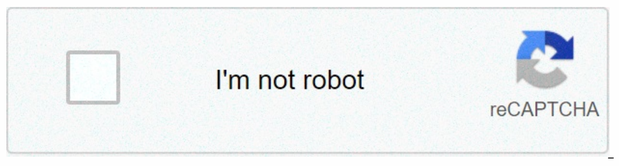


Continue

# Apache camel file transformation

The User Manual Architecture Transformer Transformer performs a declared conversion of messages according to the declared input type and/or output type on the route definition that declares the expected message type. The default camel message implements DataTypeAware, which allows you to keep the message type represented by the DataType. If the input type or output type is declared by the input type or output type of the route definition and is different from the actual message type at run time, the camel internal processor will look for transformers to convert and apply from the current message type to the expected message type. If the conversion succeeds or the message is set to an already expected type, the message data type is updated. Schemes are types of data models such as java, xml, and json, and name is the name of individual data types. If you specify only a scheme, it hits all data types with that scheme, such as wildcards. Transformer description data format transformer transform using data format endpoint transformer transformer transform using endpoint custom transformer transform using custom transformer class. Transformers must be subclasses of org.apache.camel.spi.Transformer All Transformers have a common option to specify the data types supported by transformers. You must specify both scheme or fromType and toType. Name A type of data model, such as description scheme xml or json. For example, xml applies transformers to all java -> xml and xml -> java transformations. From the data type that converts the data type to be converted. Enter the data type to convert. Name Description Type Data Format Reference to ID Here is an example of specifying the type of bound data format: binding data format binding = new Bind DataFormat(); binding type.set type (binding type. Csv). set .class settings. conversion () .fromType(com.example.order.class. .toType(csv:CSVOrder) .withDataFormat (binding) <dataFormatTransformer fromtype=java:com.example.Order totype=csv:CSVOrder> <cindy id=csvf type=Csv classtype=com.example.Order></cindy> </dataFormatTransformer> Reference to name description ref endpoint ID uri endpoint URI This is an example of specifying an endpoint URI in Java DSL: Transformers () .fromType(xml) .toType(json) .withUri (dozer:myDozer?mappingFile=mapping file=.xml.xml.). And here's an example of specifying an endpoint ref in XML DSL: <endpointTransformer ref=myDozerEndpoint fromtotype=xml totype=json></endpointTransformer> Note that transformers must be subclasses of the org.apache.camel.spi.transformer Name ref Reference to the fully qualified class name of the custom Transformer ID class name custom Transformer class.class an example of specifying a custom Transformer class. <customTransformer ></customTransformer >fromType=xmltoType=json/> For example, if you want to declare an endpoint transformer that uses the xslt component to convert xml:ABCOrder to xml:XYZOrder, you can do the following: <camelContext id=camel xmlns= amp;gt; <transformers> <endpointTransformer uri=xslt:transform.xsl fromtype=xml:ABCOrder totype=xml:XYZOrder></endpointTransformer> </transformers> .... </camelContext> If you have the following route definition, the transformer above applies when the direct:abc endpoint sends a message directly: from (direct:abc) .inputType (xml:ABCOrder) .to (direct:xyz) to (direct:xyz) .inputType (xml:XYZOrder) (somewhere:else). Data conversion means converting data from one form to another. <camelContext id=camel xmlns= amp;gt; <route> <from uri=direct:abc></from> <inputType urn=xml:ABCOrder></inputType> <to uri=direct:xyz></to> </route> <route> <from uri=direct:xyz></from> <inputType urn=xml:XYZOrder></inputType> <to uri=somewhere:else></to> </route> </camelContext> For example, XML to CSV, XML to JSON, and so on. Apache Camel data conversion can be achieved using a processor in routing logic, using beans, or using Transform() in DSL. This article describes how routing logic uses processors to convert CSV to XML. The 2.0 Camel Processor Camel Processor is one of the key building blocks and provides access to the full messages that are forwarded, such as the body, headers, and properties associated with the message. When you create a processor, you can change the elements and properties of the message body. For example, you can add custom properties to headers or convert messages from CSV to XML. A camel processor is an interface defined by org.apache.camel.processor with a single method. Public voiding process (Exchange exchange) throws input and output messages with exception 3.0 In this article, we will see how to convert CSV to XML using a processor. The following are input and output messages. 