



Camel学习分享

Apache

Camel

沈超琦 2021-08

# 目 录

Camel介绍

Camel组件分享

源码解读-ProcessorDefinition

总结



**Figure 1.1** Camel is the glue between disparate systems.

Camel 框架的核心是一个路由引擎-更准确地说，是一个路由引擎的构建器。它允许你定义你自己的路由规则。如图所示，Camel是不同系统之间的胶水。Camel 的基本原则之一是它不关于您需要处理的数据类型。

## Camel介绍

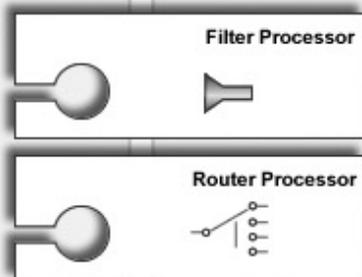


# Camel

Integration Engine And Router

## Camel Endpoints

- \* Camel can send messages to them
- \* Or Receive Messages from them



## Camel Processors

- \* Are use to wire Endpoints together
- \* Routing
- \* Transformation
- \* Mediation
- \* Interception
- \* Enrichment
- \* Validation
- \* Tracking
- \* Logging

## JMS Component

JMS API

## HTTP Component

Servlet API

## File Component

File System

## Camel Components

- \* Provide a uniform Endpoint Interface
- \* Act as connectors to all other systems



JMS Provider  
ActiveMQ | IBM |  
Tibco | Sonic ...



HTTP Client



Local File  
System

路由引擎使用路由routes作为指示位置的规范标识在哪里消息被路由。路由被camel定义的某一种DSL所定义。Processors(处理器)用于转换和操作路由期间的消息以及实现所有DSL中有定义相应名称的EIP。组件是Camel中用于添加与其他连接系统的扩展点。组件提供端点接口将这些系统暴露给Camel的其余部分。

## Camel介绍



**exchange是路由期间消息的容器**

**ExchangeID** — 标识交换的唯一 ID。

**MEP** —表示您使用的是 InOnly 还是InOut 消息模式

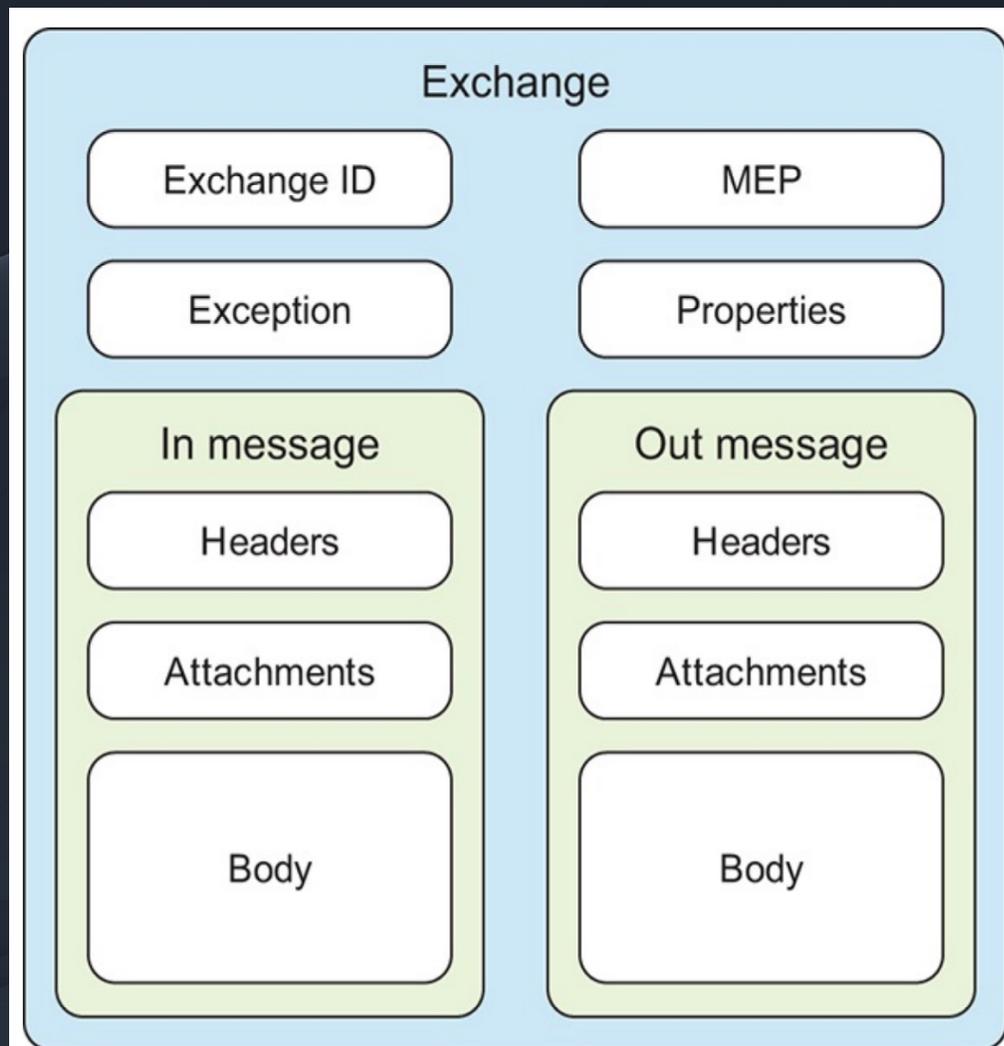
**Exception** — 如果在路由过程中的任何时间发生错误，异常将在异常字段中设置。

