Architecture distribuée TP 5 et 6

Arnaud Saval
arnaud.saval@gmail.com

Arthur Vaïsse-Lesteven
arthurvaisse@yahoo.fr

Clément Caron
caron.clement@gmail.com

Ce TP présente les EIP (*Enterprise Integration Patterns*) les plus communément rencontrés. Ce système repose sur l'échange de messages (constitués d'un corps « *body* » et d'entêtes « *headers* ») entre des points d'accès (endpoints) « producteurs » et « consommateurs » à travers des « routes ». Nous allons voir les éléments les plus couramment utilisés :

1. 1 producteur / 1 consommateur
2. Routage de messages par leurs contenus
3. Intégration de web services

Tout au long de ce TP, nous allons utiliser Camel (http://camel.apache.com); il s'agit d'une librairie open-source fournissant un support des EIP en java.

1°) Nous allons créer un système consistant à envoyer un message entre un producteur et un consommateur. Créez un nouveau projet avec maven et ajoutez les dépendances suivantes :

```
 <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-core</artifactId>
      <version>2.17.1</version>
</dependency>
<dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.6.4</version>
</dependency>
```

Pour échanger des messages, les producteurs et les consommateurs doivent partager une référence commune. Dans cette première partie, le rôle de référence commune sera assuré par le contexte que partagent à la fois le producteur et le consommateur.

On définit un consommateur comme point d'entrée d'un route grâce au mot-clé « *from* ». Le consommateur est décrit par une URI dont la première partie (schéma) désigne le type du endpoint (direct, log, file, http …).

Par exemple, pour mettre à disposition un endpoint consommateur accessible par la JVM nommée consumer-1, on écrira :

```
from("direct:consumer-1")
```

Pour « continuer la route », il suffit d'ajouter d'autres endoints à la suite. Pour afficher le message reçu dans les logs, on utilisera :

```
from("direct:consumer-1").to("log:affiche-1-log");
```

où `log:affiche-1-log` désigne le endpoint de type `log` et `affiche-1-log` sa catégorie.
Les producteurs sont créés directement depuis le contexte.
Ce producteur peut envoyer des messages à n'importe quel consommateur dont il connaît l'URI avec la commande :

```
pt.sendBody(String endpoint, Object message);
```

Créez le fichier suivant ProducerConsumer.java :

```
package eip;

import org.apache.camel.CamelContext;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.impl.DefaultCamelContext;
import org.apache.log4j.BasicConfigurator;

public class ProducerConsumer {

	public static void main(String[] args) throws Exception {
		// Configure le logger par défaut
		BasicConfigurator.configure();

		// Contexte Camel par défaut
		CamelContext context = new DefaultCamelContext();

		 // Crée une route contenant le consommateur
		RouteBuilder routeBuilder = new RouteBuilder() {

			@Override
			public void configure() throws Exception {
			   // On définit un consommateur 'consumer-1'
			   // qui va écrire le message
			   from("direct:consumer-1").to("log:affiche-1-log");
			}

		};

		// On ajoute la route au contexte
		routeBuilder.addRoutesToCamelContext(context);

		// On démarre le contexte pour activer les routes
		context.start();

		// On crée un producteur
		ProducerTemplate pt = context.createProducerTemplate();
		// qui envoie un message au consommateur 'consumer-1'
		pt.sendBody("direct:consumer-1", "Hello world !");

	}

}
```

1a°) Modifiez le fichier précédent pour envoyer des messages au consommateur à partir de l'entrée standard `System.in` (utilisez la classe `java.util.Scanner`).

1b°) Arrêtez le programme lorsqu'on entre le mot-clé « *exit* ».

2°) Le routage de messages par le contenu se fait soit selon le corps du message soit selon ses entêtes. Dans une route pour vérifier les valeurs des entêtes et choisir une route appropriée, on utilise la syntaxe suivante :

```
from("direct:route1")
            .choice()
                .when(header("entete1").isEqualTo("toto"))
                    .to("direct:routeToto")
                .when(header("entete1").isEqualTo("tata"))
                    .to("direct:routeTata")
                .otherwise()
                    .to("direct:routeTiti");
```

2a°) Ajoutez un second consommateur consumer-2 qui écrit dans le endpoint file:messages.

