

Chapitre 6

Jena

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Présentation

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Présentation

Jena est une bibliothèque de classes Java qui facilite le développement d'applications pour le web sémantique.

- ▶ Manipulation de déclarations RDF.
- ▶ Lecture et écriture RDF/XML, Notation 3.
- ▶ Stockage en mémoire ou sur disque de connaissances RDF.
- ▶ Interpréteur SPARQL.
- ▶ Gestion d'ontologies : RDF-Schema, OWL.

Logiciel libre (licence Apache) développé (initialement) par HP.

<http://jena.apache.org>

Cette présentation est issue du tutoriel Jena :

« An Introduction to RDF and the Jena RDF API »

Présentation - Exemples utilisés dans ce chapitre

Les exemples cités ici permettent de représenter des informations sur des personnes.

VCARD (<ftp://ftp.isi.edu/in-notes/rfc2426.txt>) définit une représentation informatique pour des « cartes de visite » : Nom, prénom, téléphone, e-mail, date de naissance, URL, etc.

Des données VCARD peuvent être représentées en RDF :

<http://www.w3.org/TR/vcard-rdf>

Les ressources et prédicats sont définis dans l'URI :

<http://www.w3.org/2001/vcard-rdf/3.0#>

Exemple

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#">
<rdf:Description rdf:about="http://qqqfoo.com/staff/corky">
  <vCard:FN>Corky Crystal</vCard:FN>
  <vCard:N rdf:parseType="Resource"> <!-- noeud nul -->
    <vCard:Family>Crystal</vCard:Family>
    <vCard:Given>Corky</vCard:Given>
    <vCard:Other>Jacky</vCard:Other>
    <vCard:Prefix>Dr</vCard:Prefix>
  </vCard:N>
  <vCard:BDAY>1980-01-01</vCard:BDAY>
  <vCard:TITLE>Computer Officer Class 3</vCard:TITLE>
  <vCard:ROLE>
    <rdf:Bag>
      <rdf:li>Programmer</rdf:li> <rdf:li>Scientist</rdf:li>
    </rdf:Bag>
  </vCard:ROLE>
</rdf:Description>
</rdf:RDF>
```

Compilation / Exécution

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

- ▶ Jena peut être téléchargé à partir du site <http://jena.apache.org> (Apache jena - Binary distribution)
- ▶ Tous les fichiers *jar* nécessaires à la compilation et à l'utilisation de Jena sont dans `lib`. Ils doivent tous être présents dans le `CLASSPATH` pour la compilation ou l'exécution de programmes utilisant Jena.
- ▶ On utilisera Jena 3.10.0 (requiert Java 8).
- ▶ Consulter la documentation en ligne <https://jena.apache.org/documentation/javadoc/jena/>

Triplets RDF

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Jena fournit des classes pour représenter graphes RDF (**Model**), ressources (**Resource**), propriétés (**Property**), littéraux (**Literal**).
`http://somewhere/JohnSmith vcard:FN "John Smith".`

Exemple (Création d'un triplet)

```
static String personURI="http://somewhere/JohnSmith";
static String fullName="John_Smith";

Model model=ModelFactory.createDefaultModel();

Resource johnSmith=model.createResource(personURI);

johnSmith.addProperty(VCARD.FN, fullName);
```

- ▶ **Création d'un graphe.**

`ModelFactory` est une fabrique de `Model` (graphes).

- ▶ `createDefaultModel` pour un graphe RDF « standard » en mémoire.
- ▶ `createFileModelMaker` pour un graphe RDF sur disque.
- ▶ `createOntologyModel` pour une ontologie (RDF-Schema, etc.)
- ▶ ...

- ▶ **Ajout d'une ressource.**

`createResource` ajoute une ressource à un `Model`.

Retourne une `Resource`.

- ▶ Ajout d'une propriété à une ressource.

`addProperty` ajoute une propriété.

- ▶ `addProperty(Property p, boolean o)`
- ▶ `addProperty(Property p, char o)`
- ▶ `addProperty(Property p, double o)`
- ▶ `addProperty(Property p, float o)`
- ▶ `addProperty(Property p, long o)`
- ▶ `addProperty(Property p, java.lang.Object o)`
- ▶ `addProperty(Property p, RDFNode o)`
- ▶ `addProperty(Property p, java.lang.String o)`
- ▶ `addProperty(Property p, java.lang.String o, java.lang.String l)`

`Resource` et `Literal` sont des sous-classes de `RDFNode`.

Jena contient déjà les propriétés de

- ▶ *RDF* (`org.apache.jena.vocabulary.RDF`)
`RDF.Bag`, `RDF.predicate`
- ▶ *RDF-Schema* (`org.apache.jena.vocabulary.RDFS`)
`RDFS.Class`, `RDFS.subClassOf`
- ▶ *VCARD* (`org.apache.jena.vocabulary.VCARD`)
`VCARD.FN`, `VCARD.BDAY`
- ▶ *Dublin Core* (`org.apache.jena.vocabulary.DC`)
`DC.creator`, `DC.description`

`addProperty` retourne la ressource sujet.

Exemple

