# Camel快速介绍

## 前言

### 本文档的目的

本文档是为了提供给之前未接触过Apache Camel框架的读者入门使用，主要是希望阅读完本文档之后，能对Apache Camel框架能有一个基础性的认知，并能够通过文档中的相关样例代码，掌握Apache Camel的基础用法，能自己动手，使用camel框架做一些基础性的功能，如：定时从第三方系统获取数据并处理入库

### 作者

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 作者 | 日期 | 备注 |
| V1.0 |  | 2020.07.31 | 初始版本 |

### 需要注意的点

本文档中，所有涉及到JSON操作的代码，都是使用的alibaba的JSON库fastjson，如对该库的用法不熟悉的，请自行百度，本文不再详细介绍该库的用法。

如果阅读完本文档之后，有对Apache Camel框架不清楚的地方，可以咨询作者，并提供相关的改进意见。

## Camel的基本概念

### Camel是什么

Camel是一个基于Enterprise Integration Pattern(企业整合模式，简称EIP)的开源框架。EIP定义了一些不同应用系统之间的消息传输模型，包括常见的Point-to-Point，Pub/Sub模型。

Camel框架的核心是一个路由引擎，或者更确切地说是一个路由引擎构建器。它允许您定义自己的路由规则，决定从哪个源接收消息，并确定如何处理这些消息并将其发送到其他目标。

Camel提供更高层次的抽象，使您可以使用相同的API与各种系统进行交互，而不管系统使用的协议或数据类型如何。 Camel中的组件提供了针对不同协议和数据类型的API的特定实现，开箱即用。

下图是官方提供的对camel的总结

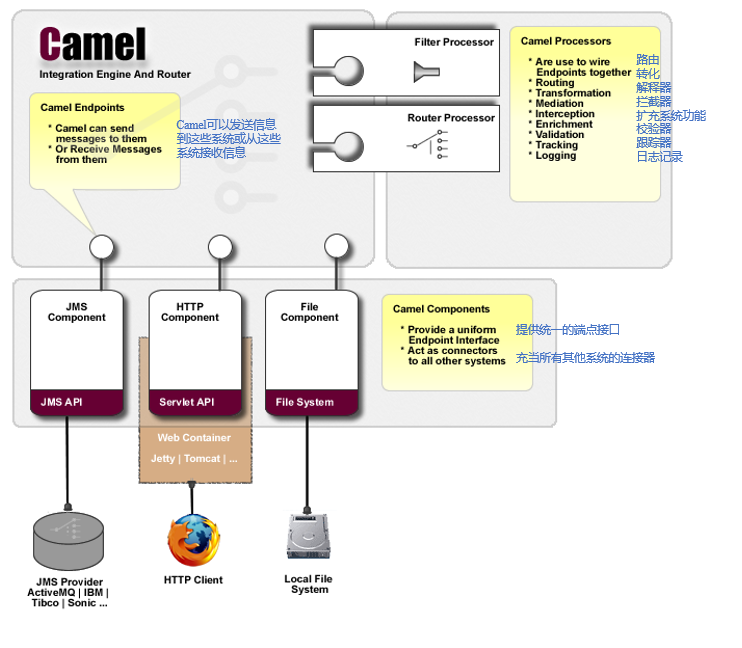


图1-a-1

路由，帮我们从这个端点连接到另外一个端点

转换器，数据格式，消息体内容的转换

解释器，在对数据处理之前，先解析成需要的格式

扩充系统的功能，通过Camel集成其他系统的功能进来

校验器，先对数据进行校验再处理

跟踪器，跟踪数据/请求的流向情况

日志，通过Camel做统一的日志记录

### Camel的使用场景

Camel可以使用的场景很多，这里不一一列举，主要列举几个我们最有可能会用到的

本节中，如果对列举的代码中所涉及到的元素不理解的，可以看下面的章节：[Camel的元素](#_Camel的元素)

#### 从外部系统获取数据

在我们的程序运行过程中，很多时候会需要从外部系统(第三方web系统、webservice、第三方数据库等)获取我们所需要的数据，而按照以前的做法，我们都是直接在我们的应用程序中去直接编写对接过程，但是这样会存在问题，就是每来一种外部服务协议(http/webservice/jdbc)，我们就需要对这种协议进行单独开发，工作量会比较大，而使用Camel的话，Camel帮我们内置了绝大部分我们会用到的协议组件，都可以直接开箱即用，能够大大减少我们的开发周期，提高我们的工作效率。

