingInflightRoutePolicy`

The **`ThrottlingInflightRoutePolicy`** is triggered when an Exchange is complete, which means that it requires at least one Exchange to be complete before it *works*.

The throttling inflight route policy has the following options:

Option	Default	Description
`scope`	`Route`	A scope for either **`Route`** or **`Context`** which defines if the current number of inflight exchanges is context based or for that particular route.
`maxInflightExchanges`	`1000`	The maximum threshold when the throttling will start to suspend the route if the current number of inflight exchanges is higher than this value.
`resumePercentOfMax`	`70`	A percentage **`0..100`** which defines when the throttling should resume again in case it has been suspended.
`loggingLevel`	`INFO`	The logging level used for logging the throttling activity.
`logger`	`ThrottlingInflightRoutePolicy`	The logger category.

ThrottlingInflightRoutePolicy compared to the [Throttler] EIP

The **`ThrottlingInflightRoutePolicy`** compared to Throttler is that it does **not** block during throttling. It does throttling that is approximate based, meaning that its more coarse grained and not explicit precise as the Throttler. The Throttler can be much more accurate and only allow a specific number of messages being passed per a given time unit. Also the **`ThrottlingInflightRoutePolicy`** is based its metrics on number of inflight exchanges where as Throttler is based on number of messages per time unit.

`ThrottlingExceptionRoutePolicy`

The **`ThrottlingExceptionRoutePolicy`** (available as of Camel 2.19) is an implementation of the circuit breaker EIP. It is triggered when an Exchange is complete (`onExchangeDone`), which means that it requires at least one Exchange to be complete before it works.

The throttling exception route policy has the following states:

- closed: the route will consume messages from the defined endpoint.
- open: the route will be suspended and will not consume messages from the defined endpoint.
  - the route is opened when a configurable number of exceptions occurs withing a specified time frame.
- half-open: the route will perform a check to see if the route can be moved from open to closed.
  - this will occur by resuming the route and checking for exceptions or by calling an implementation of the **`ThrottlingExceptionHalfOpenHandler.`**
    - If an exception is caught when the route is resumed it will re-open, otherwise it will move to the closed state.
    - If the implemenation of **`ThrottlingExceptionHalfOpenHandler`** is provided and the isReadyToBeClosed method returns true the route will be moved to the closed state. Otherwise it will be moved to the open state.

The throttling exception route policy has the following options:

Option	Default	Description
`failureThreshold`	`0`	The number of exceptions that must be caught before the circuit controlling the route is opened.
`failureWindow`	`0`	The time range, in milliseconds, in which the number of exceptions must occur in order for the circuit to be opened.
`halfOpenAfter`	`0`	Defines how long the circuit will remain open, in milliseconds, before the circuit is moved into the half-open state.
`throttledExceptions`	`null`	An optional List<Class<?>> of exceptions. If this option is set, only these exceptions will count towards meeting the failureThreshold. If this list is left as null any exception will be counted toward the failureThreshold.
`halfOpenHandler`	`null`	An optional implementation of the **`ThrottlingExceptionHalfOpenHandler.`** When provided, the policy will delegate the handling of the half-open state to this class. If it is left as null, the route will resume during the half open state. It is possible for more than one message to be read from the endpoint when the route is resumed during the half-open state.
`keepOpen`	`false`	This option (new as of Camel 2.21) allows the circuit to be placed in the open state when set to true. It overrides all other settings and the half open state will not be processed. The circuit will not be moved out of the open state until this option is set to false.

