Bean Language

Bean Language

The purpose of the Bean Language is to be able to implement an Expression or Predicate using a simple method on a bean. The bean name is resolved using a Registry, such as the Spring **`ApplicationContext`**, then a method is invoked to evaluate the Expression or Predicate. If no method name is provided then one is chosen using the rules for Bean Binding; using the type of the message body and using any annotations on the bean methods.

The Bean Binding rules are used to bind the Message Exchange to the method parameters; so you can annotate the bean to extract headers or other expressions such as XPath or XQuery from the message.

Using Bean Expressions in Java

```
from("activemq:topic:OrdersTopic")
  .filter().method("myBean", "isGoldCustomer")
    .to("activemq:BigSpendersQueue");
```

Using Bean Expressions in Spring XML

```
<route>
  <from uri="activemq:topic:OrdersTopic"/>
  <filter>
    <method ref="myBean" method="isGoldCustomer"/>
    <to uri="activemq:BigSpendersQueue"/>
  </filter>
</route>
```

Bean Attribute Now Deprecated

The **`bean`** attribute of the method expression element is now deprecated. Use the **`ref`** attribute instead.

Writing the Expression Bean

The bean in the above examples is just any old Java Bean with a method called **`isGoldCustomer()`** that returns some object that is easily converted to a **`boolean`** value in this case, as its used as a predicate.

Example:

```
public class MyBean {
  public boolean isGoldCustomer(Exchange exchange) {
          // ...
  }
}
```

We can also use the Bean Integration annotations.

Example:

```
public boolean isGoldCustomer(String body) {...}
```

or

```
public boolean isGoldCustomer(@Header(name = "foo") Integer fooHeader) {...}
```

So you can bind parameters of the method to the Exchange, the Message or individual headers, properties, the body or other expressions.

Non-Registry Beans

The Bean Language also supports invoking beans that isn't registered in the Registry. This is usable for quickly to invoke a bean from Java DSL where you don't need to register the bean in the Registry such as the Spring **`ApplicationContext`**. Camel can instantiate the bean and invoke the method if given a class or invoke an already existing instance.

Example:

```
from("activemq:topic:OrdersTopic")
  .filter().expression(BeanLanguage(MyBean.class, "isGoldCustomer"))
  .to("activemq:BigSpendersQueue");
```

The 2nd parameter **`isGoldCustomer`** is an optional parameter to explicit set the method name to invoke. If not provided Camel will try to invoke the most suitable method. If case of ambiguity Camel will thrown an Exception. In these situations the 2nd parameter can solve this problem. Also the code is more readable if the method name is provided. The 1st parameter can also be an existing instance of a Bean such as:

```
private MyBean my;

from("activemq:topic:OrdersTopic")
  .filter().expression(BeanLanguage.bean(my, "isGoldCustomer"))
  .to("activemq:BigSpendersQueue");
```

In **Camel 2.2**: you can avoid the **`BeanLanguage`** and have it just as:

```
private MyBean my;

from("activemq:topic:OrdersTopic")
  .filter().expression(bean(my, "isGoldCustomer"))
  .to("activemq:BigSpendersQueue");
```

Which also can be done in a bit shorter and nice way:

```
private MyBean my;

from("activemq:topic:OrdersTopic")
  .filter().method(my, "isGoldCustomer")
  .to("activemq:BigSpendersQueue");
```

Other Examples

We have some test cases you can look at if it'll help

- MethodFilterTest is a JUnit test case showing the Java DSL use of the bean expression being used in a filter
- aggregator.xml is a Spring XML test case for the Aggregator which uses a bean method call to test for the completion of the aggregation.

Dependencies

The Bean language is part of **`camel-core`**.