**Camel Routes with Direct and SEDA Endpoints**

When designing routes, you may sometimes want a route to have multiple inputs. This article shows the main ways to achieve this by joining routes together in Apache Camel.

Let’s illustrate this using an example. You have defined a route that validates an incoming order, by passing a message to some underlying system. Your orders initially come through JMS. But what happens when orders start coming from new sources (e.g. in a file upload, or from a web service call).

To avoid having to repeat the same route code, Camel has features built-in which allow routes to have multiple inputs, by using a range of joining components to glue these routes together.

So how does it work? Camel glues endpoints together using the components Direct, VM and SEDA.

These components join your Camel routes together in different ways. They are collectively known as Camel’s in-memory messaging components, because they allow messages to be exchanged between routes, but the message stays in memory at all times. This is an important detail, which will be discussed later.

Let’s now look at each of these components to see how they differ

##**Direct**

This has to be one of the most-asked questions by Camel beginners! What does “direct” mean in a route? It might be so frequently asked because you see the code direct: in so many Camel tutorials on the web. But what does it do?

direct is one of the most simple ways of linking your routes together. When used in a from() definition, it creates an internal, Camel-only endpoint for the consumption of messages. It’s used in the form:

direct:yourname

Where yourname is the name to give your endpoint. This can be any name you like, just like naming a method in Java.

It is often used in examples as a way to provide an entry point into a route, without having to expose a web service, or otherwise rely on an external interface.

The Direct component provides a way for routes to be joined in a synchronous way. This means that when one route sends a message to another direct endpoint using to("direct:myroute"), the route myroute will be executed in the same thread as the first route.

Let’s illustrate this with an example:

from("file:/home/files/in").to("direct:processTheFile");`

from("direct:processTheFile").to("log:samplelog");

This very simple example receives files using Camel’s File component. Each file processed is passed, as an Exchange, to the direct endpoint processTheFile. Separately, we have defined the processTheFile endpoint as the start component for a route which simply writes the Exchange to the Camel log. All of this happens synchronously, within the same thread.

This simplicity comes with some drawbacks. Direct endpoints can only be accessed by other routes that are running in the same Camel Context in the same JVM. This means that you cannot access a Direct endpoint from another Camel Context. Remember the Camel Context can be thought of as the environment where your Camel routes are defined and later instantiated.

**##Direct-VM**

This synchronous component works in a very similar way to the Direct component described above.

When used as a start component, Direct-VM allows a route to be invoked synchronously from another route.

The difference with the Direct-VM component is that direct-vm endpoints can be seen from other Camel Contexts, as long as they share the same Java Virtual Machine (JVM). This opens up possibilities of linking routes together that were not developed in the same Camel Context.

**##SEDA**

While the Direct and Direct-VM components take a synchronous approach to joining routes, SEDA does the opposite. It allows routes to be connected in an asynchronous way; that is, when a Camel route publishes a message to a seda: endpoint, the message is sent and control is returned immediately to the calling route.

A SEDA consumer endpoint contains a buffer which is used to store the incoming messages. SEDA is configured by default to create a pool of threads to process incoming messages, so several messages can be processed at once, making it potentially more performant.

In this way, SEDA can be thought of as a simple replacement for JMS queues. It provides message queue functionality, but without the overhead of running an external message broker like ActiveMQ.

A simple example could be:

from("activemq:orders").to("seda:processOrder");

from("seda:processOrder").to("file:orders/out");

In the example above, messages will be pulled from an ActiveMQ queue named orders and published to the SEDA endpoint. Messages will arrive at the processOrder SEDA endpoint and be processed by a pool of threads which write the orders to a location on disk, using the file: component.

##VM In a similar way to how Direct and Direct-VM are related, VM is a similar component to SEDA.

When used as a start component, SEDA allows a route to be invoked asynchronously from another route.

However the difference between SEDA and VM is that the VM component allows endpoints to be accessed from different Camel Contexts, if they are running in the same JVM.

Again, the VM component opens up possibilities of linking routes together that were not developed in the same Camel Context, in an asynchronous manner.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*