Apache Camel

Open-source, message oriented,
rule-based routing and mediation engine

– **Avadhut**

About Me

- I'm Avadhut!
- Working as Senior Integration Engineer
- Open-source enthusiast
- Active community member for couple of open-source projects
- Worked with Integration, Fuse, Camel, Karaf, Kafka and messaging platforms for quite a bit
- Did full production deployments, architecture review and performance tuning for couple of employers and lot of Red Hat customers

** You can find me on:

GitHub: https://github.com/kodtodya

GitLab: https://gitlab.com/kodtodya

LinkedIn: https://www.linkedin.com/in/kodtodya/

Twitter: https://twitter.com/kodtodya

Welcome...!!!

- A warm welcome to Apache Camel Components course..

Course Structure

- Part – 3 : Practicals
  - Basic component handling
  - Important EIP implementation in Camel
- Part – 4 : Advance Camel
  - Threading and transactions
  - Performance tuning

Course Structure

- Part – 3 : Camel Components
  - Direct & Direct-vm
  - SEDA
  - VM
  - Timer
  - Quartz2
  - Log
  - Cache/Ehcache
  - File/File2

- Part – 3 : Camel Components
  - Stream
  - Http/Http4
  - AHC
  - Rest
  - SQL
  - JMS
  - Mail
  - MongoDB

Course Structure

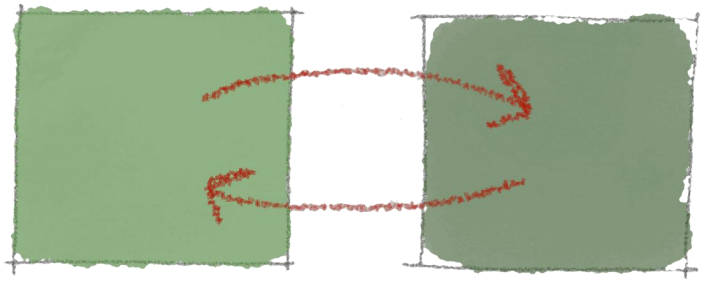
- Part – 3 : EIP implementation
  - Message
  - Message Router
  - Message Endpoint
  - Message Channels
  - Polling Consumers
  - Receipient List
  - Splitter
  - Aggregator
  - Throttler
  - Content Enricher
  - Content based routing

Apache Camel

Components

https://kodtodya.github.io/talks/

Direct Component



- Direct means "*direct*"

URI Format:

direct:someName[?options]

- One of the most simple ways to link your routes together in *same* CamelContext
- Creates synchronous endpoint
- Helps to join routes in synchronous way
- **Drawbacks:
- Direct endpoints can only be accessed by other routes but in same CamelContext
- You can not access direct endpoint in other CamelContext

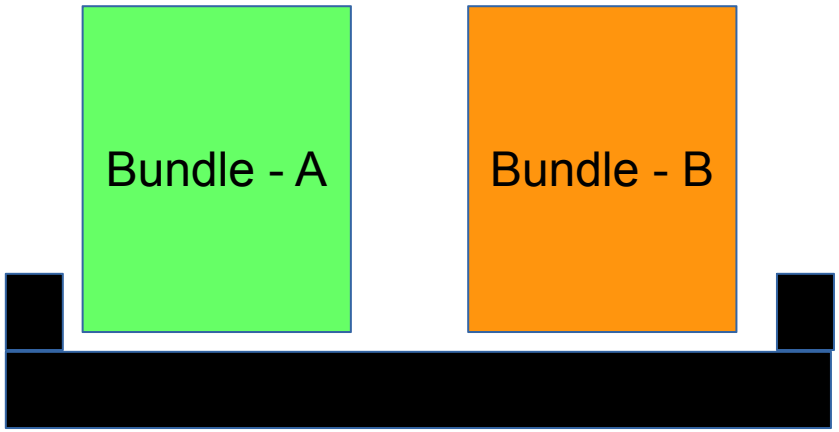
Direct Component

```
from( uri: "file:/home/kodtodya/myFiles")  // receive file
        .to("direct:fileProcessor")        // send to direct endpoint
        .to("Body is now -> ${body}");     // will print "My custom body here..!!!"

//meanwhile
from( uri: "direct:fileProcessor")         //receive from direct endpoint
    .setBody("My custom body here..!!!"); // modify message body
```

