Miskolci Egyetem

Általános Informatika Tanszék

Adatátalakítás

az **Apache Camel** használatával

Konyha József

Miskolc, 2013

Tartalomjegyzék

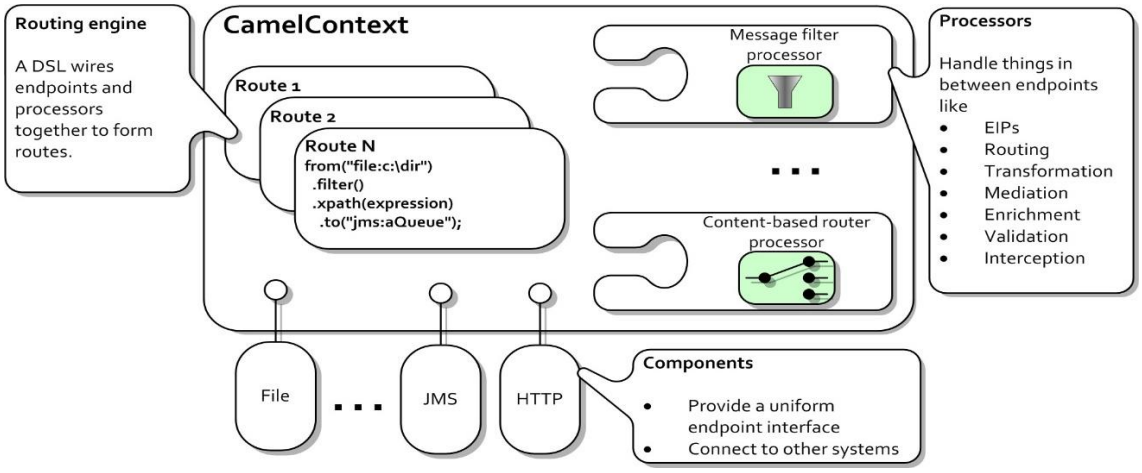
1. Bevezetés

Az Apache Camel egy nyílt forráskódú Java keretrendszer, amely arra összpontosít, hogy az integráció könnyebb és egyszerűbben elérhető legyen a fejlesztők számára. Teszi ezt azáltal, hogy:

- konkrét implementációt biztosít az összes széles körben használt vállalati integrációs mintához (EIP)
- kapcsolatot biztosít a legtöbb fejlesztési API-hoz és szállítási réteghez
- Könnyen használható domain specifikus nyelveket (DSL) biztosít a vállalati integrációs minták és a szállítás összekapcsolásához

Az 1. ábra szemlélteti a három felsorolt pont hogyan értelmezhető a Camel felépítésében. A Camel működésének megértéséhez feltétlenül szükséges megismerni a következő építőköveit:

- Komponensek (Components),
- Végpontok (Endpoints),
- Feldolgozó egységek (Processors) , és a
- Domian specifikus nyelvek (DSL).



1. ábra. A Camel felépítése

A komponensek úgynevezett kiterjesztési pontok, melyek segítségével kapcsolódásokat adhatunk külső rendszerekhez. A Camel magja nagyon kicsi, így a függőségek száma alacsonyan tartható, elősegítve a beágyazhatóságot, stb..., ennek köszönhetően mindössze 13 lényeges alkotóelemet tartalmaz. Több mint 80 külső komponens van a magon kívül. Ezen komponensek listájának kibővítéséhez a Camel végpont (Endpoint) interfészeket biztosít. Az URI-k használatával az üzenetek küldése és fogadása hasonló módon megy végbe a végpontokon. Például, ahhoz hogy üzenetet fogadjunk egy JMS üzenetsorból (AQueue), majd ezek után az üzenetet fájlba írjuk, egyszerűen használhatjuk a következő URI-kat:

- "jms:aQueue",
- "file:/tmp".

A feldolgozó egységek az üzenetek módosításához és közvetítésére használhatóak a végpontok között. Minden vállalati integrációs minta definiál legalább egy vagy több feldolgozó egységet.

A feldolgozó egységek és a végpontok összekapcsolásához a Camel többszörös domain specifikus nyelveket (DSL) definiál. Ezeket a kapcsolatokat akár XML formátumban is létrehozhatjuk. Egy példa DSL definiálására:

```
„from ("file:/tmp").to("jms:aQueue");”
```

A fenti példában definiáltunk egy routing szabályt, ami a „/tmp” mappában lévő fájlokat a memóriába tölti. Létrehoz egy új JMS üzenetet a fájl tartalmával, amit aztán az „aQueue” JMS üzenetsorba küld.

2. Adatok átalakítása

Ezen fejezetben megnézhetjük a Camel hogyan segíti a fejlesztőket az adatátalakítások terén. Elsőként áttekintjük az adatátalakítást Camel ban, majd megnézzük, hogyan lehet a különböző formátumú adatokat konvertálni más formátumokba.