针对上述场景，我们可以这样进行实现(定时从http接口获取数据并存入到本地文件中)：

public class HttpRequestJob {  
 public static void main(String[] args) throws Exception {  
 CamelContext context = new DefaultCamelContext();  
 context.start();  
  
 context.addRoutes(new RouteBuilder() {  
 public void configure() {  
 *//每秒钟执行一遍调度，往"https://www.baidu.com"发送请求，并将请求结果存入到"d:/test/atest.txt"* from("quartz://report?cron=\* \* \* \* \* ?")  
 .to("https://www.baidu.com")  
 .to("file:d:/test/?fileName=atest.txt");  
 }  
 });  
 }  
}

#### 数据格式转换

在和其他系统对接过程中，我们需要向其他系统发送数据，或者接收其他系统的数据，但是一般情况下，两边系统的数据格式都是不太一致的，比如数据字段名称不一致等，这就很容易导致现有的数据接口没办法复用，在这种情况下，我们按以前的做法有三种解决方式：

1.让别的系统按照我们系统的接口规范定制。

2.额外部署一个专门用于接口数据转换的工程。

3.我们的系统升级，提供多种数据格式。

如果是使用Camel，我们可以很容易就能做出来一个转换的处理器，甚至通过动态脚本的方式，系统都不需要重启就可以完成对接。

针对上述场景，我们可以这样进行实现(提供一个接口，这个接口的功能是将另一个http接口的json数据进行转换，添加一个数据总数total字段)：

public class HttpRequestJob {  
 public static void main(String[] args) throws Exception {  
 CamelContext context = new DefaultCamelContext();  
 context.start();  
  
 context.addRoutes(new RouteBuilder() {  
 public void configure() {  
 from("jetty:http://0.0.0.0:88/convert")  
 *//这个接口将返回数据：{"datas":[{"id":1},{"id":2},{"id":3}],"name":"test"}  
 //由于本例是直接通过to("http://xxx")的方式获取http接口数据，所以需要开启bridgeEndpoint参数，实际使用可以通过ProducerTemplate的方式来调用http接口* .to("http://0.0.0.0:99/sourceData?bridgeEndpoint=true")  
 .process(new Processor() {  
 @Override  
 public void process(Exchange exchange) throws Exception {  
 *//获取原始接口返回的数据* String resultData=exchange.getIn().getBody(String.class);  
 JSONObject jsonObject= JSON.*parseObject*(resultData);  
 JSONArray datas=jsonObject.getJSONArray("datas");  
 jsonObject.put("total",datas.size());  
 exchange.getOut().setBody(jsonObject.toJSONString());  
 }  
 });  
  
 *//这是模拟的原始数据服务，不是转换过程的代码，这段代码是提供原始数据服务的* from("jetty:http://0.0.0.0:99/sourceData")  
 .setBody(constant("{\"datas\":[{\"id\":1},{\"id\":2},{\"id\":3}],\"name\":\"test\"}"));  
 }  
 });  
 }  
}

### Camel的元素

#### CamelContext

CamelContext是整个Camel的核心容器，可以把它看作为camel的运行系统，camel的所有东西都需要依托它来进行。

#### Component

Camel的组件，每个组件对应了camel所能支持的协议，当然我们也可以自己自定义组件，来对协议进行更高层次的封装，比如camel提供了基础的HTTP组件，通过它我们可以进行HTTP请求的发送，我们也可以自定义封装比如ekp组件，然后在发送请求时，针对ekp系统来做自动免登，屏蔽在调用ekp的http接口时需要手动登录这个操作。

#### Registry

Camel的注册器，通过CamelContext.getRegistry()获取，然后可以注册一系列bean到camel的容器中，这样可以将一些bean对象传入到组件中，比如我们使用jetty组件时，需要有basic auth，而jetty组件提供了鉴权的入口，我们就可以将我们的鉴权配置通过这个registry传入

Eg:

from("jetty:http://0.0.0.0:8888/basicAuthTest?handlers=securityHandler")  
 .setBody(exchange -> "test result")  
 .to("log:testlog");

定义了一段路由，需要有basic auth鉴权，然后handlers后面对应的参数就是我们在rigistry进行bind的key

#### Route

Camel的路由，路由是Camel的核心抽象。定义路由最简单的方法就是作为一个处理器链

如：from("jetty:http://0.0.0.0:88/test")  
 .process(new Processor() {  
 @Override  
 public void process(Exchange exchange) throws Exception {  
 exchange.getOut().setBody("query=123&p=10");  
 }  
 })  
 .to("http://www.baidu.com");

