UNIVERSITÉ NATIONALE DU VIETNAM À HANOÏ
INSTITUT DE LA FRANCOPHONIE POUR L'INNOVATION

Option : Systèmes Intelligents et Multimédia (SIM)

Promotion : 22 Année académique 2017-2018

Module : ONTOLOGIE ET WEB SEMANTIQUE

Apache Jena

TP D'ONTOLOGIE

Présenté par :
SONFACK SOUNCHIO Serge
OUMAROU ALTINE Mohamadou Aminou

Encadrant : Dr TA Tuan Anh

Table des matières

Table des figures

Liste des tableaux

1 Introduction

[2] Jena est un framework Java pour la création d'applications Web sémantiques. Il fournit des bibliothèques Java étendues pour aider les développeurs à développer du code gérant RDF, RDFS, RDFa, OWL et SPARQL conformément aux recommandations publiées du W3C . Jena comprend un moteur d'inférence basé sur des règles pour effectuer un raisonnement basé sur les ontologies OWL et RDFS, ainsi que diverses stratégies de stockage pour stocker les triples RDF en mémoire ou sur disque.
Jena a toujours été un projet open-source et a été largement utilisé dans une grande variété d'applications web et de démonstrateurs sémantiques.Jena est entrée en incubation avec Apache en novembre 2010 et a obtenu son diplôme en avril 2012.De ce fait notre TP se portera sur Apache jena

2 Problématique

La recherche d'information reste un problème d'actualité même avec le nombre grandissant de moteur de recherche que nous avons aujourd'hui. En effet lorsque nous entrons une information dans un moteur de recherche, généralement nous voulons avoir directement l'information recherché et non une vaste liste d'information ou nous devons rechercher l'information.
Telle est le cas lorsque nous souhaitons savoir dans que pays se trouve une ville ainsi que des information complémentaire.
Pour répondre a cette problématique, nous allons allons utiliser le framework Apache Jena et la base de connaissance Dbpedia.

3 Architecture et environnement

3.1 Architecture

Pour le développement nous nous somme inspirés de l'architecture suivante

Dans se schéma, nous avons la couche Jena qui est chargée de tout ce qui est RDF dans l'application [3]

3.2 RDF

Ressource Description Framework (RDF) est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs méta-données, de façon à permettre le traitement automatique de telles descriptions. Développé par le W3C, RDF est le langage de base du Web sémantique.

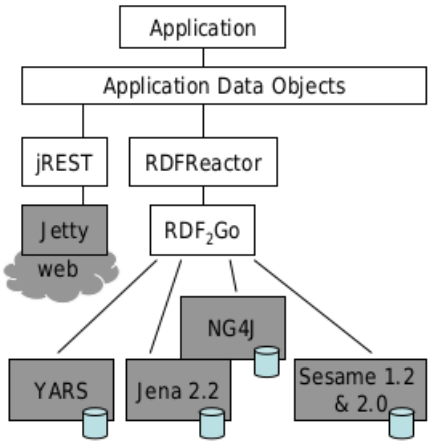

FIGURE 1 – Architecture

3.2.1 ARQ SPARQL

[5] ARQ est une moteur SPARQL RDF Query language qui permet a partir de jena d'interroger les bases de données de type triplestore.

3.3 Triple store

3.3.1 TDB

TDB est un composant de Jena pour le stockage et la requête RDF. Il prend en charge l'ensemble des API Jena. TDB peut être utilisé comme magasin RDF haute performance sur une seule machine. Cette documentation décrit la dernière version, sauf indication contraire.

Voici la documentation de la version standard actuelle de TDB. Ceci est également appelé TDB1 pour le distinguer de la prochaine génération TDB2. Les bases de données TDB1 et TDB2 ne sont pas compatibles.

Un magasin TDB est accessible et géré avec les scripts de ligne de commande fournis et via l'API Jena. Lorsqu'il est utilisé à l'aide de transactions, un ensemble de données TDB est protégé contre la corruption, les terminaisons de processus inattendues et les pannes du système.