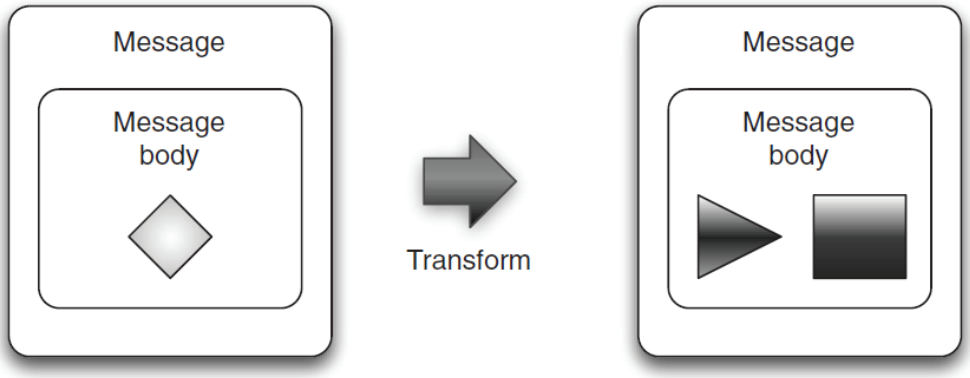
2.1. Adatátalakítási áttekintés

Camelban több lehetőségünk is van adatok átalakítására, amiket hamarosan meg is nézünk. De előtte lássunk egy gyorstalpallót adatátalakításból Camelban.

Az adatok átalakítása tág fogalom, mely kétféle átalakítást foglalhat magába:

- Adat formátumának átalakítása. Pl: CSV -> XML
- Adat tipusának átalakítása. Pl: Integer -> String

Az 2. ábra szemlélteti az elvet, melyben átalakul az üzenet belső felépítése, egyik formátumból egy másikba. Ez az átalakulás magába foglalhatja mind az adatok, mind pedig az adatok típusának átalakulását.



2. ábra. Üzenetek formai és tipus átalakítása

A legtöbb esetben az adatátalakítás amivel szembe nézünk a Camellel az a formai átalakítás, ahol általában két protokoll között szükséges közvetítés. A Camel beépített tipusátalakítóval rendelkezik, ami automatikusan konvertálja a típusokat, ami nagyban megkönnyíti a végfelhasználók munkáját.

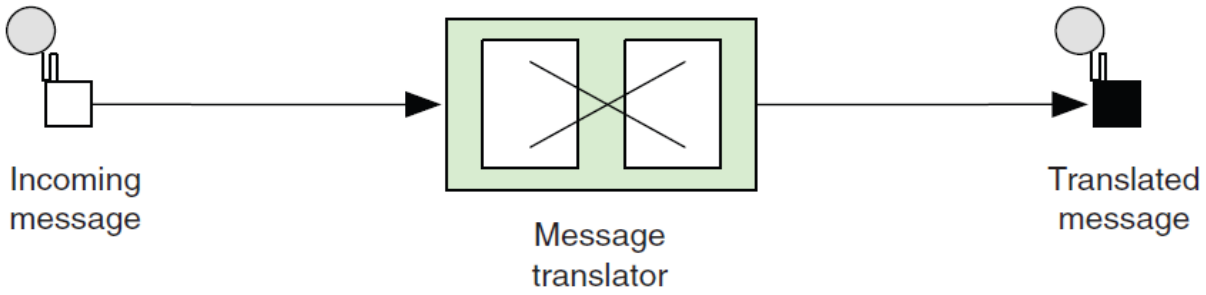
2.2. Adatátalakítás EIP-k használatával Javaban:

Az adatok leképezése az a folyamat, amelyben adatokat alakítunk át két különböző adatmodell között, ez az adatintegráció kulcsmomentuma. Sok meglévő szabvány létezik adatmodellekre, melyeket különböző szervezetek és bizottságok hoztak létre.

A Camel nagy szabadságot nyújt az adatok leképezésére, mert megengedi a Java programozási nyelv használatát. A következőkben látható hogyan lehet az adatokat leképezni a „Feldolgozó egység” (processor) használatával, ami egy Camel API. A Camelban használhatunk Beaneket is, ami egy jó gyakorlat lehet, mert így a Cameltől független adatleképezéseket is létrehozhatunk.

2.2.1. A „Message Translator EIP” használata

Az alábbi ábrán látható a „Message Translator EIP”. A minta átalakít egy üzenetet egy formátumról egy másik formátumra.