```
Resource johnSmith =  
  model.createResource(personURI)  
    .addProperty(VCARD.FN, fullName)  
    .addProperty(VCARD.TITLE, "Officer");
```

Exemple

```
String personURI    = "http://somewhere/JohnSmith";
String givenName    = "John";
String familyName   = "Smith";
String fullName     = givenName + "␣" + familyName;
```

```
Model model = ModelFactory.createDefaultModel();
```

```
Resource johnSmith
= model.createResource(personURI)
  .addProperty(VCARD.FN, fullName)
  .addProperty(VCARD.N,
    model.createResource()
      .addProperty(VCARD.Given, givenName)
      .addProperty(VCARD.Family, familyName));
```

Triplets RDF - Parcours d'un graphe

Un `Model` est un ensemble de `Statement`.

Tout appel à `addProperty` crée un nouveau `Statement`.

La méthode `listStatements()` retourne un itérateur `StmtIterator` qui permet de parcourir tous les `Statement` d'un `Model`.

- ▶ `hasNext` retourne un booléen
- ▶ `nextStatement` retourne le `Statement` suivant.

`Statement` permet d'accéder au sujet (`getSubject`), prédicat (`getPredicate`) et objet (`getObject`).

Exemple

```
StmtIterator iter = model.listStatements();
while (iter.hasNext()) {
    Statement stmt = iter.nextStatement();
    Resource subject = stmt.getSubject();
    Property predicate = stmt.getPredicate();
    RDFNode object = stmt.getObject();

    System.out.print(subject.toString());
    System.out.print("␣" + predicate.toString() + "␣");
    if (object instanceof Resource)
        System.out.print(object.toString());
    else
        System.out.print("␣\" + object.toString() + "\"");
    System.out.println("␣.");
}
```

Exemple (Sortie du programme)

```
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N
anon:14df86:ecc3dee17b:-7fff .
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#
Family "Smith" .
```

Entrées - sorties

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Dans la classe `Model` :

```
Model write(java.io.OutputStream out,  
java.lang.String lang, java.lang.String base)
```

- ▶ `lang` est le format de sortie du modèle : "RDF/XML" (défaut), "RDF/XML-ABBREV" (RDF/XML plus compact), "N3" (Notation 3).
- ▶ `base` : URI de base pour les URI relatives : Enlève `base` à toutes les URI qui commencent par `base`.
`null` → Écrire des URI absolues uniquement.

Sauvegarde

Exemple

```
model.write(System.out);
```

Exemple

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#' >
  <rdf:Description rdf:about='http://somewhere/JohnSmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A0">
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
</rdf:RDF>
```

Sortie rapide...mais les nœuds nuls ne sont pas des nœuds nuls.

Exemple

```
model.write(System.out, "RDF/XML-ABBREV");
```

Exemple

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#">
  <rdf:Description rdf:about="http://somewhere/JohnSmith">
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:parseType="Resource">
      <vcard:Family>Smith</vcard:Family>
      <vcard:Given>John</vcard:Given>
    </vcard:N>
  </rdf:Description>
</rdf:RDF>
```

Chargement

La méthode `read` de la classe `Model` permet de lire un document RDF.

- ▶ `read(java.lang.String url)` Lecture d'un document RDF/XML.
- ▶ `read(java.lang.String url, java.lang.String lang)`
- ▶ `read(java.io.InputStream in, java.lang.String base, java.lang.String lang)` Lecture sur un flux d'entrée d'un document au format passé, en utilisant l'URI de base passée. Si `base` est "" pas de conversion des URI relatives.
 - ▶ Cette méthode peut être utilisée sur un `InputStream` retourné par la méthode `open` de `FileManager`. `FileManager` permet d'ouvrir des documents sans se préoccuper du mode d'accès (`file`, `http`), et en utilisant un accès plus performant (copie locale d'un document distant).

Exemple

