POJO Consuming

@Consume

To consume a message you use the @Consume annotation to mark a particular method of a bean as being a consumer method. The uri of the annotation defines the Camel Endpoint to consume from.

e.g. lets invoke the `onCheese()` method with the String body of the inbound JMS message from ActiveMQ on the cheese queue; this will use the Type Converter to convert the JMS ObjectMessage or BytesMessage to a String - or just use a TextMessage from JMS

```
public class Foo {

  @Consume(uri="activemq:cheese")
  public void onCheese(String name) {
    ...
  }
}
```

The Bean Binding is then used to convert the inbound Message to the parameter list used to invoke the method .

What this does is basically create a route that looks kinda like this

```
from(uri).bean(theBean, "methodName");
```

When using more than one CamelContext

When you use more than 1 CamelContext you might end up with each of them creating a POJO Consuming; therefore use the option `context` on **@Consume** that allows you to specify which CamelContext id/name you want it to apply for.

Using context option to apply only a certain CamelContext

See the warning above.

You can use the `context` option to specify which CamelContext the consumer should only apply for. For example:

```
  @Consume(uri="activemq:cheese", context="camel-1")
  public void onCheese(String name) {
```

The consumer above will only be created for the CamelContext that have the context id = `camel-1`. You set this id in the XML tag:

```
<camelContext id="camel-1" ...>
```

Using an explicit route

If you want to invoke a bean method from many different endpoints or within different complex routes in different circumstances you can just use the normal routing DSL or the Spring XML configuration file.

For example

```
from(uri).beanRef("myBean", "methodName");
```

which will then look up in the Registry and find the bean and invoke the given bean name. (You can omit the method name and have Camel figure out the right method based on the method annotations and body type).

Use the Bean endpoint

You can always use the bean endpoint

```
from(uri).to("bean:myBean?method=methodName");
```

Using a property to define the endpoint

Available as of Camel 2.11

The following annotations @Consume, @Produce, @EndpointInject, now offers a `property` attribute you can use to define the endpoint as a property on the bean. Then Camel will use the getter method to access the property.

This applies for them all

The explanation below applies for all the three annotations, eg @Consume, @Produce, and @EndpointInject

For example

```
public class MyService {
  private String serviceEndpoint;

  public void setServiceEndpoint(String uri) {
     this.serviceEndpoint = uri;
  }

  public String getServiceEndpoint() {
     return serviceEndpoint
  }

  @Consume(property = "serviceEndpoint")
  public void onService(String input) {
     ...
  }
}
```

The bean `MyService` has a property named `serviceEndpoint` which has getter/setter for the property. Now we want to use the bean for POJO Consuming, and hence why we use @Consume in the onService method. Notice how we use the `property = "serviceEndpoint` to configure the property that has the endpoint url.

If you define the bean in Spring XML or Blueprint, then you can configure the property as follows:

```
<bean id="myService" class="com.foo.MyService">
  <property name="serviceEndpoint" value="activemq:queue:foo"/>
</bean>
```

This allows you to configure the bean using any standard IoC style.

Camel offers a naming convention which allows you to not have to explicit name the property.
Camel uses this algorithm to find the getter method. The method must be a getXXX method.

1. Use the property name if explicit given
2. If no property name was configured, then use the method name
3. Try to get the property with name*Endpoint* (eg with Endpoint as postfix)
4. Try to get the property with the name as is (eg no postfix or postfix)
5. If the property name starts with **on** then omit that, and try step 3 and 4 again.

So in the example above, we could have defined the @Consume annotation as

```
  @Consume(property = "service")
  public void onService(String input) {
```

Now the property is named 'service' which then would match step 3 from the algorithm, and have Camel invoke the getServiceEndpoint method.

We could also have omitted the property attribute, to make it implicit

```
@Consume
public void onService(String input) {
```

Now Camel matches step 5, and loses the prefix **on** in the name, and looks for 'service' as the property. And because there is a getServiceEndpoint method, Camel will use that.

Which approach to use?

Using the @Consume annotations are simpler when you are creating a simple route with a single well defined input URI.

However if you require more complex routes or the same bean method needs to be invoked from many places then please use the routing DSL as shown above.