定义了一个路由，功能是建立一个http监听器，监听本机88端口，

访问:

本机ip:88/test 这个HTTP路径时，会将HTTP请求的body设置为” query=123&p=10”并将http请求转发到”http://www.baidu.com”这个地址

#### Endpoint

Camel的端点，端点是模拟系统可以通过其发送或接收消息的通道路由的末端的Camel抽象。

在我们建立路由时，一般会有from 和 to 来表示从哪里接收消息，将消息发送到哪里去，这两个都需要定义endpoint，我们可以使用代码的方式，定义Endpoint对象的bean，也可以直接使用uri的方式，来定义Endpoint，camel会自动帮我们将uri字符串解析为Endpoint对象。

#### Processor

Camel的处理器，我们一般在这里写我们需要对信息做什么处理

#### Exchange

Camel的消息实体，camel的整个路由过程中，exchange对象都是同一个，我们在处理过程中，都是通过这个exchange对象来传递消息，在上一个processor中，如果我们没有改变exchange中的内容，会继续沿用到上上个processor中的消息数据

#### Consumer

Camel的消费者，我们在定义路由的时候,from定义的就是consumer

当然，我们也可以通过ConsumerTemplate的方式来定义Consumer，我们可以通过这个方式，来自定义一个consumer来监听一个请求，当然使用这个方式的话，这个监听只有一次有效，监听到了一次之后，就不再有效了

#### Producer

Camel的发布者，我们在定义路由的时候，to定义的就是Producer

当然我们也可以通过ConsumerTemplate的方式来定义Producer，我们可以通过这个方式，来自定义一个Producer使用Camel的通信能力发送一个请求并获取请求结果。

#### 元素之间的包含关系

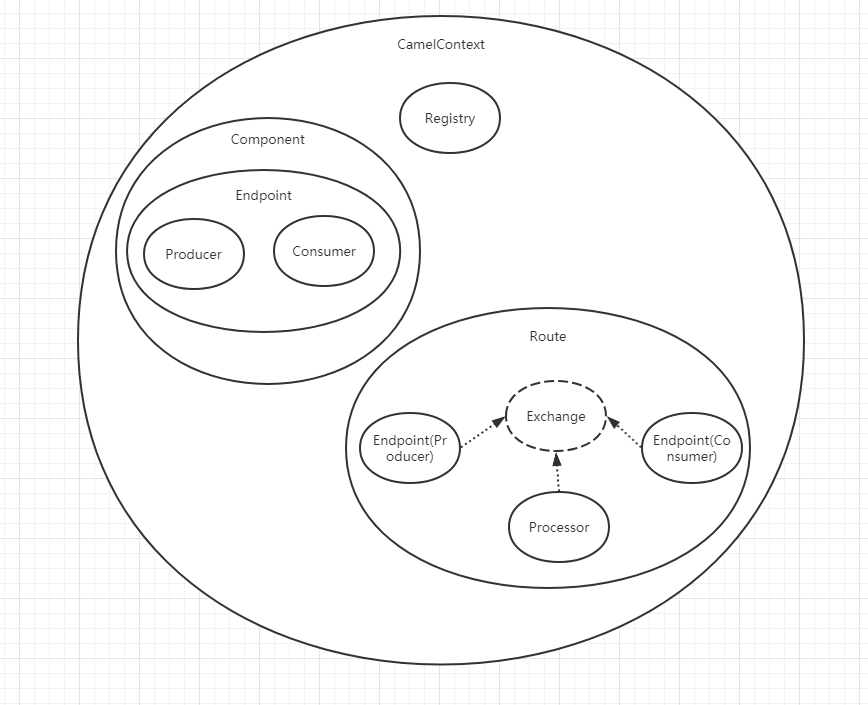


图1-d-1

如上图

CamelContext是整个Camel系统，CamelContext中可以定义Component、Registry、Route

Component中定义有Endpoint

Endpoint中会有Producer和Consumer，分别表示这个Endpoint作为Producer和作为Consumer时，分别应该提供什么功能。

Route会有Endpoint和Processor，Endpoint分别有Producer(to调用时)和Consumer(from调用时) Processor是处理器，用来处理整个Route流转过程中的数据转换等一系列操作，而Exchange是整个Route周期中都会存在的，且整个Route周期都是同一个Exchange，在Endpoint和Processor中都可以对Exchange中的数据进行重新设值