3. ábra. Egy üzenet áthaladt az átalakítón

Három különböző módon használhatjuk az EIP-t:

- Feldolgozó egység használatával (Processor)
- Bean használatával
- <transform> használatával

Átalakítás a „feldolgozó egység” használatával

Minden Camel feldolgozó egységnek egy interface (*„org.apache.camel.Processor”)* a következő metódussal:

public void process(Exchange exchange) throws Exception;

A „Processor” egy alacsony szintű API, ahol közvetlenül a Camel kicserélő egységével dolgozhatunk. Teljes hozzáférést ad minden mozdítható Camel részhez a *CamelContext*ből, ami így kicserélhető a *getCamelContext* metódus használatával.

Nézzünk egy példát. Generáljunk napi riportokat egy autójavító műhely újonnan érkezett megrendeléseiből egy CSV formátumú fájlba. A cég saját formátumát használja az új megrendelések felvételére, de szerencsére rendelkeznek egy HTTP szolgáltatással ami egy megadott dátumhoz megadja az

aznapi megrendelések listáját. A feladat tehát egy HTTP szolgáltatás által generált kimenetet CSV formátumúvá formálni majd fájlba menteni a napi riportot.

Mivel szeretnénk egy gyors prototipust létrehozi ezért a Camel „Processor” feldolgozó egységét használjuk:

- Első lépésben le kell töltenünk az adatokat a HTTP szervíztől. Mivel ez String típusú ezért String tipusú adatot kell beállítanunk az exchange-nek,
- A Stringben lévő adatokat szétbontjuk, majd lokális változókban tárolju,
- Felépítjuk a CSV fájlt az előbb lekészített változóinkból,
- majd kicseréljük a tartalmi részünket, az új formátumba öntött (CSV) adatainkkal.

```
1  import org.apache.camel.Exchange;
2  import org.apache.camel.Processor;
3  public class OrderToCsvProcessor implements Processor {
4      public void process(Exchange exchange) throws Exception {
5          String custom = exchange.getIn() .getBody(String.class);
6          String id = custom.substring(0, 9);
7          String customerId = custom.substring(10, 19);
8          String date = custom.substring(20, 29);
9          String items = custom.substring(30);
10         String[] itemIds = items.split("@");
11         StringBuilder csv = new StringBuilder();
12         csv.append(id.trim());
13         csv.append(",").append(date.trim());
14         csv.append(",").append(customerId.trim());
15         for (String item : itemIds) {
16             csv.append(",").append(item.trim());
17         }
18         exchange.getIn().setBody(csv.toString());
19     }
20 }
```

4. ábra. „Processor” használata az adatok átalakításához

Mindezekután az elkészült EIP-t az alábbi ábrán látható módon használhatjuk:

```
1  from("quartz://report?cron=0+0+6+*+*+?")
2  .to("http://riders.com/orders/cmd=received&date=yesterday")
3  .process(new OrderToCsvProcessor())
4  .to("file://riders/orders?fileName=report-${header.Date}.csv");
```

5. ábra. Az elkészült „OrderToCsvProcessor EIP” használata

A fenti példában a Quartz feladatüzemezőt használjuk a művelet elvégéséhez. Az ütemezésen szerint a feladat minden nap 6 órakor fut le. A Job letölti az adatokat, feldolgozza (átdolgozza), majd CSV formátumban fájlba menti.

A *getIn* és *getOut* metódusok használata

A Camel két metódust biztosít az üzenetek letültéséhez: *getIn* és *getOut*. a getIn metódus visszaadja a bejövő üzenetet, míg a getOut metódussal hozzáférhetünk a kimenő üzenetekhez.

Kétféle lehetősége van a felhasználónak eldönteni, hogy melyiket szeretné használni:

- Csak olvasható, ahol a felhasználó csak olvasni tudja az üzenetet
- Írható, ahol a felhasználó átformálhatja az üzenetet.

Az első esetben a getOut használata célszerű. Az elmélet szerint ez helyes, de a gyakorlatban kis hiba csúszhat az elképzelésünkba a getOut használata közben: a beérkező üzenet fejléc része és a csatolmány el fog veszni. Ezt gyakran nem szeretnénk, így saját magunknak kell gondoskodnunk a fejléc és a csatolmányok lemásolásáról és a kimenő üzenethez való csatolásról, ami hosszadalmas lehet.

Egy alternatíva lehet a változások közvetlen beállítására a kimenő üzeneten a getIn metódus használatával, és nem pedig a getOut használatával mint általában. Ezt a gyakorlatot használjuk a könyvben is.

A „Processor” használatának van egy hátránya is: a Camel API-t kell használnunk. A következő fejezetben megnézzük, hogyan használhatunk Beaneket az adatok átalakítására.

2.2.2. Adatátalakítás Beanek használatáva

A Bean-ek használata egy jó gyakorlat, mert megengedi a felhasználónak hogy olyan Java könyvtárakat és kódokat használjon amilyet akar.