```
Model model = ModelFactory.createDefaultModel();  
model.read("file:///home/moi/base.rdf", "RDF/XML");
```

Exemple

```
Model model = ModelFactory.createDefaultModel();  
InputStream in = FileManager.get().open("base.rdf");  
if (in != null)  
    model.read(in, "");
```

Manipulation d'un graphe RDF

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Accès à une ressource / déclaration

`createResource(String uri)` (et `getResource`) de `Model` retournent une `Resource` dont l'URI est passée, ou création d'une ressource.

`getProperty(Property)` de `Resource` retourne un `Statement` (ou `NULL`). **Attention** : En retourne un même si plusieurs.

Exemple

```
String johnSmithURI = "http://somewhere/JohnSmith/";  
Resource vcard=model.createResource(johnSmithURI);  
Resource name=vcard.getProperty(VCARD.N).getResource();
```

`getResource` de `Statement` retourne l'objet ressource, alors que `getString` retourne l'objet littéral.

Sur une `Resource`, `getNamespace` retourne l'espace de nom, et `getLocalName` le nom local.

Parcours des déclarations concernant un prédicat donné, concernant un sujet donné :

méthode `listProperties` de `Resource`

Exemple

```
StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);  
while (iter.hasNext())  
    System.out.println(iter.nextStatement().getObject().toString());
```

Parcours d'un graphe RDF

- ▶ `listStatements` de `Model` retourne un itérateur permettant de parcourir toutes les déclarations.
- ▶ `listStatements(Resource s, Property p, RDFNode o)` de `Model` retourne un itérateur permettant de parcourir certaines déclarations.
- ▶ `listSubjects` retourne un `ResIterator` permet de parcourir toutes les ressources qui sont sujet d'au moins une déclaration.
- ▶ `listSubjectsWithProperty(Property, RDFNode)` pour le parcours des ressources sujet d'un prédicat de la propriété passée, de l'objet passé.
- ▶ `createProperty(string uri)` retourne une `Property` identifiée par l'uri.

- ▶ Méthode générale de sélection : `listStatements(Selector)`. Le `Selector` définit quelles déclarations doivent être parcourues. `SimpleSelector(subject, predicate, object)` permet de sélectionner simplement des déclarations.

Exemple

```
StmtIterator it = model.listStatements(  
    new SimpleSelector(null, VCARD.FN, (RDFNode) null) {  
        public boolean selects(Statement s)  
        {return s.getString().endsWith("Smith");}  
    });
```

- ▶ `union(Model)` de `Model` retourne un nouveau `Model`, fusion du graphe passé avec le graphe courant. Les ressources de même URI sont fusionnées.
- ▶ `intersection(Model)` retourne un nouveau `Model` ne contenant que les déclarations communes.
- ▶ `difference(Model)` retourne un nouveau `Model` qui contient les déclarations qui sont dans le graphe courant mais pas dans le graphe passé.

Conteneurs

Dans `Model` :

- ▶ `createBag`
- ▶ `createAlt`
- ▶ `createSeq`

Ces trois méthodes existent en deux variantes :

- ▶ Sans paramètre. Création d'un conteneur anonyme.
- ▶ Avec paramètre `String`. Création d'un conteneur identifié par l'URI passée.

Sur les conteneurs : `add`, `contains`, `size`, etc.

SPARQL

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Utilisation en ligne de commande

Jena est fourni avec un interpréteur SPARQL pouvant être utilisé en ligne de commande.

Utilisation

- ▶ Définir la variable d'environnement `JENA_HOME`. (anciennes versions)
- ▶ Rendre exécutables les fichiers du répertoire `bin` de Jena si nécessaire.
- ▶ `$JENA_HOME/bin/sparql`
 - ▶ `--graph URL` : graphe de base.
 - ▶ `--namedGraph URL` (optionnel) : graphe nommé.
 - ▶ `--query FICHER` : fichier requête.
 - ▶ Pas de `FROM` / `FROM NAMED` dans le fichier requête dans le cas où `--graph` et `--namedGraph` sont utilisés.

Utilisation depuis un programme Java

- ▶ Créer une `Query` à partir de la méthode `create` de `QueryFactory`, cette méthode prenant comme paramètre une `String` (le texte de la requête SPARQL).
- ▶ Créer un `QueryExecution` à partir de la méthode `create` de `QueryExecutionFactory`, cette méthode prenant comme paramètre une `Query` et un `Model`.
- ▶ Appeler `execSelect` sur le `QueryExecution`, cette méthode retourne un `ResultSet`.
- ▶ Parcourir le `ResultSet` par `hasNext` et `nextSolution`, retournant un `QuerySolution`. Cette classe dispose des méthodes `getResource` et `getLiteral` qui prennent comme paramètre un nom de variable du `SELECT`.
- ▶ Terminer l'exécution du `QueryExecution` par `close`.

Utilisation depuis un programme java

Exemple