**Properties(属性)** — 类似于消息头，但它们持续整个交换的持续时间。属性用于包含全局级别的信息，而消息头是特定的

**In message** - 这是输入消息

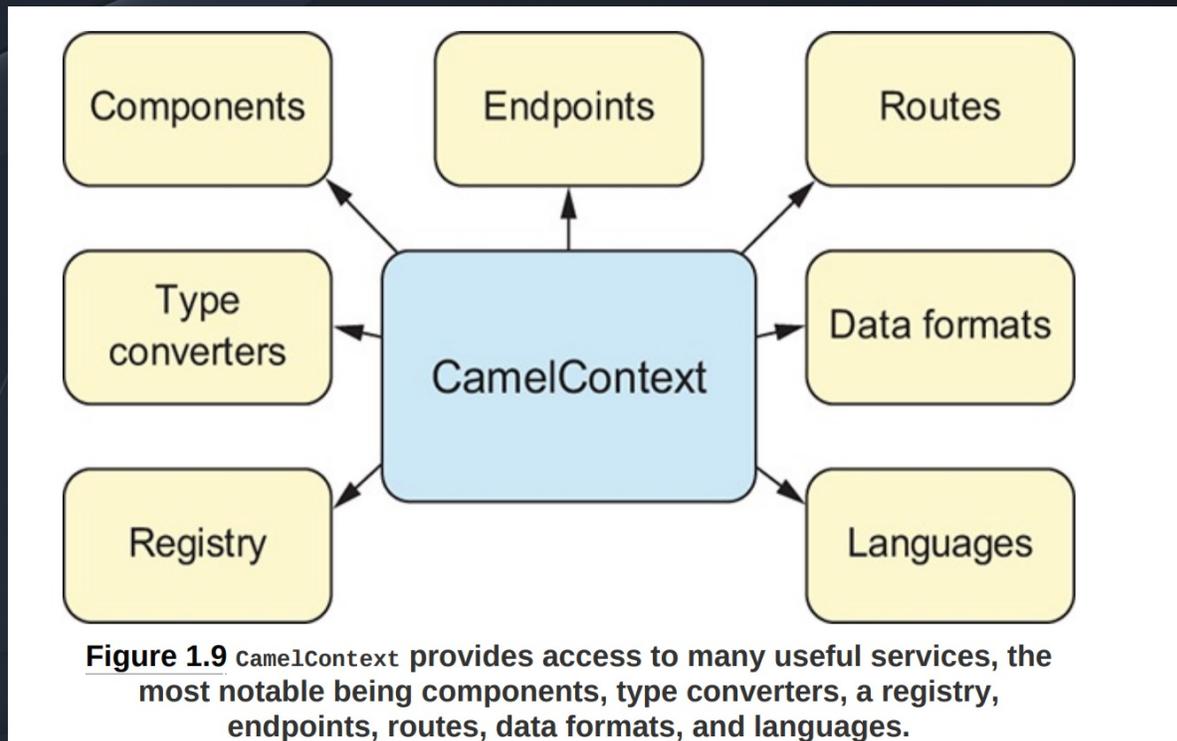
**Out message** —仅当MEP 是 InOut。 out 消息包含回复消息。