Un ensemble de données TDB ne doit être directement accessible que depuis une seule machine virtuelle Java à la fois, sans quoi une corruption des données peut se produire. À partir de la version 1.1.0, TDB inclut une protection automatique contre l'utilisation de plusieurs JVM, ce qui l'empêche dans la plupart des cas.

Si vous souhaitez partager un ensemble de données TDB entre plusieurs applications, utilisez notre composant Fuseki qui fournit un serveur SPARQL pouvant utiliser TDB pour le stockage permanent et fournit les protocoles SPARQL pour la requête, la mise à jour et la mise à jour REST via HTTP.

3.3.2 Fuseki

Apache Jena Fuseki est un serveur SPARQL. Il peut être exécuté en tant que service de système d'exploitation, en tant qu'application Web Java (fichier WAR) et en tant que serveur autonome. Sa sécurité à l'aide d'Apache Shiro et dispose d'une interface utilisateur pour la surveillance et l'administration du serveur.

Il fournit les protocoles SPARQL 1.1 pour la requête et la mise à jour ainsi que le protocole SPARQL Graph Store.

Fuseki est étroitement intégré à TDB pour fournir une couche de stockage persistante transactionnelle robuste et intègre la requête de texte Jena et la requête spatiale Jena. Il peut être utilisé pour fournir le moteur de protocole pour d'autres systèmes de requête et de stockage RDF.

3.4 OWL

3.4.1 Ontology API

[1] [4] Grâce à l'API d'ontologie, Jena vise à fournir une interface de programmation cohérente pour le développement d'applications d'ontologie, indépendamment du langage d'ontologie que vous utilisez dans vos programmes.

L'API d'ontologie Jena est indépendante de la langue : les noms de classe Java ne sont pas spécifiques au langage sous-jacent. Par exemple, la classe Java OntClass peut représenter une classe OWL ou une classe RDFS. Pour représenter les différences entre les différentes représentations, chacun des langages d'ontologie a un profil, qui répertorie les constructions autorisées et les noms des classes et des propriétés.

3.4.2 Inference API

Le sous-système d'inférence Jena est conçu pour permettre le branchement d'une gamme de moteurs d'inférence ou de raisonneurs sur Jena. De tels moteurs sont utilisés pour dériver des assertions RDF supplémentaires issues de certains RDF de base, ainsi que de toute information d'ontologie facultative et des axiomes et règles associés au raisonneur. L'utilisation principale de ce mécanisme consiste à prendre en charge l'utilisation de langages tels que RDFS et OWL, qui permettent d'inférer des faits supplémentaires à partir des données

d'instance et des descriptions de classes. Cependant, la machine est conçue pour être assez générale et, en particulier, elle inclut un moteur de règles générique qui peut être utilisé pour de nombreuses tâches de traitement ou de transformation RDF.

3.5 Utils et developpement

3.5.1 importation de librairie / Maven

Pour faciliter l'importation des différentes librairies de notre projet, nous avons utilises outil de gestion et d'automatisation de production des projets logiciels Java en général et Java EE en particulier. Notre fichier se présente ainsi :

Listing 1 – XML des librairies utilises

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http:
       //maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>vn.edu.ifi.jena</groupId>
   <artifactId>jena</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>

   <name>jena</name>
   <description>Demo project for Spring Boot</description>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.0.4.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.
          sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.
          reporting.outputEncoding>
      <java.version>1.8</java.version>
   </properties>

   <dependencies>
      <dependency>
         <groupId>org.webjars</groupId>
         <artifactId>bootstrap</artifactId>
         <version>3.3.7</version>
      </dependency>