#### 元素的流转关系

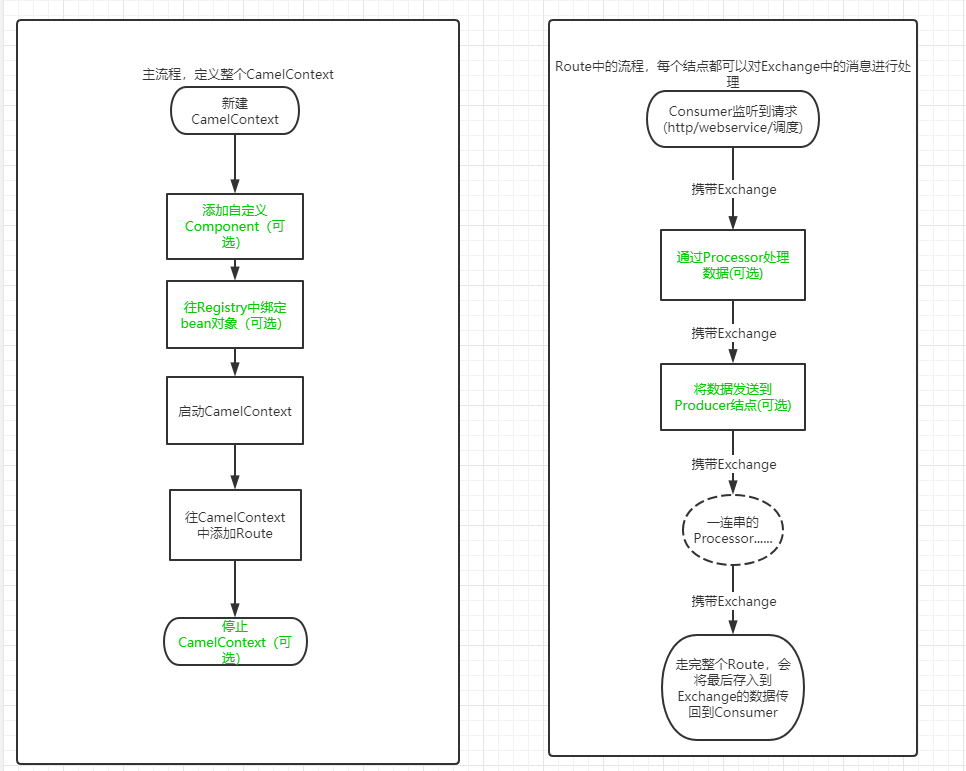


图1-d-2

### Camel的三种定义方式

比如，现在有一个需求，每秒钟打印一行日志，信息主体是”Hello World”

通常来讲，我们会使用如下三种方式来实现，当然，也可以使用其他的方式，这里主要介绍我们常用的如下三种方式

#### 使用纯Java代码的方式

public class HelloWorld\_Java {  
 public static void main(String[] args) throws Exception {  
 DefaultCamelContext camelContext=new DefaultCamelContext();  
 camelContext.start();  
 camelContext.addRoutes(new RouteBuilder() {  
 @Override  
 public void configure() throws Exception {  
 from("quartz://report?cron=\* \* \* \* \* ?")  
 .setBody(constant("Hello World"))  
 .to("log:mainInfo");  
 }  
 });  
 }  
}

#### 使用xml的方式

<routes xmlns="http://camel.apache.org/schema/spring">  
 <route>  
 <from uri="quartz://report?cron=\* \* \* \* \* ?"/>  
 <setBody>  
 <constant>Hello World</constant>  
 </setBody>  
 <to uri="log:mainInfo"></to>  
 </route>  
</routes>

public class HelloWorld\_Xml {  
 public static void main(String[] args) throws Exception {  
 DefaultCamelContext camelContext=new DefaultCamelContext();  
 camelContext.start();  
 *//获取xml内容* InputStream xmlStream=HelloWorld\_Xml.class.getResourceAsStream("/route.xml");  
 byte[] data=new byte[xmlStream.available()];  
 xmlStream.read(data);  
 xmlStream.close();  
 String xml=new String(data);  
 *//通过xml字符串添加route* XMLRoutesDefinitionLoader xmlRoutesDefinitionLoader=new JaxbXMLRoutesDefinitionLoader();  
 RoutesDefinition routesDefinition=xmlRoutesDefinitionLoader.createModelFromXml(camelContext,xml, RoutesDefinition.class);  
 camelContext.addRouteDefinitions(routesDefinition.getRoutes());  
 }