Exchange在路由的整个生命周期中是相同的，但是消息可以改变，



**Figure 1.7** A Camel exchange has an ID, MEP, exception, and properties. It also has an *in* message to store the incoming message, and an *out* message to store the reply.

Service	Description(描述)
Components(组件)	包含使用的组件。骆驼能够装载通过类路径上的配置自动加载或在xml中被定义
Endpoints(端点)	包含了被使用的端点
Routes(路由)	包含了被使用的路由
Typeconverters(类型转换器)	包含加载的数据格式。
Registry(注册表)	包含允许您查找 bean 的注册表
Languages	包含加载的语言。Camel 允许您使用多种语言来创建表达式。如XPath,cron等



**exchange**是路由期间消息的容器

**ExchangeID** — 标识交换的唯一 ID。

**MEP** —表示您使用的是 InOnly 还是InOut 消息模式

**Exception** — 如果在路由过程中的任何时间发生错误，异常将在异常字段中设置。

**Properties(属性)** — 类似于消息头，但它们持续整个交换的持续时间。属性用于包含全局级别的信息，而消息头是特定的

**In message** - 这是输入消息

**Out message** —仅当MEP 是 InOut。 out 消息包含回复消息。

Exchange在路由的整个生命周期中是相同的，但是消息可以改变，

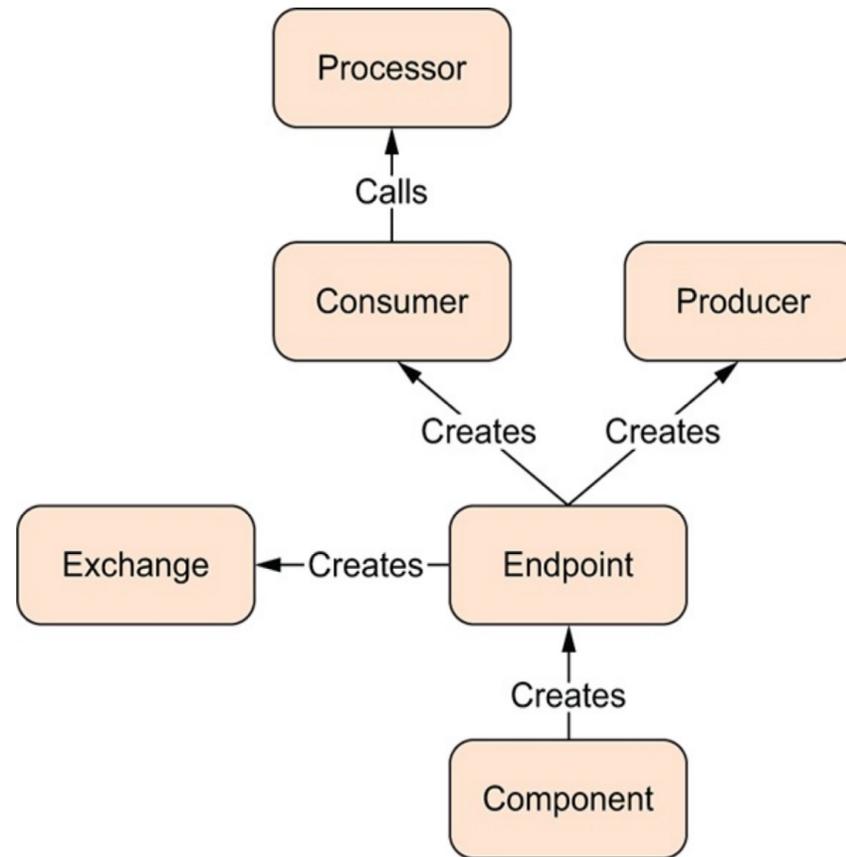


Figure 1.13 How endpoints work with producers, consumers, and an exchange

配合断点演示

# 组件介绍

JDBC组件

toD组件

Groovy组件(语言) Simple组件(语言)

# JDBC组件



jdbc组件使您能够通过JDBC访问数据库，在消息正文中发送SQL查询（SELECT）和操作（INSERT，UPDATE等）。

此组件只能用于定义生产者端点，也就是说不能在from () 语句中使用JDBC组件。

假设现在有两个库中的a,b两个表,我需要从test2库的a表中把id查询出来复制到test3库的b表中,现在就让我们用camel的jdbc组件来实现这个功能路由.



```
<route xmlns="http://camel.apache.org/schema/spring" autoStartup="false"
id="jdbc-splite">
  <from uri="jetty:http://0.0.0.0:1314/jdbc"/>
  <setBody>
    <constant>select * from user </constant>
  </setBody>
  <to uri="jdbc:slave2"/>
  <split>
    <simple>${body}</simple>
    <setBody>
      <simple>insert into user
values('${body[user_name]}','${body[age]}')</simple>
    </setBody>
    <to uri="jdbc:slave3"/>
  </split>
  <setBody>
    <constant>测试成功</constant>