```
Model model = FileManager.get().loadModel("file:personnes.n3");
String queryString = "PREFIX_m: <http://exemple.fr/mv#>_SELECT_?
    uri_?nom_?photo_WHERE_{_?uri_m:nom_?nom_.OPTIONAL_{_?uri_m:
    photo_?photo_.}_}";
Query query = QueryFactory.create(queryString) ;
QueryExecution qexec=QueryExecutionFactory.create(query,model);
try {
    ResultSet results = qexec.execSelect() ;
    for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution() ;
        Resource uri = soln.getResource("uri") ;
        Resource photo = soln.getResource("photo") ;
        Literal nom = soln.getLiteral("nom") ;
        System.out.print(uri.toString()+" "+ nom.toString()+" ");
        if (photo != null) System.out.print(photo.toString());
        System.out.println();
    }
} finally { qexec.close(); };
```

RDF-Schema

Présentation

Compilation / Exécution

Triplets RDF

Entrées - sorties

Manipulation d'un graphe RDF

SPARQL

RDF-Schema

Jena permet de gérer des ontologies (RDF-Schema, OWL) et de faire des raisonnements en utilisant les connaissances de l'ontologie.

Une ontologie est un **Model**. Et plus particulièrement un **OntModel**, qui est une spécialisation de **Model** qui contient des méthodes spécifiques aux ontologies :

- ▶ Création d'une classe : **createClass** retourne une **OntClass** (**OntClass** est une spécialisation de **Resource**)
- ▶ Création d'une propriété : **createObjectProperty** retourne une **ObjectProperty** (**ObjectProperty** est une spécialisation de **Resource**)

Création d'une ontologie

Méthode `createOntologyModel` de `ModelFactory` en passant comme paramètre :

- ▶ `OntModelSpec.RDFS_MEM` : Ontologie RDF-Schema en mémoire, aucun raisonnement.
- ▶ `OntModelSpec.RDFS_MEM_RDFS_INF` : Ontologie RDF-Schema en mémoire, raisonnement prenant en compte la transitivité des relations de spécialisation, domaine, co-domaine.
- ▶ OWL, etc.

Souvent, une ontologie est chargée à partir d'un flux : Utilisation des méthodes `read` (idem `Model`).

Les éléments d'une ontologie (`OntClass`, `ObjectProperty`) sont des spécialisations de `OntResource` qui définit les méthodes :

- ▶ `getComment`, `setComment`, ...
- ▶ `getLabel`, `setLabel`, ...
- ▶ `getSeeAlso`, `setSeeAlso`, ...
- ▶ `getIsDefinedBy`, `setIsDefinedBy`, ...

Classes : OntClass

Exemple

```
String exns = "http://www.exemple.com/voc#";  
OntClass veh = m.createClass(exns + "Vehicule");  
OntClass voi = m.createClass(exns + "Voiture");  
voi.addSuperClass(veh);
```

Exemple

```
OntClass cl = m.getOntClass(exns + "Vehicule");  
for (ExtendedIterator i = cl.listSubClasses(); i.hasNext();) {  
    OntClass c = (OntClass) i.next();  
    System.out.print(c.getLocalName() + "␣");  
}
```

- ▶ `hasxxx`, `listxxx`, `setxxx`, `addxxx` :
`listSubClasses`, `listSuperClasses`, `hasSuperClass`, etc.
- ▶ `listInstances` retourne un itérateur sur toutes les ressources de ce `rdf:type`.
- ▶ `listDeclaredProperties` retourne un itérateur sur toutes les propriétés de cette classe. (si paramètre `false` uniquement la classe, si `true` propriétés des superclasses).

Propriétés : ObjectProperty

- ▶ `hasxxx`, `listxxx`, `setxxx`, `addxxx` :
 - ▶ `subProperty`, `superProperty`
 - ▶ `domain` Si plusieurs domaines : intersection des domaines
 - ▶ `range` Si plusieurs `range` : intersection des `range`

Exemple

