Apache Camel

Enterprise Integration op zijn best

In dit artikel wil ik jullie graag vertellen over een framework dat het verdient om meer onder de aandacht te komen. Apache Camel is één van mijn favoriete open source projecten in de Java wereld. Apache Camel maakt integratie tussen verschillende applicaties met verschillende protocollen en verschillende berichtformaten gemakkelijk. In dit artikel een korte introductie in het framework.

Toen ik begon met programmeren, was het heel normaal dat we applicaties schreven die volledig stand-alone waren. Dit was nog voor de komst van internet. Die tijd is nu voorbij en je kan je bijna niet meer voorstellen dat je zulke applicaties nog zou schrijven. Bijna elke applicatie integreert met meerdere andere applicaties (services), zelf geschreven services en services afgenomen van derden. Deze services kunnen sterk variëren in het soort protocol dat ze gebruiken. Te denken valt aan onder andere protocollen, zoals HTTP, SOAP, REST, FTP, JMS, JMX, LDAP, JDBC en nog veel, heel veel meer. Dit wordt gedaan om diverse redenen, zoals logging, het bijhouden van statistieken, het aanroepen van benodigde functionaliteit, het verzamelen van benodigde data en dergelijke. Buiten de verschillende applicaties en protocollen wordt ook nog gecommuniceerd in zeer verschillende dataformaten. Zo bestaan vele standaarden, zoals XML, CSV, JSON, PDF, SQL etc. Natuurlijk heb je ook nog de maatwerk formaten. De mogelijkheden zijn dan ook ongekend. Dit is ons dagelijks werk en het valt niet meer weg te denken.

Service Oriented Architecture (SOA) dat de laatste tien jaar veel gevolg heeft gekend en waar nu de MicroService architectuur uit voortvloeit, geeft een sterke indicatie hoe tegenwoordig tegen softwareontwikkeling wordt aangekeken.Aangezien dit ons dagelijks werk is, zou het niet heel moeilijk moeten zijn om deze uitdagingen op te lossen. Op zichzelf zijn de meeste van de integratie oplossingen niet heel moeilijk te implementeren. Het zit hem echter in de hoeveelheid en de herhaalbaarheid van vele van deze integratie oplossingen. Je bent eigenlijk elke keer weer het wiel aan het uitvinden. De code wordt al snel onoverzichtelijk en moeilijk te begrijpen voor andere programmeurs.

Gregor Hohpe en Bobby Woolf merkten op dat veel integratie uitdagingen en hun oplossingen vrijwel gelijk zijn. Zij hebben daar in 2004 het boek '*Enterprise Integration Patterns*' (EIP) over geschreven. Het boek beschrijft de meest voorkomende Integratie patterns en manieren om deze uit te werken. Ze geven niet de concrete uitwerkingen, omdat elk probleem weer net anders is.

Dus wat is Camel nu eigenlijk en waarom zou ik het moeten/willen gebruiken? Camel is een framework dat alle Enterprise Integration Patterns van het hierboven genoemde boek implementeert. Je kunt, als je de EIP's binnen je applicatie geïdentificeerd hebt, deze op een overzichtelijke en declaratieve manier implementeren met behulp van Camel. Je kan dit doen met bijvoorbeeld Java, Scala, XML en Groovy met behulp van Domain Specific Languages (DSL) hiervoor ontwikkeld.

In dit artikel zal alleen ingegaan worden op de Java DSL. Bijna alle technologieën die je je kunt voorstellen, zijn beschikbaar en het is heel gemakkelijk om custom componenten zelf te implementeren. Wellicht is dit het beste te illustreren met een voorbeeld (zie **Listing 1**).

Deze **CustomFormatRoute** class doet in een paar regels code erg veel. Camel werkt met zogenaamde Routes en deze Route monitored een "/inbox" folder op bestanden. Elke keer als een nieuwe bestand in die folder wordt geplaatst, gaat deze Route aan het werk en wordt de body van het geplaatste bestand gelogd. Daarna wordt deze met een **customFormatToCsv** processor getransformeerd naar een CSV bestand. Aangezien het hier om een maatwerk formaat gaat, moet je de converter ook zelf schrijven. Camel maakt dit gemakkelijk. In dit geval met behulp van een spring component dat via de Spring Dependency Injection is geïnjecteerd (zie **Listing 2**).