  </setBody>
</route>
```

```
from(uri:"jetty:http://0.0.0.0:1314/jdbc").id("jdbc-splite").autoStartup(false)
  .setBody(constant(value:"select * from user "))
  .to("jdbc:datasource")
  .split(body())
  .setBody(simple(value:"insert into user values('${body[user_name]}','${body[age]}')"))
  .to("jdbc:datasource")
  .end()
  .setBody(constant(value:"测试成功"))
```



# toD组件



个人认为 toD组件其实是对to组件的更大宽容度的引申,可以在toD组件中使用simple组件来达到动态改变路由地址的作用,to可以做到的所有事情,toD也可以做到.就像camel官网所说的一样 *We made to and toD separated on purpose. As the former allows Camel to optimize as it knows its a static endpoint, and the latter is dynamic.* 主要的区别是是否可为动态路由

假设我们现在有个另外一个需求,对外提供统一的接口,根据用户header中的值去到不同的队列中去推送数据  
这个时候你可能会说那普通的to组件不能实现吗,答案是否定的,在官网中to组件中的参数在运行时是不能改变的.也就是说to是一个静态的路由,

```
from(uri: "jetty:http://0.0.0.0:3331/toD").setExchangePattern(ExchangePattern.InOnly).id("toD演示").convertBodyTo(String.class, charset: "UTF-8")
    .toD("activemq:"+ "${header.toQueue}");

//每2s产生一条随机数
//
//
//
from("timer://foo?fixedRate=true&period=1000s").id("项目演示之---toD").convertBodyTo(String.class, "UTF-8")
    .bean("randomMath").filter(groovy("Integer.parseInt(body) %2 == 0; ")).to("activemq:0Num").end()
    .to("activemq:jiNum");

from(uri: "jetty:http://0.0.0.0:3333/toD").id("toD发送组件").convertBodyTo(String.class, charset: "UTF-8").transform(groovy(expression: "" +
    "result = body"
)) .toD(uri: "${header.toUrl}" + "?bridgeEndpoint=true", cacheSize: 1)
    ;

//
//
//
from(uri: "jetty:http://0.0.0.0:3334/toD")
    .id("toD接收返回组件").convertBodyTo(String.class, charset: "UTF-8").to("log:ss").setBody(simple(value: "hello World"+ ++count))
    ;
```