- Direct endpoint using to(“direct:fileProcessor”), the route myRoute will be executed in same thread as a first route and response message will be returned

Direct-Vm Component



- Direct-Vm supports communication across CamelContext instances

- One of the most simple ways to link your routes together in *different* CamelContext in *same JVM*
- Creates synchronous endpoint
- Helps to join routes in synchronous way
- Authorizes to develop applications using Transactions - Tx

URI Format:

direct-vm:someName[?options]

SEDA Component

URI Format:

seda:someName[?options]

- Provides asynchronous SEDA behavior
- Allows you to join routes together using a simple queue
- Queues are only visible within a single CamelContext
- This component does not implement any kind of persistence or recovery
- If the VM terminates while messages are yet to be processed
- SEDA creates own buffer to store the incoming messages
- SEDA creates pool of threads to process incoming process
- SEDA provides queue-like functionality, but without the overhead of running an external message broker like ActiveMQ

SEDA Component

```
rest("/orders").post()                  // receive an order via REST
    .to("seda:processOrder");           // send to the seda queue
    .setBody("Thanks for ordering!"); // create a REST response

from("seda:processOrder")               // receive from the seda queue
    .log("Processing an order...")
    .to("file:orders/out");
```

- Messages are exchanged on a BlockingQueue and consumers are invoked in a separate thread from the producer
- When a message is sent to a SEDA endpoint, it is stored in a basic in-memory queue, and control is returned back to the calling route immediately
- Then, independently, a SEDA consumer picks up the message from the queue, and begins processing it

VM Component

- Provides asynchronous SEDA behavior, exchanging messages on a BlockingQueue and invoking consumers in a separate thread pool
- Exchanging messages on a BlockingQueue and invoking consumers in a separate thread pool
- VM supports communication across CamelContext instances
- VM is an extension to the SEDA component

URI Format:

vm:queueName[?options]

```
from("direct:foo").to("vm:bar");

from("vm:bar?concurrentConsumers=5").to("file://output");
```


Drawback of SEDA and VM

- The biggest drawback of using in-memory messaging like SEDA and VM is that if the application crashes, there's a big chance you'll lose all your messages.
- It doesn't matter, if your application does not impact if message goes missing. But think back to the order processing example at the top of this article. If an order gets lost during a server outage, this potentially means lost business. (Uh-oh.) So, choose wisely.
- There isn't a hard and fast rule. The right solution always depends on your use case.

Summary

**Synchronous→ request/response **Asynchronous → fire & forget

Component	Type	Within same CamelContext	Within same JVM
Direct	Synchronous	Yes	--
Direct-VM	Synchronous	Yes	Yes
SEDA	Asynchronous	Yes	--
VM	Asynchronous	Yes	Yes

Timer Component

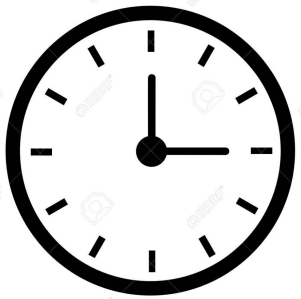
- Timer component is used to generate message exchanges when a timer fires You can only consume events from this endpoint
- Where *timerName* is the name of the Timer object, which is created and shared across endpoints
- So if you use the same name for all your timer endpoints, only one Timer object and thread will be used.

