Using semantic web resources on a spark cluster

O. Filangi

IGEPP's Metabolic Profiling and Metabolomic Platform (P2M2, Rennes)

SANSA

Big Data RDF Processing and Analytics Stack built on Apache Spark and Apache Jena

- **github : https://github.com/SANSA-Stack/SANSA-Stack**
- **documentation : http://sansa-stack.github.io/SANSA-Stack/**

A free and open source Java framework for building Semantic Web and Linked Data applications.
https://jena.apache.org/

- **RDF API**
- **TRIPLESTORE**
- **API**

SANSA Layers

- RDF : Managing RDF resources
- Query : SPARQL request management
- Inference : Using a reasoner

Sansa IO - Read/Write Layer

- Represent data in multiple formats (e.g. RDD, DataFrames, Dataset)
- Allow transformation among these formats
- Compute dataset statistics and apply functions to URIs,
- literals, subjects, objects → Distributed LODStats

RDD[Triple] / DataSet[Triple]

```
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.riot.Lang

val lang = Lang.NTRIPLES
val path= "...../file.nt"

// 1) RDD
val triplesRDD : RDD[Triple] = spark.rdf(lang)(path)

// 2) Dataset
val triplesDS : Dataset[Triple] = {
import net.sansa_stack.rdf.spark.model.TripleOperations
triplesRDD.toDS()
}
```

`RDFFormat`s and Jena syntax names

The string name traditionally used in `model.write` is mapped to RIOT `RDFFormat` as follows:

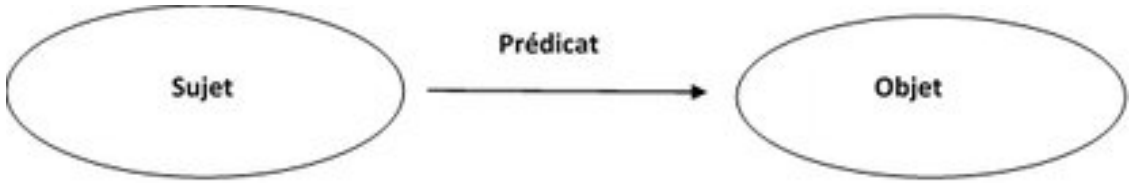
Jena writer name	RIOT RDFFormat
`"TURTLE"`	`TURTLE`
`"TTL"`	`TURTLE`
`"Turtle"`	`TURTLE`
`"N-TRIPLES"`	`NTRIPLES`
`"N-TRIPLE"`	`NTRIPLES`
`"NT"`	`NTRIPLES`
`"JSON-LD"`	`JSONLD`
`"RDF/XML-ABBREV"`	`RDFXML`
`"RDF/XML"`	`RDFXML_PLAIN`
`"N3"`	`N3`
`"RDF/JSON"`	`RDFJSON`

The Jena "Triple"

- getSubject() returning a Resource
- getObject() returning an RDFNode
- getPredicate() returning a Property

Triple
Node : getSubject() Node : getPredicate() Node : getObject()

Node
LiteralLabel : getLiteral() RDFDatatype : getLiteralDatatype() string : getLiteralDatatypeURI() Object: getLiteralValue() string : getLocalName() string : getName() string : getURI() boolean: isBlank() boolean: isConcrete() boolean: isURI() boolean: isLiteral()



Docs :

- https://jena.apache.org/documentation/rdf/index.html
- https://jena.apache.org/tutorials/rdf_api.html
- https://www.javadoc.io/doc/org.apache.jena/jena-core/3.3.0/org/apache/jena/graph/Node.html

DataFrame

```
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.riot.Lang

// 3) Dataframe
val triplesDF : DataFrame = spark.read.rdf(lang)(path)
// or
val triplesDF : DataFrame  = {
import net.sansa_stack.rdf.spark.model.TripleOperations
triplesRDD.toDF()
}

triplesDF.printSchema
root
 |-- s: string (nullable = false)
 |-- p: string (nullable = false)
 |-- o: string (nullable = false)
```

Collect and filter data

```
//DataFrame
triplesDF.select("s","o")
            .filter(triplesDF("s")
            .contains("CHEMBL20"))
            .distinct.collect()

// RDD[Triple]
triplesRDD.filter( triple => triple.getSubject().getURI().contains("CHEMBL20"))
              .map( triple => (triple.getSubject().getURI(),triple.getObject().getURI()) )
              .distinct.collect()

//DataSet[Triple]
triplesDS.filter( triple => triple.getSubject().getURI().contains("CHEMBL20"))
            .map( triple => (triple.getSubject().getURI(),triple.getObject().getURI()) )
            .distinct.collect()
```