```
OntClass mot = m.createClass(exns + "Moteur");  
ObjectProperty pcomp = m.createObjectProperty(exns + "composant");  
ObjectProperty pmot = m.createObjectProperty(exns + "moteur");  
pmot.addSuperProperty(pcomp);  
pmot.addDomain(veh);  
pmot.addRange(mot);
```

`createDatatypeProperty` pour les propriétés dont le co-domaine est formé de littéraux.

Habituellement, l'ontologie est stockée dans un (des) fichier, et les connaissances factuelles dans un (des) fichier...

- ▶ Créer un **Modèle** pour l'ontologie.
- ▶ Créer un **Modèle** pour les faits.
- ▶ Créer un modèle (permettant les inférences de type RDF-Schema) qui est l'union des deux.
- ▶ Travailler sur le modèle permettant les inférences.

Ontologies et connaissances factuelles

Exemple (Ontologie : `exemplerdfs/rdfs.rdf`)

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY ex 'file:exemplerdfs/rdfs.rdf#'>
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:ex="&ex;" xmlns:
  ="&ex;">
  <rdf:Description rdf:about="&ex;mum">
    <rdfs:subPropertyOf rdf:resource="&ex;parent"/>
  </rdf:Description>
  <rdf:Description rdf:about="&ex;parent">
    <rdfs:range rdf:resource="&ex;Person"/>
    <rdfs:domain rdf:resource="&ex;Person"/>
  </rdf:Description>
  <rdf:Description rdf:about="&ex;age">
    <rdfs:range rdf:resource="&xsd;integer" />
  </rdf:Description>
```

Exemple (Faits : exemplerrdfs/rdfs.rdf)

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#' >
  <!ENTITY ex 'file:exemplerrdfs/rdfs.rdf#' >
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:ex="&ex;">
  <ex:Teenager rdf:about="http://www.exemple.com/colin">
    <ex:mum rdf:resource="rosy" />
    <ex:age>13</ex:age>
  </ex:Teenager>
</rdf:RDF>
```

Exemple

```
public static void printStatements(Model m, Resource s,
    Property p, Resource o)
{
    for (StmtIterator i=m.listStatements(s,p,o); i.hasNext();)
    {
        Statement stmt = i.nextStatement();
        System.out.println("□-□" + PrintUtil.print(stmt));
    }
}
```

Ontologies et connaissances factuelles

Exemple

```
public static void main(String args[]) {
    Model schema = FileManager.get().loadModel("file:exemplerdfs/rdfs.
        rdf");
    Model data = FileManager.get().loadModel("file:exemplerdfs/rdf.rdf"
        );
    InfModel infmodel = ModelFactory.createRDFSModel(schema, data);

    Resource colin = infmodel.getResource("http://www.exemple.com/colin
        ");
    System.out.println(colin);
    printStatements(infmodel, colin, RDF.type, null);

    Resource Person = infmodel.getResource("file:exemplerdfs/rdfs.rdf#
        Person");
    printStatements(infmodel, Person, RDF.type, null);
}
```

- ▶ **RDF:types de colin :**
 - ▶ **file:exemplerdfs/rdfs.rdf#Teenager**
Déclaration du type dans le fichier RDF.
 - ▶ **file:exemplerdfs/rdfs.rdf#Person**
Parce que :
il a une propriété **mum**, et que la propriété **mum** est une spécialisation de **parent...**
or, le domaine de **parent** est **Person**
 - ▶ **rdfs:Resource**
- ▶ **RDF:types de Person**
 - ▶ **rdfs:Class**
 - ▶ **rdfs:Resource**
- ▶ et **rosy** est une **Person**

Exemple

```
ValidityReport validity = infmodel.validate();
if (validity.isValid())
    System.out.println("OK");
else
{
    System.out.println("Conflicts");
    for (Iterator i = validity.getReports(); i.hasNext(); )
    {
        ValidityReport.Report report =
            (ValidityReport.Report)i.next();
        System.out.println("□-□" + report);
    }
}
```

Conflicts

- Error (dtRange): Property file:exemplerdfs/rdfs.rdf#age has a typed range Datatype[<http://www.w3.org/2001/XMLSchema#integer> -> class java.math.BigInteger] that is not compatible with "13"

Correction du document RDF

```
<ex:age rdf:datatype= "http://www.w3.org/2001/XMLSchema#integer">13</ex:age>
```