URI Format:

timer:timerName[?options]

```
<route>
  <from uri="timer://foo?fixedRate=true&amp;period=60000"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```


Quartz2 Component



- Quartz component provides a scheduled delivery of messages using the Quartz Scheduler
- Each endpoint represents a different timer (in Quartz terms, a Trigger and JobDetail)
- The component uses either a CronTrigger or a SimpleTrigger.
- If no cron expression is provided, the component uses a simple trigger. If no groupName is provided, the quartz component uses the Camel group name.

Quartz2 Component

URI Format:

quartz2://timerName?options
quartz2://groupName/timerName?options
quartz2://groupName/timerName?cron=expression
quartz2://timerName?cron=expression

```
<bean id="quartz2" class="org.apache.camel.component.quartz2.QuartzComponent">
    <property name="startDelayedSeconds" value="5"/>
</bean>
```


Log Component

- Log component logs message exchanges to the underlying logging mechanism.
- The default logger logs every exchange (regular logging). But Camel also ships with the Throughput logger, which is used whenever the groupSize option is specified.

URI Format:

log:loggingCategory[?options]

Ehcache Component

- Ehcache component enables you to perform caching operations using Ehcache 3 as the Cache Implementation.
- Supports producer and event based consumer endpoints
- Cache consumer is an event based consumer and can be used to listen and respond to specific cache activities.

URI Format:

ehcache://cacheName[?options]

```
CacheManager manager = CacheManagerBuilder.newCacheManager(new XmlConfiguration("ehcache.xml"));
EhcacheIdempotentRepository repo = new EhcacheIdempotentRepository(manager, "idempotent-cache");

from("direct:in")
    .idempotentConsumer(header("messageId"), idempotentRepo)
    .to("mock:out");
```


File2 Component

- File component provides access to file systems, allowing files to be processed by any other Camel Components or messages from other components to be saved to disk.
- You can read & write file using file producer and file consumer
- You can even write batch to process data
- You can use expressions like File language

URI Format:

file:directoryName[?options]
or
file:///directoryName[?options]

```
from("file://inbox?move=.done").to("bean:handleOrder");
```


Stream Component

- Stream component provides access to the System.in, System.out and System.err streams as well as allowing streaming of file and URL.
- In addition, the file and url endpoint URIs are also supported with stream.
  - stream:file?fileName=/foo/bar.txt stream:url[?options]
- If the stream:header URI is specified, the stream header is used to find the stream to write to. This option is available only for stream producers (that is, it cannot appear in from()).
- Stream component supports either String or byte[] for writing to streams. Just add either String or byte[] content to the message.in.body

Stream Component

URI Format:

stream:in[?options]
stream:out[?options]
stream:err[?options]
stream:header[?options]

stream:file?fileName=/foo/bar.txt
stream:url[?options]

```
from("stream:file?fileName=/server/logs/server.log&scanStream=true&scanStreamDelay=1000")
  .to("bean:logService?method=parseLogLine");
```


HTTP4 Component

- HTTP4 component provides HTTP based endpoints for calling external HTTP resources (as a client to call external servers using HTTP).
- camel-http4 uses Apache HttpClient 4.x while camel-http uses Apache HttpClient 3.x.
- Camel will store the HTTP response from the external server on the OUT body. All headers from the IN message will be copied to the OUT message, so headers are preserved during routing. Additionally Camel will add the HTTP response headers as well to the OUT message headers.

HTTP4 Component

URI Format:

http4:hostname[:port][/resourceUri][?options]


```
from("direct:start")
        .to("http4://oldhost");
```

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD, constant(org.apache.camel.component.http4.HttpMethods.POST))
  .to("http4://www.google.com")
  .to("mock:results");
```


AHC Component

- Provides HTTP based endpoints for consuming external HTTP resources (as a client to call external servers using HTTP)
- Uses the Async Http Client library
- Default ports are 80 for HTTP and 443 for HTTPS
- AsyncHttpClient client uses a AsyncHttpClientConfig to configure the client

URI Format:

ahc:http://hostname[:port][/resourceUri][?options]

ahc:https://hostname[:port][/resourceUri][?options]

REST Component

- Allows to define REST endpoints using the Rest DSL and plugin to other Camel components as the REST transport
- The path and uriTemplate option is defined using a REST syntax where you define the REST context path using support for parameters
- Camel offers a REST styled DSL which can be used with Java or XML
- The intention is to allow end users to define REST services using a REST style with verbs such as GET, POST, DELETE etc.