A Beaneket előírások nélkül használhatunk Camelban. A Camel hivatkozhat bármilyen Beanre amit választunk, így használhatjuk a már kész Beanjeinket azok újraírása vagy újrafordítása nélkül.

Lássuk akkor, hogyan használhatunk Beaneket a fent ismertetett „Processor” helyett.

```java
public class OrderToCsvBean {
    public static String map(String custom) {
        String id = custom.substring(0, 9);
        String customerId = custom.substring(10, 19);
        String date = custom.substring(20, 29);
        String items = custom.substring(30);
        String[] itemIds = items.split("@");
        StringBuilder csv = new StringBuilder();
        csv.append(id.trim());
        csv.append(",").append(date.trim());
        csv.append(",").append(customerId.trim());
        for (String item : itemIds) {
            csv.append(",").append(item.trim());
        }
        return csv.toString();
    }
}
```

6. ábra. Példa Bean használatára

A feladat ugyan az, mint az előbbiekben, azonban itt kissé máshogy járunk el. Az első említésre méltó kölünbség hogy itt nincs szükségünk semmilyen Camel importra. Ez azt jelenti, hogy a Beanünk teljesen független a Camel API-tól. Egy másik eltérés az előzőekhez képest hogy saját tetszésünk szerint nevezhetjük el metódusunkat. Jelent esetben egy *map* nevű statikus metódust írtunk.

A metódus paraméterlistája a „szerződésünk”, ami azt jelenti hogy az első paraméter (String Custom) az üzenet testét tartalmazza, amit át szeretnénk majd alakítani. A metódusunk visszatérési tipusa szintén String, ami azt jelenti, hogy az átalakított adatunk típusa szintén String lesz. A Camel futás közben kapcsolódik a metódusunkhoz.

Az *OrderToCsvBean* használata Camelban szintúgy, mint az előbbiekben rendkívül egyszerű. Lásd: 7. ábra.

```
from("quartz://report?cron=0+0+6+*+*+?")
.to("http://riders.com/orders/cmd=received&date=yesterday")
.bean(new OrderToCsvBean())
.to("file://riders/orders?fileName=report-${header.Date}.csv");
```

7. ábra. Bean használata Camelban.

2.2.3. Adataátalakítás a Java DSL transform() metódusával

A transform() egy metódus Java DSL-ben ami Camel útirányítónak használható az üzenetek átalakításához. A kifejezések használatának engedélyezésével a transform() metódus nagyfokú rugalmasságot biztosít és a kifejezések közvetlen használatával a DSL-ben néha időt takaríthatunk meg. Nézzük ezt a következő példán.

```
1  from("direct:start")
2  .transform(body().regexReplaceAll("\n", "<br/>"))
3  .to("mock:result");
```

8. ábra. A transform metódus használata.

Tegyük fel hogy HTML dokumentumot szerkesztünk és igény mutatkozik a „\n” újsor karakterek kicserélésére a HTML ben használatos
 tagra. Ezt a műveletet egy, a Camelba beépített kifejezéssel elkészíthetjük. A 8. ábrán látható keresés és csere reguláris kifejezések alapján.

Ahogy a fenti ábrán látható a *transform()* metódus megszólítja a Camel-t, hogy az üzenetet egy kifejezés használatával kell átalakítani. A Camel segíti munkánkat egy mintaépítő szolgáltatással is, hogy a különböző kifejezéseket összefűzzük. Ez láncolt metódushívással történik, ami a Builder pattern alapja. A Builder patternről bővebben a http://en.wikipedia.org/wiki/Builder_pattern címen lehet olvasni.

Az előző mintakódban összefűztük a *body()* és a *regexReplaceAll()* függvényeket. Ez a kifejezés a következőképp értendő: vedd az üzenet tartalmát és a reguláris kifejezésnek megfelelően cseréld ki a „\n” újsor karaktereket a HTML ben használatos
 tagekre.

A Direct komponens

A fenti példában használjuk még a direct komponenest, mint a bemeneti forrást a láncolt műveletek elindításához (*from("direct:start"*)). A Direct komponens közvetlen hívási lehetőséget nyújt a hívó és a hívott között. Így csak a Camel-en belüli kapcsolatok hívhatóak, külső rendszerek nem tudnak közvetlenül üzeneteket küldeni így. A komponens használat a Camelen belül engedélyezett, így pl. közös útvonalak használatához vagy teszteléshez jó.