2b°) Ajoutez une nouvelle route consumer-all qui va envoyer le message à consumer-2 si l'entête **destinataire** contient « écrire » ou à consumer-1 sinon.

2c°) Modifiez le producteur pour qu'il n'appelle que le consommateur consumer-all, et pour qu'il ajoute l'entête **destinataire** avec la valeur « écrire » si le message envoyé commence par la lettre « w ».

3°) Un des principaux intérêts des EIP vient de leur capacité d'intégration de services existants. Nous allons intégrer certaines fonctionnalités du service REST du précédent TP. Pour cela nous avons besoin de gérer le protocole HTTP (http://camel.apache.org/http) avec la dépendance :

```
 <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-http</artifactId>
      <version>2.17.1</version>
</dependency>
```

Pour faire un appel à une adresse en HTTP, vous devez définir la méthode (GET, POST, PUT, DELETE) et ensuite faire un appel au endpoint. L'exemple suivant fait un appel avec la méthode GET au service publié à l'adresse 127.0.0.1 sur le port 8084 :

```
from("direct:Citymanager")
      .setHeader(Exchange.HTTP_METHOD,constant("GET"))
      .to("http://127.0.0.1:8084/all")
      .log("reponse received : ${body}")
```

Lancer votre serveur REST.

3a°) Appelez la méthode pour rechercher un animal par son nom et affichez le résultat.

3b°) Intégrez le service Geonames décrit ici : http://www.geonames.org/export/geonames-search.html pour qu'il retourne la position géographique des zoos où se trouvent les animaux recherchés.

4°) En exécutant plusieurs instances de Tomcat (3 ou plus, suivant les performances de vos machines), faites une fusion des différentes données à l'aide de Camel avant de les renvoyer à l'utilisateur. Il convient que les données identiques doivent être filtrées, afin d'éviter à l'utilisateur de faire le tri manuellement.

5°) En vous inspirant du code à cette adresse : https://gitlab.com/ad2017/TP-EIP, reproduisez les résultats de la question 4 à l'aide de SpringBoot.

Bonus°) Camel propose une fonctionnalité qui permet de se passer de la plupart des lignes de code pour définir les contextes, routes, producteurs et consommateurs. Pour chacune des questions précédentes 1, 2, 3, 4, 5, proposez un fichier XML qui fournit l'ensemble des réponses demandées. L'appel au fichier XML se fait en modifiant la création du CamelContext à l'aide de ces dépendances et de ce morceau de code :

```
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-context</artifactId>
	<version>4.1.5.RELEASE</version>
</dependency>
<dependency>
	<groupId>org.apache.camel</groupId>
	<artifactId>camel-spring</artifactId>
	<version>2.17.1</version>
</dependency>
```

```
FileSystemXmlApplicationContext bean = new
		FileSystemXmlApplicationContext("resources/camelcontext.xml");
CamelContext context = bean.getBean("nomdubean", CamelContext.class);
```

Vous aurez à rendre ce projet en binôme pour le 11 Mai 2017. Tout retard sera pénalisé suivant une règle de calcul exponentielle (1 point, 3 points, 6 points, etc.).

Le projet est à rendre par e-mail sous la forme d'une adresse vers un projet GIT accessible depuis le web. Vous ajouterez votre encadrant pour qu'il puisse avoir accès à votre projet en vous assurant que celui-ci contienne :

1. Les noms et prénoms de votre binôme dans un fichier **AUTEURS** à la racine.
2. Le **code** intégral commenté.
3. Une documentation de votre service dans un fichier **README** à la racine.
4. Un **rapport** sur les problèmes rencontrés et les solutions que vous y avez apportés.

Toute configuration « en dur » (de type *C:\Users\Bobby\etc*), est à proscrire, et sera pénalisé, de même que l'absence de commentaires dans votre code.