# Groovy组件(语言)



与其说groovy是camel的一个组件倒不如说他是groovy的一种嵌入语言来的更精确,我们可以在很多组件中嵌入groovy(或者simple)语言,通过groovy我们可以做很多事情,例如数据格式转换,动态处理消息等

假设在范勇庆的交互服务中需要一个消息头,我们需要将这个消息头从xml转成json(反一下或者对中间某个节点进行处理),并且这件事是动态的,那我们后台写死的组件就不适应了,这时候我们就可以用groovy来帮助 we 实现这个需求

```
def xml = new XmlSlurper();
def header = headers['name'];
def body = body;"
def input = xml.parseText(header)
def s = input.MessageHeader.ServiceNo.text();
def h
=JXUtils.xmlToJson(header).getJSONObject('EsbMessage').getJSONObject('MessageHeader').get('ServiceNo').toString();
input.MessageHeader.ServiceNo = 'fixed';
def k = input.MessageHeader.ServiceNo.text();
result = 'json解析:'+h + '-----xml解析'+ s+ '-----修改后的内容:'+k";
```

# simple组件(语言)



据作者说simple表达式在创建时只是一种非常简单的语言，但后因为简单好用变得更加强。它主要旨在成为一种非常小且简单的语言，用于评估表达式(*evaluating Expressions*)和谓词(*Predicates*)，而无需任何新的依赖项或对Xpath的了解；所以它非常适合在骆驼中进行使用。当您在 Camel 路由中需要一些基于表达式的脚本时，simple基本能覆盖95%的使用场景。

**Simple可以在很多地方使用比如  
setbody,setheader,filter等等**

在学习的过程中我在很多地方使用了simple,我就不举单独的路由了,把一些可以用simple的地方都一一列举一下,总的来说就是用 `{}` 把你需要的东西包起来,就像用jquery一样.

`.transform().simple( "Bonjour ${header.me}" )` 从消息头中获取me  
`.choice()..when(simple( "${body} == '1'" ))` 作为Predicate使用,可用作判断条件  
`.toD( "http://0.0.0.0:8899/api/test/together2?parameter=${body}" )` 在toD组件中使用,在参数中拼接url  
`.setHeader( "h1" , simple( "${in.headers.name}" ))` 在seyHeader组件中使用,拼装消息头  
`.setBody(simple( "${in.headers.name}" ))` 在seyBody组件中使用,拼装消息体

当然我们也可以写一些简单的小逻辑

```
<when id="ldap_response_adding">  
<simple>${body} contains '沈' and ${body} contains '超琦'</simple>  
</when>
```