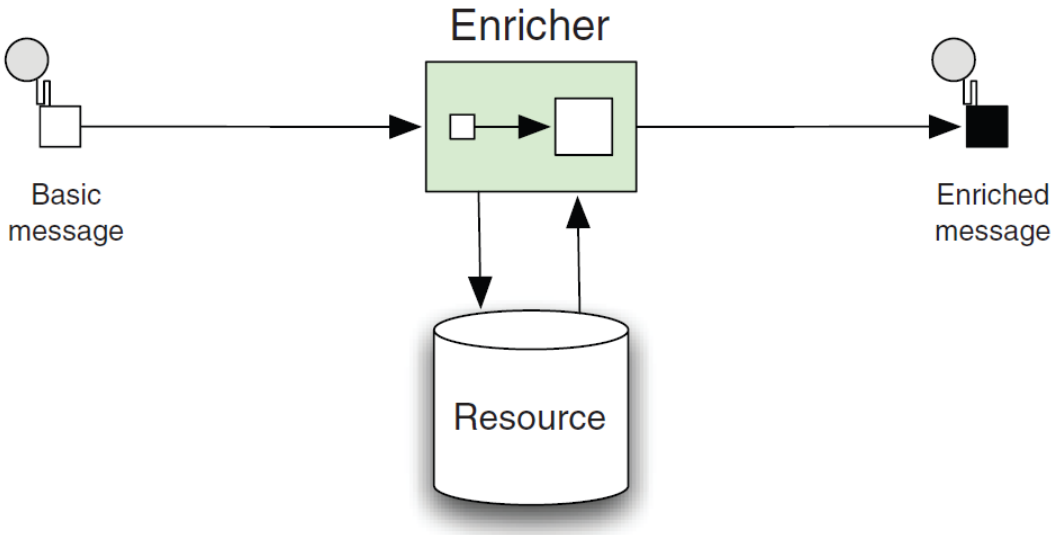
A Camel engedélyezi általunk definiált kifejezések használatát is. Ez hasznos lehet ha teljes felügyeletet szeretnénk és saját Java kódunkat szeretnénk használni. Például tekintsük a következő kódrészletet.

```
1   from("direct:start")
2   .transform(new Expression() {
3       public <T> T evaluate(Exchange exchange, Class<T> type)
4           String body = exchange.getIn().getBody(String.class
5           body = body.replaceAll("\n", "<br/>");
6           body = "<body>" + body + "</body>";
7           return (T) body;
8       }
9   })
10  .to("mock:result");
```

Amint a fenti ábrán látjuk, a kódban inline Camel kifejezést használunk, ami lehetővé teszi számunkra Java kód használatát az *evaluate* metódusban. Ezek után hasonlóan járunk el mint a Camel Processornál, amit fentebb már tárgyaltunk.

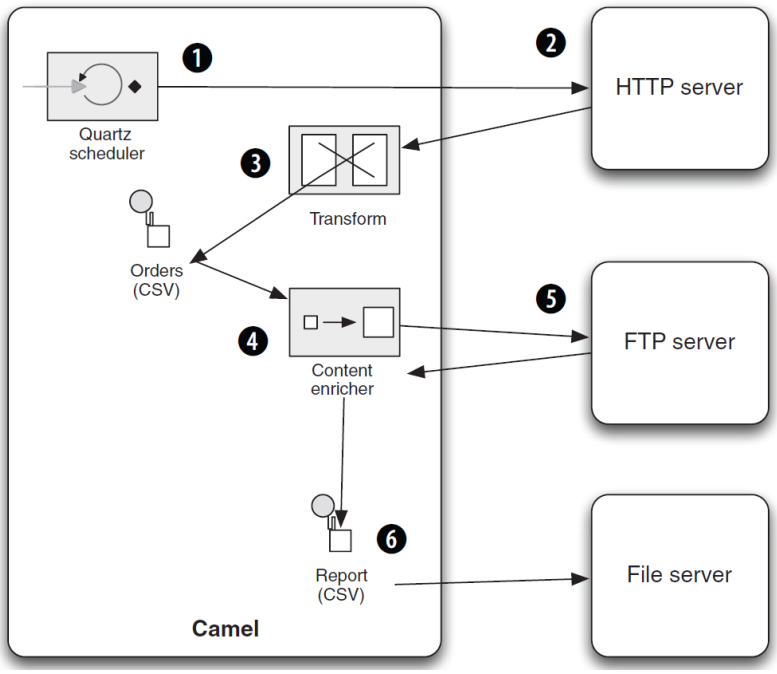
2.2.4. A „Content Enricher EIP” használata

Ebben a fejezetben bemutatásra került a „*Content Enricher EIP*”, aminek segítségével az üzenetek külső forrásból származó adatokkal bővíthetőek a Camel API segítségével.



10. ábra. Az Enricher működésének szemléltetése.

Az Enricher megértésének könnyítéséhez most visszaugrunk az első körben tárgyalt HTML -> CSV adatátalakítónkhoz. Jelenlegi feladatunk a felhalmozott megrendelések, információk egyesítése és hozzácsatolása egy már létező riporthoz.