In the example below, a simple route is configured to open after 2 exceptions are thrown within 30 seconds of each other. When 60 seconds have expired the route will be moved into the half-open state. The check performed during the half-open state will be delegated to the CustomHalfOpenHandler. This class provides an option to check for resources that may be failing independent of resuming the route.

```
@Override
public void configure() throws Exception {
        int threshold = 2;
        long failureWindow = 30000;
        long halfOpenAfter = 60000;

           ThrottlingExceptionRoutePolicy policy = new ThrottlingExceptionRoutePolicy(threshold, failureWindow,
halfOpenAfter, null);
        policy.setHalfOpenHandler(new CustomHalfOpenHandler());

        from(url)
                .routePolicy(policy)
        .log("${body}")
                .to("log:foo?groupSize=10")
        .to("mock:result");
   }
```

`ScheduledRoutePolicy` (Simple and Cron based) using camel Quartz

For more details check out the following links

Configuring Policy

You configure the route policy as follows from Java DSL, using the **`routePolicy`** method:

```
  RoutePolicy myPolicy = new MyRoutePolicy();
  from("seda:foo").routePolicy(myPolicy).to("mock:result");
```

In Spring XML its a bit different as follows using the **`routePolicyRef`** attribute:

```
<bean id="myPolicy" class="com.mycompany.MyRoutePolicy"/>

<route routePolicyRef="myPolicy">
    <from uri="seda:foo"/>
    <to uri="mock:result"/>
</route>
```

Configuring Policy Sets

Available as of Camel 2.7

`RoutePolicy` has been further improved to allow addition of policy sets or a collection of policies that are concurrently applied on a route. The addition of policies is done as follows.

In the example below, the route **`testRoute`** has a **`startPolicy`** and **`throttlePolicy`** applied concurrently. Both policies are applied as necessary on the route.

```
<bean id="date" class="org.apache.camel.routepolicy.quartz.SimpleDate"/>

<bean id="startPolicy" class="org.apache.camel.routepolicy.quartz.SimpleScheduledRoutePolicy">
  <property name="routeStartDate" ref="date"/>
  <property name="routeStartRepeatCount" value="1"/>
  <property name="routeStartRepeatInterval" value="3000"/>
</bean>

<bean id="throttlePolicy" class="org.apache.camel.impl.ThrottlingInflightRoutePolicy">
  <property name="maxInflightExchanges" value="10"/>
</bean>

<camelContext id="testRouteContext" xmlns="http://camel.apache.org/schema/spring">
  <route id="testRoute" autoStartup="false" routePolicyRef="startPolicy, throttlePolicy">
    <from uri="seda:foo?concurrentConsumers=20"/>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

Using `RoutePolicyFactory`

Available as of Camel 2.14

If you want to use a route policy for every route, you can use a **`org.apache.camel.spi.RoutePolicyFactory`** as a factory for creating a **`RoutePolicy`** instance for each route. This can be used when you want to use the same kind of route policy for every routes. Then you need to only configure the factory once, and every route created will have the policy assigned.

There is API on CamelContext to add a factory, as shown below

```
context.addRoutePolicyFactory(new MyRoutePolicyFactory());
```

And from XML DSL you just define a **`<bean>`** with the factory

```
<bean id="myRoutePolicyFactory" class="com.foo.MyRoutePolicyFactory"/>
```

The factory has a single method that creates the route policy

```
    /**
     * Creates a new {@link org.apache.camel.spi.RoutePolicy} which will be assigned to the given route.
     *
     * @param camelContext the camel context
     * @param routeId      the route id
     * @param route        the route definition
     * @return the created {@link org.apache.camel.spi.RoutePolicy}, or <tt>null</tt> to not use a policy for
this route
     */
    RoutePolicy createRoutePolicy(CamelContext camelContext, String routeId, RouteDefinition route);
```

Note you can have as many route policy factories as you want. Just call the **`addRoutePolicyFactory`** again, or declare the other factories as **`<bean>`** in XML.

See Also

- Route Throttling Example for an example using this in practice with the **`ThrottlingInflightRoutePolicy`**
- ScheduledRoutePolicy for information on policy based scheduling capability for camel routes
- MetricsRoutePolicyFactory for information on a policy using the metrics component to expose route statistics using the metrics library.
- Architecture