Ivo Woltring is een Software Engineer en Codesmith bij JTech Ordina.

CAMEL IS EEN FRAMEWORK DAT ALLE ENTERPRISE INTEGRATION PATTERNS IMPLEMENTEERT

Na de conversie wordt het CSV formaat gelogd en vervolgens naar een jms queue genaamd 'csv' gestuurd. Hierna wordt de door Camel zelf geleverde converter gebruikt om het CSV formaat te unmarshallen naar Java. Het csv component van Camel unmarshalled csv naar een List van Lists (**.csv()**). Dit is voldoende om daarna met een simpele adapter deze te mappen naar een Order class met behulp van een bean die in dit geval direct in de route wordt geïnjecteerd door middel van bean name reference (**.bean("csvToOrder", "map")**), zie **Listing 3**.

De Order class heeft methods om met behulp van Jackson zichzelf te mappen naar JSON (zie **Listing 4** op de volgende pagina).

Dit wordt weer gelogd en als bestand in de "/outbox" folder met de originele filename van het begin van de Route, maar nu aangevuld met de extensie ".json". Vervolgens wordt het bericht weer getransformeerd naar een Order door middel van een method reference op de Order class. Deze Order wordt gemarshalled naar XML met behulp van het camel-jaxb component. Dit wordt weer gelogd en naar een ftp site gestuurd waarvan de credentials en endpoint uit de spring application.yml of application.properties gehaald worden. Deze hele route blijft dit doen, zolang de applicatie blijft draaien.

Al met al een krachtig staaltje werk in amper 15 functionele regels code. Het is ook nog eens declaratief, waardoor het goed te volgen is, zelfs voor minder technische teamleden.

Hoe werkt Camel?

Camel werkt met een paar basis concepten. Routes, Componenten en Endpoints. Alles binnen Camel vindt plaats in één of meerdere Routes. Een route bestaat uit from(...).to(...) combinaties. De routes krijgen hun bestemmingen door Endpoints te definiëren in de vorm van URIs

```
package nl.ivonet.route.eip.messaging_systems;
//imports here

@Component
public class CustomFormatRoute extends RouteBuilder {
  private final CustomFormatToCsvProcessor customFormatToCsv;

  @Autowired
  public CustomFormatRoute(final CustomFormatToCsvProcessor customFormatToCsv) {
  this.customFormatToCsv = customFormatToCsvProcessor;
  }

  @Override
  public void configure() throws Exception {
  from("file:///inbox/")
   .log("Custom formatted file:\n${body}")
   .process(this.customFormatToCsv)
   .log("Csv formatted:\n${body}")
   .to("jms:queue:csv")
   .unmarshal()
   .csv()
   .bean("csvToOrder", "map")
   .log("Json formatted:\n${body}")
   .to("file:///outbox?fileName=${header.CamelFileName}.json")
   .transform(method(Order.class, "fromJson"))
   .marshal()
   .jaxb()
   .log("Xml marshalled:\n${body}")
   .to("ftp://{{ftp.user.name}}:{{ftp.user.password}}@{{ftp.host}}");
  }
}
```

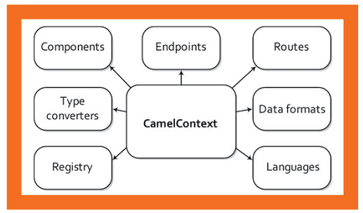
Listing 1

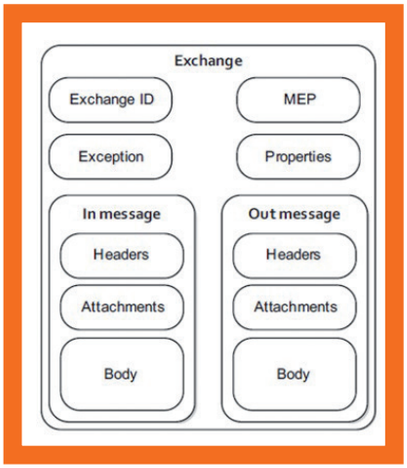
```
package nl.ivonet.route.eip.messaging_systems;
//imports here

@Component
public class CustomFormatToCsvProcessor implements Processor {

   @Override
   public void process(final Exchange exchange) throws Exception {
       final String body = exchange.getIn().getBody(String.class);
       //process custom formatted "body" to csv here
       exchange.getOut().setBody(/*nieuwe output*/);
   }
}
```