11. ábra. Az Enricher műveleti lépései.

Ahogyan a fenti ábrán látható a Quartz ütemező minden nap 6 órakkor elindítja a folyamatláncot amit Camelban definiáltunk. Az adatokat a HTTP szerverről általunk definiált formába önti, ami jelen esetben CSV. Ezen a ponton jön az Enricher, ami kiegészíti a szerverről letöltött adatokat, az FTP szerverről származó adatokkal. Majd ezek után az elkészült riportot eltárolja az FTP szerverünkre.

Mielőtt beleásnánk magunkat a művelet forráskódjába, meg kell néznünk, a Camelban, hogyan is működik az Enricher. A Camel két műveletet nyújt a DSLben hogy implementáljuk a mintát:

- pollEnrich – A művelet az üzenethez fűz más forrásból származó adatokat egy fogyasztót (*consumer*.) használva
- enrich – A művelet az üzenethez fűz más forrásból származó adatokat egy gyártót (*producer*) felhasználva.

A különbség a *pollEnrich* és az *enrich* műveletek között az, hogy amíg a *pollEnrich* „*consumer*”-t addig az *enrich* „*producer*”-t használ az adatok forrástól való letöltésére. A különbség nagyon fontos: a fájl komponens akár mind a kettővel is használható, az *enrich* írni fogja az üzenetet mint egy fájlt. A *pollEnrich* olvassa a fájlt mint a forrást. A HTTP komponens csak *enrich*-el használható.

A Camel a *org.apache.camel.processor.AggregationStrategy* interface-t használja az eredmények összefésüléséhez a forrásból származó eredeti üzenettel, valahogy így:

Exchange aggregate(Exchange oldExchange, Exchange newExchange);

Az „*aggregate*” metódus egy callback függvény amit implementálnunk kell. A metódusnak 2 paramétere van: az első tartalmazza az eredeti a második az új, már bővített forrást. A feladatunk az üzenet bővítése Java kód használatával majd visszaadni a bővített, összefűzött üzenetet. Ez így talán egy kicsit kusza lehet, nézzük meg a gyakorlatban:

```
1  from("quartz://report?cron=0+0+6+*+*+?")
2  .to("http://riders.com/orders/cmd=received")
3  .process(new OrderToCSVProcessor())
4  .pollEnrich("ftp://riders.com/orders/?username=rider&password=secret",
5      new AggregationStrategy() {
6      public Exchange aggregate(Exchange oldExchange,
7          Exchange newExchange) {
8          if (newExchange == null) {
9              return oldExchange;
10         }
11         String http = oldExchange.getIn()
12             .getBody(String.class);
13         String ftp = newExchange.getIn()
14             .getBody(String.class);
15         String body = http + "\n" + ftp;
16         oldExchange.getIn().setBody(body);
17         return oldExchange;
18     }
19 })
20 .to("file://riders/orders");
```

12. ábra. pollEnrich a gyakorlatban.

A 12. ábrán látható a pollEnrich használata, a távoli FTP szerververről történő egyéb megrendelések beolvasása és összefűzése az eredeti üzenettel a Camel *AggregationStrategy* segítségével.

A folyamatot a Quartz feladatütemező indítja minden reggel 6 órakkor. A HTTP szerverről leolvasott adatokat, megrendeléseket a processor segítségével CSV formátumúvá alakítjuk.

Ezek után szükségünk van kibővíteni a meglévő adatokat, a távoli szerveren lévő megrendelésekkel. Ezt a feladatot a pollEnrich végzi el, ami beolvassa a távoli fájlt.

Az adatok összefűzéséhez az *AggregationStrategy* metódust használjauk. Első lépésben ellenőrizzük volt e már létrehozott adatunk vagy sem. Ha a *newExchange* értéke null, akkor visszaadjuk az *oldExchange*-t, mivel ebben az esetben nem történt változás. Ekkor a már létező adatok mellé nem kerül új információ. Ha létezik a távoli fájl, akkor a létező adatok után, hozzáfűzzük annak tartalmát az eredeti üzenethez. Ezek után visszadjuk a (már módosított) *oldExchange*-t. Példánk utolsó lépésében a CSV riport fájlba írásához a *file* komponenst használjuk.

A PollEnrich lekérdezésekkel „tölti le” az üzenetek és három időtúllépési módot kínál:

- pollEnrich(timeout = -1) – Elküldi az üzenetet, majd vár amíg válasz nem érkezik. Ebben az esetbena program futása blokkolódik amíg üzenet nem érkezik a kérésre.
- pollEnrich(timeout = 0) – Ha létezik üzenet, akkor azt rögtön küldi. Ellenkező esetben null értékkel tér vissza. Soha sem vár üzenetre, így nem blokkolja a programunk futását. Alapértelmezett esetben ez a beállítás az érvényes.