      <dependency>
         <groupId>org.webjars</groupId>
```

```
36          <artifactId>jquery</artifactId>
37          <version>3.1.1</version>
38      </dependency>
39
40      <!-- https://mvnrepository.com/artifact/org.apache.jena/
            jena-arq -->
41      <dependency>
42          <groupId>org.apache.jena</groupId>
43          <artifactId>jena-arq</artifactId>
44          <version>3.0.0</version>
45      </dependency>
46
47      <!-- https://mvnrepository.com/artifact/org.apache.jena/
            jena-core -->
48      <dependency>
49          <groupId>org.apache.jena</groupId>
50          <artifactId>jena-core</artifactId>
51          <version>3.0.0</version>
52      </dependency>
53
54      <!-- https://mvnrepository.com/artifact/org.apache.jena/
            jena -->
55      <dependency>
56          <groupId>org.apache.jena</groupId>
57          <artifactId>jena</artifactId>
58          <version>3.0.0</version>
59          <type>pom</type>
60      </dependency>
61
62      <dependency>
63          <groupId>org.springframework.boot</groupId>
64          <artifactId>spring-boot-starter-web</artifactId>
65      </dependency>
66
67      <dependency>
68          <groupId>org.springframework.boot</groupId>
69          <artifactId>spring-boot-starter-test</artifactId>
70          <scope>test</scope>
71      </dependency>
72  </dependencies>
73
74  <build>
75      <plugins>
76          <plugin>
77              <groupId>org.springframework.boot</groupId>
78              <artifactId>spring-boot-maven-plugin</artifactId>
79          </plugin>
80      </plugins>
81  </build>
82
83
84 </project>
```

3.5.2 Quelques codes Java

Listing 2 – XML des librairies utilises

```
1         package vn.edu.ifi.webjena.vn.edu.ifi.jena;
2
3  import java.io.ByteArrayOutputStream;
4  import java.util.Arrays;
5  import java.util.stream.Collectors;
6
7  import com.google.gson.Gson;
8  import org.apache.jena.query.Query;
9  import org.apache.jena.query.QueryExecutionFactory;
10 import org.apache.jena.query.QueryFactory;
11 import org.apache.jena.query.ResultSet;
12 import org.apache.jena.query.ResultSetFormatter;
13 import org.apache.jena.sparql.engine.http.QueryEngineHTTP;
14
15 public class RdfQuery {
16     public String Query(String ville){
17         System.out.println("avant "+ville);
18         String[] villeTab = ville.split(" ");
19         System.out.println(" taille "+villeTab.length);
20         if(villeTab.length > 1) {
21             for(int i = 0 ; i< villeTab.length; i++) {
22                 villeTab[i] = Character.toUpperCase(villeTab[i
                       ].charAt(0))+villeTab[i].substring(1);
23             }
24             ville = Arrays.stream(villeTab).collect(Collectors.
                   joining("_"));
25         }else {
26             ville = Character.toUpperCase(ville.charAt(0))+
                   ville.substring(1);
27         }
28         ville = ville.trim();
29         System.out.println(ville);
30         // TODO Auto-generated method stub
31         String service = "http://dbpedia.org/sparql";
32         String queryString = "";
33         queryString = "PREFIX rdfs: <http://www.w3.org/2000/01/
               rdf-schema#> SELECT ?label ?abstract " +
34                 "WHERE {" +
35                 "<http://dbpedia.org/resource/"+ville+"> <http
                       ://dbpedia.org/ontology/country> ?y ."+
36                 "?y rdfs:label ?label ."+
37                 "?y <http://dbpedia.org/ontology/abstract> ?
                       abstract ."+
38                 "FILTER (LANG(?label) = 'en') ."+
39                 "FILTER (LANG(?abstract)='en') ."+
40                 "}";
41         //System.out.println(queryString);
42         Query query = QueryFactory.create(queryString);
43         QueryEngineHTTP qexec = QueryExecutionFactory.
               createServiceRequest(service, query);
44         ResultSet results = qexec.execSelect();
45         ByteArrayOutputStream outputStream = new
               ByteArrayOutputStream();
46
47         ResultSetFormatter.outputAsJSON(outputStream, results);
48
49         // and turn that into a String
```

