Writing Components

Writing Components

Apache Camel is designed to make it very easy to drop in new components whether they be routing components, transformers, transports etc. The idea of a component is to be a factory and manager of Endpoints.

Here are the main steps to writing a component.

- write a POJO which implements the Component interface. The simplest approach is just to derive from DefaultComponent
- to support auto-discovery of your component add a file to *META-INF/services/org/apache/camel/component/FOO* where FOO is the URI scheme for your component and any related endpoints created on the fly. The latter file should contain the definition of the component class. For example if your component is implemented by the *com.example.CustomComponent* class, the service file should contain the following line - *class=com.example.CustomComponent* .

Users can then either explicitly create your component, configure it and register with a CamelContext or users can use a URI which auto-creates your component.

Writing Endpoints

When implementing an Endpoint you typically may implement one or more of the following methods

- createProducer() will create a Producer for sending message exchanges to the endpoint
- createConsumer() implements the Event Driven Consumer pattern for consuming message exchanges from the endpoint via a Processor when creating a Consumer
- createPollingConsumer() implements the Polling Consumer pattern for consuming message exchanges from the endpoint via a PollingConsumer

Typically you just derive from DefaultEndpoint and implement the createProducer() and / or createConsumer() methods. The createPollingConsumer() method will be created by default for you in the DefaultEndpoint class.

If your endpoint is a polling-centric component you can derive from DefaultPollingEndpoint and then implement createPollingConsumer(); the createConsumer() method will be created for you which avoids you having to write any polling code.

Annotating your Endpoint

As of **Camel 2.12** if you want to benefit from the automatic generation of HTML documentation for all the parameters on your endpoint as part of the maven site reports, you need to annotate your Endpoint's parameters.

So this means you add a @UriEndpoint annotation to your Endpoint class and then annotate each parameter you wish to be configured via the URI configuration mechanism with @UriParam (or @UriParams for nested configuration objects).

In addition its recommended that your Component implementation inherit from the UriEndpointComponent base class as that means your Component will automatically generate better metadata for the ComponentConfiguration API.

Refer to the Endpoint Annotations guide for details.

Using a Processor

If you are writing a simple endpoint which just processes messages in some way, you can just implement a Processor and use that to create an endpoint.

Dependency injection and auto-discovery

When using auto-discovery the CamelContext will default to its Injector implementation to inject any required or optional dependencies on the component. This allows you to use auto-discovery of components via URIs while still getting the benefits of dependency injection.

For example your component can depend on a JDBC DataSource or JMS ConnectionFactory which can be provided in the ApplicationContext in Spring or Module in Guice.

So you can if you prefer configure your Component using an IoC framework like Spring or Guice; then add it to the CamelContext. Or you can let the Component auto-inject itself as the endpoints are auto-discovered.

Options

If your component has options you can let it have public getters/setters and Camel will automatically set the properties when the endpoint is created. If you however want to take the matter in your own hands, then you must remove the option from the given parameter list as Camel will validate that all options are used. If not Camel will throw a ResolveEndpointFailedException stating some of the options are unknown.

The parameters is provided by Camel in the createEndpoint method from DefaultComponent:

```
protected abstract Endpoint<E> createEndpoint(String uri, String remaining, Map parameters)
```

The code is an example from the SEDA component that removes the size parameter:

```
    public BlockingQueue<Exchange> createQueue(String uri, Map parameters) {
        int size = 1000;
        Object value = parameters.remove("size");
        if (value != null) {
            Integer i = convertTo(Integer.class, value);
            if (i != null) {
                size = i;
            }
        }
        return new LinkedBlockingQueue<Exchange>(size);
    }
```

See Also

- Configuring Camel
- Endpoint
- Component
- Creating a new Camel Component with Maven