- pollEnrich(timeout > 0) – Elküldi az üzentet, majd ha nincs létező üzenet, akkor is vár a megadott ideig egy üzenet érkezésére. Ha nincs üzenet a blokkolódás maximum a megadott ideig tarthat.

A gyakorlatban (hacsak nem szükségszerű) nem ajánlott az első beállítás használata. Helyette a másik kettő közül érdemes kiválasztani a nekünk legmegfelelőbbet az adott alkalmazáshoz.

Bővítés az *enrich* használatával

Az *Enrichet* akkor használhatjuk, ha az aktuális üzenet más forrásból való bővítésére, kiegészítésére van szükségünk kérés-válasz típusú üzenetváltás használatáva.

Jó példa lehet az üzenetünk bővítése egy webszolgáltatás hívásából származó válaszüzenettel. De nézzünk egy másik példát, használjuk a Spring XML-t üzenetünk bővítéséhez (13. ábra).

```
1  <bean id="quoteStrategy" class="camelinaction.QuoteStrategy"/>
2  <route>
3      <from uri="activemq:queue:quotes"/>
4      <enrich url="mina:tcp://riders.com:9876?textline=true&sync=true"
5      strategyRef="quoteStrategy"/>
6      <to uri="log:quotes"/>
7  </route>
```

13. ábra. Enrich használata Spring XML-ben.

A fenti példában a Camel *mina* elnevezésű komponensét használjuk a TCP szállítási rétegben, kérés-válasz típusú üzenetkezelésre beállítva szinkron adatkapcsolattal.

Az eredeti üzenet és a távoli szervertől származó üzenet „összefűzésére” egy aggregációs stratégiát kell választanunk. Ezt a *strategyRef* attribútum segítségével tehetjük meg. Ahogy a példában látható az id attrubútumhoz hoz hozzárendeltük a „*quoteStrategy*”-t, ami tartalmazza az *AggregationStrategy* aktuális implementációját, ami használni szeretnénk.

Sokat tanultunk már arról hogyan is alakíthatunk át adatokat a Camel és Java kód használatával. A következő fejezetben megnézzük hogyan is működik ez az XML adatok, dokumentumok világában. A Camelban az XSLT

komponens lesz segítségünkre az XML adatok átalaításához, amivel XSLT stílusokat használhatunk.

2.3. XML adatátalakítás

A Camel 2 lehetőséget nyújt számunkra XML formátumú adatok átalakítására:

- XSLT komponens – adatátalakításhoz használaható, XML formátumú adatok más formába való átalakításához XSLT stílus használatáva.
- XML marshaling – összevonhatunk vagy szétbonthatunk objektumokat XML formátumba vagy XML formátumból készíhetünk objektumokat.

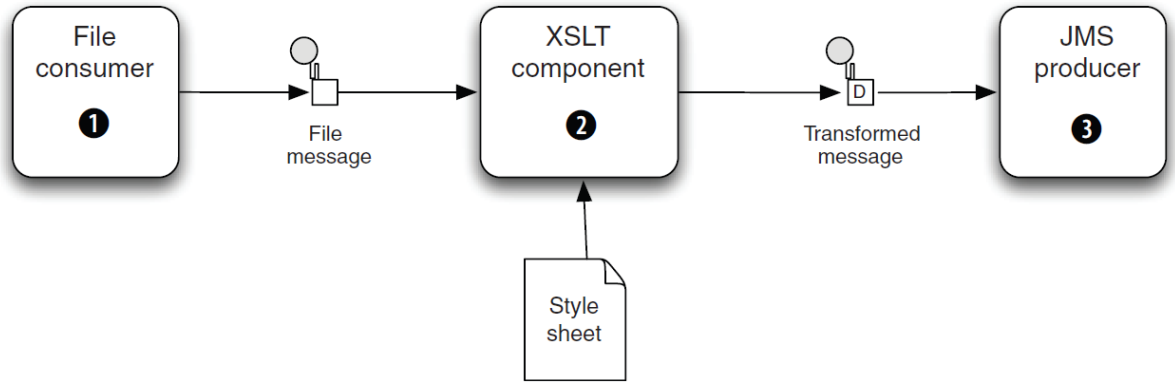
A következő fejezetekben mindkét lehetőséget részletesen áttekintjük, gyakorlati példákkal bemutatjuk.

2.3.1. XML adatátalakítás XSLT-vel

Az XSL Transformations (XSLT) egy deklaratív XML alapú nyelv, amit XML dokumentumok más formátumokba való alakításához használhatunk. Például XSLT-t használhatunk ha egy XML dokumentumból szeretnénk létrehozni egy HTML weblapot, vagy akár ha egy XML dokumentumot át szeretnénk struktúrálni, de a formátuma XML maradna. Az XSLT erős és sokoldalú, viszont elég komplex nyelv, ami miatt időbe telhet a teljes

megismerése és megértése. Érdemes lehet többször átgondolni mit és milyen eszközökkel is szeretnénk elérni, mielőtt az XSLT használata mellett döntünk.