```
50          String json = new String(outputStream.toByteArray());
51
52          Gson gson = new Gson();
53
54          System.out.println("Country: " + gson.toJson(json));
55
56
57          return  json ;
58          //return  gson.toJson(json) ;
59          // and turn that into a String
60          /*String json = new String(outputStream.toByteArray());
61          return  json ;*/
62        /*  System.out.println(json);
63        for ( ; results.hasNext() ; ) {
64             QuerySolution soln = results.nextSolution() ;
65             System.out.println(soln);
66             System.out.println(soln.getLiteral("label"));
67         }*/
68      }
69  }
```

Listing 3 – XML des librairies utilises

```
1       package vn.edu.ifi.webjena.vn.edu.ifi.webjena.controller;
2
3
4   import org.springframework.web.bind.annotation.ModelAttribute;
5   import org.springframework.web.bind.annotation.RequestMapping;
6   import org.springframework.web.bind.annotation.RequestMethod;
7   import org.springframework.web.bind.annotation.RestController;
8   import org.springframework.web.servlet.ModelAndView;
9   import vn.edu.ifi.webjena.vn.edu.ifi.jena.RdfQuery;
10  import vn.edu.ifi.webjena.vn.edu.ifi.webjena.data.Town;
11
12  @RestController
13  public class SearchDbpedia {
14
15      @RequestMapping(value = "/webjena")
16      public  String  search(){
17          System.out.print("ici");
18          return "index";
19      }
20
21      @RequestMapping(value = "/getcountry", method =
            RequestMethod.POST)
22      public ModelAndView getCountry(@ModelAttribute(value = "
            town") Town town){
23          RdfQuery rdf = new RdfQuery();
24          String country;
25          country = rdf.Query(town.getTname());
26
27          ModelAndView modelAndView = new ModelAndView();
28          modelAndView.setViewName("index");
29          modelAndView.addObject("country",country);
30          System.out.print(country);
31
32          return  modelAndView;
33      }
```