# 配合IDEA断点解析

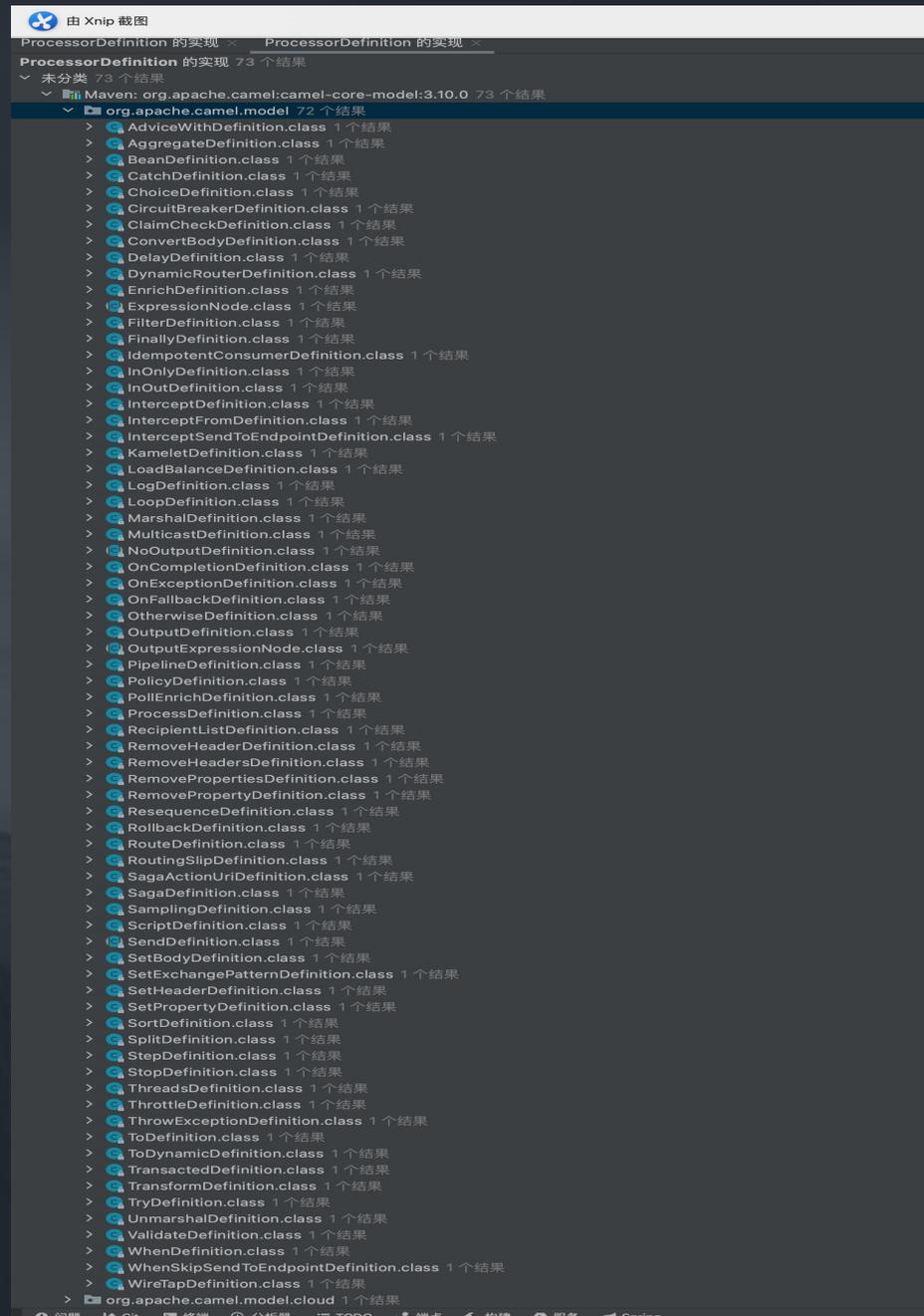
## 源码解读-ProcessorDefinition

processorDefinition作为一个泛型抽象类，其是几乎所有Route定义节点类的基类（诸如from()对应的FromDefinition，to()对应的ToDefinition，setProperty()对应的 SetPropertyDefinition等等都是直接或间接继承自该类），因此其内部所声明的方法，重要性自然是不言而喻。本文接下来我们就来一起大致粗略的看一下其内部原理以及构造



# ProcessorDefinition的实现类

我们已经可以看到一些熟悉的面孔——  
OnExceptionDefinition,  
OnCompletionDefinition, ChoiceDefinition等等。  
甚至还有与Route直接相关的RouteDefinition。



# addOutput方法

---

addOutput作为processDefin中的一个重要的方法,下面就让我们通过idea的断点来着重关注choiceRoute这个略显复杂的路由来管中窥豹,这个routeDefine对象包含了所有的路由路线,对我们理解camel的部分设计理念有很多的帮助.



# 总结

---



总体来说大家对camel的理解基本还处于一种能用部分组件解决单一或简单问题的状况,对于复杂路由大家都还处于一种难以驾驭的情况,后续在正式启用后肯定会出现很多问题,相对于某成熟的流程引擎我们还有很多路要走,比如异常处理,流程追踪,debug调试,内存处理等情况,鉴于camel国内资料较少,我们很多时候都要借助外网,但是这就造成了很多其他的成本,但是相信在大家的共同努力下肯定能够渡过难关,把流程引擎很好的落地.



谢谢大家

Thank you