Listing 2

```
package nl.ivonet.route.eip.messaging_systems;
//imports here

@Component
public final class CsvToOrder {
   public String map(final List<List<String>> csv) {
       final Order order = new Order();
       for (final List<String> record : csv) {
           //process to csv here
       }
       return order.asJson();
   }
}
```

Listing 3





Camel zorgt ervoor dat de data binnen de route altijd een Exchange object is waardoor elk component op dezelfde manier om kan gaan met het bericht. Ook zelf geschreven componenten.

Het voorbeeld dat hierboven is beschreven, illustreert een wat meer conventionele uitdaging die we vaak in ons dagelijks werk meemaken. Camel kan echter ook de nieuwere communicatie uitdagingen aan zoals bijvoorbeeld integratie met Twitter (zie **Listing 5**).

Deze route zal op interval zoeken naar tweets met de tag #metoo en de body van het Exchange bericht loggen. Twitter legt limieten op, op het aantal API calls dat gedaan mag worden binnen een bepaalde tijdseenheid. Daar is voor het gemak in dit voorbeeld even geen rekening mee gehouden.

Kan ik er ook mee spelen?

Het antwoord hierop is volmondig: ja. Als je in eerste instantie zo snel mogelijk kennis wil maken met dit framework (zonder je zorgen te hoeven maken over hoe je alles moet bootstrappen), dan is het het makkelijkst om naar **https://start.spring.io/** te gaan en daar een project te genereren met minimaal het *Web* en *Apache Camel* opgenomen als dependencies. Genereer het project en spelen maar!

Als je hier dieper in wil duiken, dan is het boek "*Camel in Action, Second Edition*" door Claus Ibsen een aanrader en ook natuurlijk de Apache Camel website. Een gedegen kennis van de Enterprise Integration Patterns zal ook een hoop inzicht geven in wat Camel allemaal voor ons kan doen en waarom het zo goed zou zijn als iedere programmeur dit framework beter zou leren kennen. ■

```
package nl.ivonet.route.eip.messaging_systems;
//imports here

@Data
public class OrderLine {
   private String orderId;
   private LocalDateTime dateTime;
   private Address address;
   private Integer numberOfItems;
}
package nl.ivonet.route.eip.messaging_systems;
//imports here

@Data
@XmlRootElement
public class Order {
   private List<OrderLine> orderLine;

   public void add(final OrderLine orderLine) {
       if (this.orderLine == null) {
           this.orderLine = new ArrayList<>();
       }
       this.orderLine.add(orderLine);
   }

   public String asJson() {
       final ObjectMapper mapper = new ObjectMapper();
       mapper.findAndRegisterModules();
       try {
           return mapper.writeValueAsString(this);
       } catch (final JsonProcessingException e) {
           throw new IllegalStateException(e);
       }
   }
}
```

Listing 4

```
package nl.ivonet.camel_demo_twitter;
//imports hier
/**
* Configureer deze properties in de spring application.properties of application.yml
* de keys zijn aan te maken via https://apps.twitter.com en dan zal spring-boot er voor
* zorgen dat dit auto geconfigureerd wordt
* camel.component.twitter-search.consumer-key=
* camel.component.twitter-search.consumer-secret=
* camel.component.twitter-search.access-token=
* camel.component.twitter-search.access-token-secret=
* camel.component.twitter-search.enabled=
* camel.component.twitter-search.resolve-property-placeholders=
*/
@Component
public class TwitterDemoRoute extends RouteBuilder {
   @Override
   public void configure() throws Exception {
       from("twitter-search:keywords=#metoo")
               .log("\n===\n${body}\n===");
   }
}
```

Listing 5

SOURCES

https://en.wikipedia.org/wiki/Apache_Camel
http://camel.apache.org/
http://camel.apache.org/transport.html
http://www.enterpriseintegrationpatterns.com/
Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions door Gregor Hohpe en Bobby Woolfe
https://github.com/IvoNet/camel-demo
Camel in Action, Second Edition door Claus Ibsen