A Camelban az XSLT egy komponens formájában található a camel-spring.jar könyvtárállományban, mivel így kihasználja a spring által nyújtott erőforrásokat. Ez nagyfokú rugalmasságot jelent a stíluslapok betöltésénél mivel a spring különböző helyekről is lehetővé teszi a betöltést úgymint a classpath, egy fájl elérési útvonala vagy ez akár HTTP-ről is lehetséges.



14. ábra. Az XSLT Camelban.

Az XSLT komponens használata rendkívül egyszerű, mivel itt is egy Camelban használatos komponensről beszélünk. A következő példában lévő route bemutatja használatát valamint az elvi felépítése is látható a 14. ábrán.

```
1  from("file://rider/inbox")
2  .to("xslt://camelinaction/transform.xsl")
3  .to("activemq:queue:transformed")
```

15. ábra. XSLT route Camelban.

A fájl fogyasztó beolvassa az új fájlokat és az XSLT komponenshez továbbítja a beolvasott adatokat, ami a stíluslapot felhasználva átalakítja a

beolvasott adatokat. Az átalakítás után az üzenet a JMS szervernek lett továbbküldve, ami a JMS sorba teszi az üzenetet.

Prefix	Example	Description
<none>	xslt://camelinaction/transform.xsl	If no prefix is provided, Camel loads the resource from the classpath
classpath:	xslt://classpath:com/mycompany/transform.xml	Loads the resource from the classpath
file:	xslt://file:/rider/config/transform.xml	Loads the resource from the filesystem
http:	xslt://http://rider.com/styles/transform.xsl	Loads the resource from an URL

16. árba. Az XSLT komponens lehetséges prefixumai

Mint már korábban leírtam a Camel XSLT komponense a Springet használja a stíluslapok betöltésére. A 16. ábrán látható prefixum felsorolásból választhatunk, milyen forrásból kívánjuk betölteni az XML fájlt.

2.3.2. XML adatátalakítás objektummarshallingal

Minden szoftverfejlesztő, aki dolgozott már az XML adatformátummal tudja mekkora kihívást jelent az alacsonyszintű XML API használata, amit a Java nyújt a fejlesztőknek. Az alap Java API helyett szívesebben dolgoznak hagyományos Java objektumokkal és alkalmazzák az összevonást Java objektumok és XMLbeli megfelelőjük között.

A Camelban ezen műveletek használatát a *data formats* komponens hivatott megkönnyíteni. A következőekben röviden bemutatom az XStream és a JAXB adat formátumok használatát.

2.3.2.1. Adatátalakítás az XStreammel

Az XStream egy egyszerű kódkönyvtár, mellyel adatokat szerializálhatunk objektumokból XML fájlokba illetve onnan vissza Java objektumokká. A használatához szükségünk lesz a *camel-xstream.jar* fájlra.

```
1  <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
2      <dataFormats>
3          <xstream id="myXstream"/>
4      </dataFormats>
5      <route>
6          <from uri="direct:foo"/>
7          <marshal ref="myXstream"/>
8          <to uri="activemq:queue:foo"/>
9      </route>
10 </camelContext>
```

17. ábra. az XStream adatátalakítás XML fájlba

Első lépésben deklarálnunk kell az adatformátumot, majd a route tagben megadjuk honnan és hová szeretnénk az adatokat átalakítani.

```
1  from("direct:foo").marshal().xstream().to("uri:activemq:queue:foo");
```

18. ábra. XStream használata Java DSL-ben.

2.3.2.2. Adatátalakítás JAXB-vel

A 3.6. ábrán látható hogyan használjuk a JAXB –t routokban a PurchaseOrder objektum XMLbe való szerializálásához mielőtt egy JMS sorba küldenénk. Ezek után visszaalakítjuk az objektumot XMLből.

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <dataFormats>
        <jaxb id="jaxb" contextPath="camelinaction"/>
    </dataFormats>
    <route>
        <from uri="direct:order"/>
        <marshal ref="jaxb"/>
        <to uri="activemq:queue:order"/>
    </route>
    <route>
        <from uri="activemq:queue:order"/>
        <unmarshal ref="jaxb"/>
        <to uri="direct:doSomething"/>
    </route>
</camelContext>
```

19. ábra. JAXB xml konfiguráció

Első lépésben szükségünk van a JAXB adatformátum deklarálására. Második lépésünk a Route megadása, amivel definiáljuk hogyobjektumból XMLba majd XMLből objektummá szeretnénk alakítani az adatokat.