URI Format:

rest://method:path[:uriTemplate]?[options]

SQL Component

- Component allows you to work with databases using JDBC queries
- In case of SQL, the query is a property of the endpoint and it uses message payload as parameters passed to the query
- This component uses spring-jdbc behind the scenes for the actual SQL handling
- SQL Component can be used as a Transactional Client
- SQL component also supports:
  - A JDBC based repository for the Idempotent Consumer EIP pattern
  - A JDBC based repository for the Aggregator EIP pattern

URI Format:

```
sql:select * from table where id=:#name_of_the_parameter order by name[?options]
```


JMS Component

URI Format:

jms:queue:<QUEUE_NAME>

jms:topic:<TOPIC_NAME>

- Allows messages to be sent to (or consumed from) a JMS Queue or Topic
- Uses Spring's JMS support for declarative transactions, including Spring's JmsTemplate for sending and a MessageListenerContainer for consuming
- Reuses Spring-2's JmsTemplate for sending messages
- This is not ideal for use in a non-J2EE container and typically requires some caching in the JMS provider to avoid poor performance

Mail Component

- Provides access to Email via Spring's Mail support and the underlying JavaMail system
- You can set your custom resolver on the MailComponent instance or on the MailEndpoint instance
- POP3 has some limitations and end users are encouraged to use IMAP if possible
- Uses SUN JavaMail, which only trusts certificates issued by well known Certificate Authorities (the default JVM trust configuration)

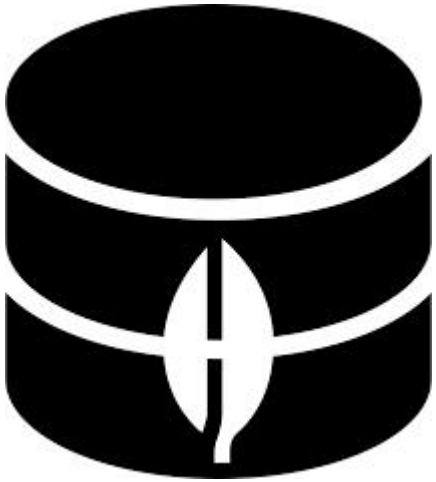
URI Format:

```
smtp://[username@]host[:port][?options]

pop3://[username@]host[:port][?options]

imap://[username@]host[:port][?options]
```


Mongo Component



- MongoDB is a very popular NoSQL solution and the camel-mongodb component integrates Camel with MongoDB allowing you to interact with MongoDB collections both as a producer (performing operations on the collection) and as a consumer (consuming documents from a MongoDB collection)
- MongoDB revolves around the concepts of documents (not as is office documents, but rather hierarchical data defined in JSON/BSON) and collections

URI Format:

```
mongodb:connectionBean?database=databaseName&collection=collectionName&operation=operationName[&moreOptions...]
```


Let's revise – Part - 3

- Part – 3 : Camel Components
  - Direct & Direct-vm
  - SEDA
  - VM
  - Timer
  - Quartz2
  - Log
  - Cache/Ehcache
  - File/File2

- Part – 3 : Camel Components
  - Stream
  - Http/Http4
  - AHC
  - Rest
  - SQL
  - JMS
  - Mail
  - MongoDB

Questions ?

https://kodtodya.github.io/talks/

Thank you..!!!

LinkedIn, GitHub, GitLab, Twitter: @kodtodya

https://kodtodya.github.io/talks/