```
34 }
```

3.5.3 Spring web

Pour notre application, nous avons utilises le framework Java Spring ; qui est un framework MVC et qui nous offre la possibilité de développer au dessus de notre framework Apache Jena.

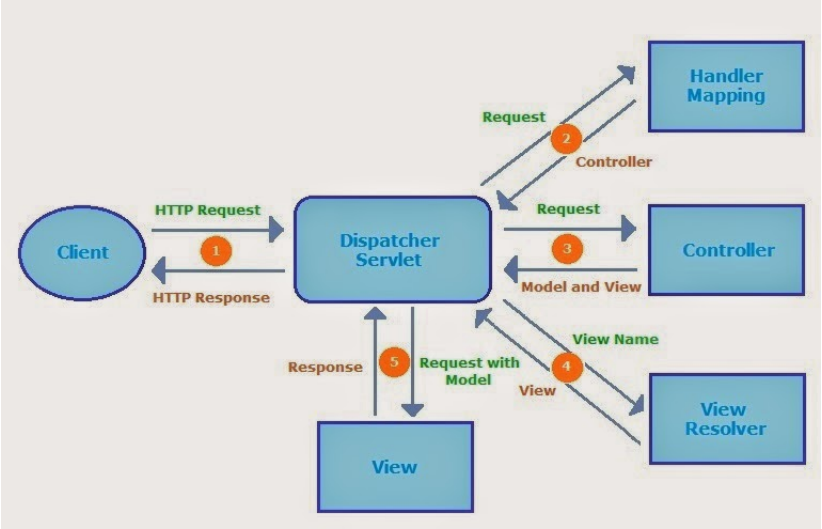

FIGURE 2 – Framework Spring

3.5.4 Interface web

Pour l'interface web de notre application, nous avons utilisés le moteur de template thymleaf et les librairies suivantes :

— Jquery, est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web3. La première version est lancée en janvier 2006 par John Resig.

— bootstrap, Bootstrap est une collection d'outils utile à la création du design (graphisme, animation et interactions avec la page dans le navigateur ... etc. ...) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub.

FIGURE 3 – Jquery et Bootstrap

3.5.5 DBpedia

DBpedia est un projet universitaire et communautaire d'exploration et extraction automatiques de données dérivées de Wikipédia.

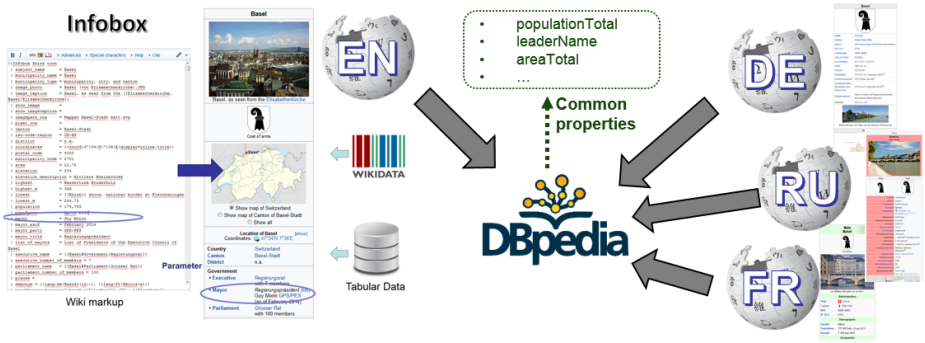

FIGURE 4 – DBpedia

Son principe est de proposer une version structurée et normalisée au format du web sémantique des contenus de Wikipedia. DBpedia vise aussi à relier à Wikipédia (et inversement[pas clair]) des ensembles d'autres données ouvertes provenant du Web des données : DBpedia a été conçu par ses auteurs comme l'un des « noyaux du Web émergent de l'open data »1 (connu également sous le nom de Web des données) et l'un de ses possibles points d'entrée. Ce projet est conduit par l'université de Leipzig, l'université libre de Berlin et l'entreprise OpenLink Software.

4 Test

Dans cette section nous allons afficher les captures des différentes partie de notre application.

1. Page d'accueil

FIGURE 5 – Accueil

2. Recherche et résultat pour **Hanoi** En mettant le nom de la ville de Hanoi on aura comme résultats le pays du nom de Vietnam

(a) recherche (b) résultats

5 Conclusion et perspectives

Par venu au terme de notre travail pratique, sur l'outil de développement d'application de web sémantique Apache Jena ; nous avons étudiés ce que c'est que framework jena ainsi que ses différentes fonctionnalités.
Nous avons pour l'utilisation décidé de faire une application web permettant de connaître le pays d'une ville en interrogeant la base de connaissance DBpedia. Ce travaille nous a permis de comprendre non seulement comment développer une application de web sémantique, mais aussi d'utiliser le langage de requête SPARQL pour récupérer les données de DBpedia.
Pour l'avenir nous comptons utiliser les outils de traitement de langage naturel comme les NER (Named Entity Recognition) pour traiter les requêtes et de permettre l'utilisation plus naturelle de notre application

Ci-dessous le liens github pour acceder au code source de l'application
https ://github.com/sonfack/spring_jena_dbpedia
https ://jena.apache.org/index.html

Références

[1] Matthew Horridge and Sean Bechhofer. The owl api : A java api for owl ontologies. *Semantic Web*, 2(1) :11–21, 2011.

[2] Apache Jena. semantic web framework for java, 2007.

[3] Apache Jena. A free and open source java framework for building semantic web and linked data applications. *Available online : jena. apache. org/(accessed on 28 April 2015)*, 2015.

[4] Shaileshkumar K Patel and Harshad B Bhadka. Semantic web technology and ontology designing for e-learning environments. *Shaileshkumar K. Patel et al,/(IJCSIT) International Journal of Computer Science and Information Technologies*, 6(1) :48–51, 2015.

[5] Matthew Perry, Prateek Jain, and Amit P Sheth. Sparql-st : Extending sparql to support spatiotemporal queries. In *Geospatial semantics and the semantic web*, pages 61–86. Springer, 2011.