3.1 Input message (CSV) 1,Robert,3000 2,James,3000 3,Steve,5000 4,Brian,7000 5,christian,6000 3.2 Output message (XML) <Employees> <Employee> <EmployeeID>1</EmployeeID> <Name>Robert </Name> <Salary>3000</Salary> </Employee> <Employee> <EmployeeID>2</EmployeeID> <Name>James</Name> <Salary>3000</Salary> </Employee> <Employee> <EmployeeID>3</EmployeeID> <Name>Steve</Name> <Salary>5000</Salary> <</Salary> </Employee> <Employee> <EmployeeID>4</EmployeeID> <Name>Brian </Name> <Salary>7000</Salary> <<<<<<Employee> <Employee > <EmployeeID>5</EmployeeID> <Name>Christian</Name> <Salary>6000</Salary> </Employee> </Employees> 4.0 The code in this class displays the Camel route defined in this class, and the process root calls the MyTransform class and actually converts the CSV to an XML message. package com.demo;import Import <a0>. </a0> Import <a0>. </a0> Public class CSVToXML conversion { Public static void main (string[] args) throws an exception { Camel context _ctx=New default camel context() _ctx.addRoutes (new root builder(){ public void configuration () ) ( \\ApacheCamel\\Demo1\IN) .process (new My Transform) .to (file:D:\ApacheCamel\\Demo1\OUT?file name=emp.xml ) } }); _ctx start thread.sleep(4000);_ctx stop();} } The process method retrieves the input data as an exchange object. In this section, you load the exchange object as a string and convert the CSV to XML. Finally, the converted message is set as an exchange. Import package com.demo;. Import <a0>. </a0> The public class MyTransform implements a processor @Override throws a public void process (exchange) exception { string myString = exchange.getIn() getBody (string .class). String[] line separator = myString.split (system.getProperty). sb = new string buffer ()sb.append()<Employees> (string line data: line separator) { string [] comma-delimited =line data.split(,),;sb.append(<Employee>sb.append() <EmployeeID>+Comma Separator[0].toString();</EmployeeID>sb.append(<Name>+Comma Separator[1].toString()). </Name>sb.append(<Salary>+Comma Separator[2].toString();</Salary>; sb.append()</Employee>} sb.append( )</Employees>System.Out.Printon (My Processor Complete). returns a value of 100000000 } } If you use 5.0 Spring DSL Spring DSL, you can use the following XML to get the same results: <?xml version=1.0 accessing=UTF-8?>Marshaling and unmarshalling xml using CSV marshaling and unmarshalling 6.1 Definition of employee bean class package com.lealnamel.processor;Public class Employee { Private String Employee ID; private string salary; public string getEployeeID() { return employee ID; } public void setName (string name) { name = name; } public string getSalary() { return salary; } public void set salary (string salary) { salary = salary; @Override } public string toString() { return employee { + EmployeeID= + EmployeeID + , Name= + name + '" <beans xmlns= amp;gt; <bean id=myTransform class=com. CSVToXML.demo.MyTransform></bean> <camelContext xmlns= amp;gt; <route> <from uri=file:D:\\ApacheCamel\\Demo1\IN></from> <process ref=myTransform><</process> <to uri=file:D:\\ApacheCamel\\Demo1\\OUT?fileName=emp.xml></to> </route> </camelContext > </beans> Define the 6.2 processor package com.leancemel.processor. Import. Import Import <a0>. </a0> Import <a0>. </a0> The public class custom processor XStream implements the org.apache.camel.Processor { public void process { string newBody = exchange.getIn() getBody (string .class). String tokenized tokenizer = new string tokenizer (newBody,,); employee = new employee (; meanwhile (talkniser.hasmore element())){employee.set employeeeID((((string) talkizer.nextElement();employee.set name (string)talknaiser.nextElement();employee.set salary (string) talknaiser.nextElement();; Exchange.getIn() Configuration Body (Employee);} } } 6.3 com.leannamuel.route.xstream;imports that define camel root packages. Import <a0>. </a0> Import <a0>. </a0> Import. Import <a0>. </a0> Import <a0>. </a0> Extends a public class. map<String, string=> alias = new hashmap<String, string=>(); alias.put (employee, employee.class.getName()); xstreamDefinition.setAliases (alias); Returns xstreamDefinition; } 7.0 Conclusion It is very easy to convert one message format to another using Apache Camel. Camel converts messages in a variety of ways. This article uses processors in routing logic, and the next article explains how to use bean and Transform () methods to convert messages. You learned how to convert CSV to XML using Apache Camel. Camel. </String,></String,>