#### 使用Groovy脚本的方式

def addRoute(CamelContext camelContext){  
 camelContext.addRoutes(new RouteBuilder() {  
 @Override  
 void configure() throws Exception {  
 from("quartz://report?cron=\* \* \* \* \* ?")  
 .setBody(constant("Hello World"))  
 .to("log:mainInfo");  
 }  
 })  
}  
  
addRoute(context)

public class HelloWorld\_Groovy {  
 public static void main(String[] args) throws Exception{  
 DefaultCamelContext camelContext=new DefaultCamelContext();  
 camelContext.start();  
 *//获取groovy脚本字符串* InputStream groovyStream=HelloWorld\_Groovy.class.getResourceAsStream("/route.groovy");  
 byte[] data=new byte[groovyStream.available()];  
 groovyStream.read(data);  
 groovyStream.close();  
 String groovy=new String(data);  
 *//执行脚本* GroovyScriptEngineFactory groovyScriptEngineFactory=new GroovyScriptEngineFactory();  
 ScriptEngine engine = groovyScriptEngineFactory.getScriptEngine();  
 Bindings binding=engine.createBindings();  
 binding.put("context",camelContext);  
 engine.eval(groovy,binding);  
 }  
}

## Camel和我们现有做法的区别

#### 使用上的区别

列：

需要提供一个http接口，将传入的一串id（通过逗号分隔），转换成json数组

在不使用camel的情况下(springmvc)：

@RequestMapping("/parseIdsToJsonArray")  
public String parseIdsToJsonArray(@RequestBody String ids){  
 String[] idArr=ids.split(",");  
 JSONArray jsonArray=new JSONArray(Arrays.*asList*(idArr));  
 return jsonArray.toJSONString();  
}

使用camel:

defaultCamelContext.addRoutes(new RouteBuilder() {  
 @Override  
 public void configure() throws Exception {  
 from("jetty:http://0.0.0.0:88/parseIdsToJsonArray")  
 .process(new Processor() {  
 @Override  
 public void process(Exchange exchange) throws Exception {  
 String ids=exchange.getIn().getBody(String.class);  
 String[] idArr=ids.split(",");  
 JSONArray jsonArray=new JSONArray(Arrays.*asList*(idArr));  
 exchange.getOut().setBody(jsonArray.toJSONString());  
 }  
 });  
 }  
});

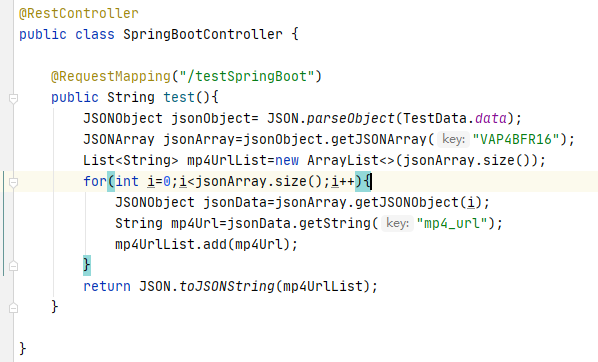
可以看出来，camel中，要提供一个http接口，直接通过一串链式的代码就可以完成，通过这种方式吗，我们天然可以将它和groovy这种动态脚本结合起来，在系统运行时添加对外提供的http接口

#### 性能上的区别

测试Springboot、CamelJava定义路由、CamelGroovy脚本定义路由三种方式的性能区别

请求进入服务后，服务会处理一段固定的json数据，并返回处理结果

测试的服务运行逻辑如下：





注：测试过程是通过本机运行服务，另一台机器运行jmeter测试的，非真实生产环境，有可能会受本机当时电脑运行状态运行，该测试报告仅供参考。

## 附录

### 样例代码

每个样例工程中，都有ReadMe.md，介绍了每个样例所实现的内容，推荐使用idea打开工程

本文档提供了两个版本的camel样例

camel-my-examples.zip对应jdk1.8 camel版本3.4.0

camel-my-examples-jdk1.7.zip 对应jdk1.7 camel版本2.17.7



官网样例：<https://github.com/apache/camel-examples/>

### 相关链接

#### 官网

<https://camel.apache.org/>

#### github

<https://github.com/apache/camel>

#### 官网内置component信息（在使用内置的component的时候，可以到这里查看具体的参数有哪些，对应的参数传递类型等信息）

<https://camel.apache.org/components/latest/index.html>

#### 技术博客

<https://blog.csdn.net/simba_cheng/category_9271875.html>

### 书籍