RDF Triple Ops - RDD

```scala
// RDF Triple Ops
// ------------
import org.apache.jena.graph.NodeFactory
import net.sansa_stack.rdf.spark.model._

val triplesWithPropertyInchikey = triplesChebi.find(None,Some(NodeFactory.createURI("http://purl.obolibrary.org/obo/chebi/inchikey")), None)
// count -> res11: Long = 138722

println("Number of triples: " + triplesWithPropertyInchikey.distinct.count())
// Number of triples: 138722
println("Number of subjects: " + triplesWithPropertyInchikey.getSubjects.distinct.count())
// Number of subjects: 138722
println("Number of predicates: " + triplesWithPropertyInchikey.getPredicates.distinct.count())
// Number of predicates: 1
println("Number of objects: " + triplesWithPropertyInchikey.getObjects.distinct.count())
// Number of objects: 137814

val subjects = triplesChebi.filterSubjects(_.isURI()).collect.mkString("\n")
val predicates = triplesChebi.filterPredicates(_.isVariable()).collect.mkString("\n")
val objects = triplesChebi.filterObjects(_.isLiteral()).collect.mkString("\n")
```

https://github.com/p2m2/tp-big-data-scala-spark-sansa/blob/main/examples/ChebiSansaTripleOpsStatsQualityAssessment.scala

RDF Triple Ops (KG) - RDD/DataSet

```
def getDatasetTest : Dataset[Triple] = {
  val taxonomyPath = "rdf-files-test/pc_taxonomy_test.ttl"
  val meshPath = "rdf-files-test/mesh_test.nt"
  val assoforumChebiMesh = "rdf-files-test/triples_assos_chebi_mesh_test.ttl"

  spark.rdf(Lang.TURTLE)(taxonomyPath).toDS()
      .union(spark.rdf(Lang.NT)(meshPath).toDS())
      .union(spark.rdf(Lang.TURTLE)(assoforumChebiMesh).toDS())
}
```

- union
- intersection

RDF Statistic - RDD

```
//RDF Statistic
// ------------

import net.sansa_stack.rdf.spark.stats._

val rdf_stats_prop_dist = triplesChebi.statsPropertyUsage()

rdf_stats_prop_dist.take(5)
/*
res13: Array[(org.apache.jena.graph.Node, Int)] =
Array(
 (http://www.w3.org/2000/01/rdf-schema#subClassOf,322738),
 (http://www.geneontology.org/formats/oboInOwl#hasAlternativeId,18515),
 (http://www.geneontology.org/formats/oboInOwl#date,1),
 (http://www.geneontology.org/formats/oboInOwl#hasOBONamespace,161792),
 (http://www.w3.org/2000/01/rdf-schema#subPropertyOf,6))
*/
```

https://github.com/p2m2/tp-big-data-scala-spark-sansa/blob/main/examples/ChebiSansaTripleOpsStatsQualityAssessment.scala

Sansa IO – RDF Quality Assessment

Data quality metrics related to contextual dimensions (type S refers to a subjective metric, O to an objective one).

Dimension	Metric	Description	Type
Completeness	schema completeness	no. of classes and properties represented / total no. of classes and properties [4,15,50]	O
	property completeness	no. of values represented for a specific property / total no. of values for a specific property [4,15]	O
	population completeness	no. of real-world objects are represented / total no. of real-world objects [4,15,26,50]	O
	interlinking completeness	no. of instances in the dataset that are interlinked / total no. of instances in a dataset [22]	O
Amount-of-data	appropriate volume of data for a particular task	ratio of no. of semantically valid association rules to the no. of non-trivial rules[20] [10]	O
	appropriate amount of data	use of the apriori algorithm to detect poor predicates based on the occurrence dependencies among predicates [10]	O
	amount of triples	no. of triples present in a dataset [14]	O
	coverage	scope (no. of entities) and level of detail (no. of properties) [14]	O
Relevancy	usage of meta-information attributes	counting the occurrence of relevant terms within these attributes or using vector space model and assigning higher weight to terms that appear within the meta-information attributes [4]	S
	retrieval of relevant resources	sorting documents according to their relevancy for a given query [4]	S

http://sansa-stack.net/quality-assessment-of-rdf-datasets-at-scale/

Quality Assessment for Linked Open Data: A Survey

A Scalable Framework for Quality Assessment of RDF Datasets

. Gezim Sejdiu ; Anisa Rula ; Jens Lehmann ; and Hajira Jabeen. In *Proceedings of 18th International Semantic Web Conference*, 2019.

Sansa IO – RDF Quality Assessment

```
import net.sansa_stack.rdf.spark.io.RDFReader
import org.apache.jena.riot.Lang
import org.apache.jena.graph.Triple
import org.apache.spark.rdd.RDD
import net.sansa_stack.rdf.spark.qualityassessment._
import scala.util.Try

val path : String = "/rdf/metabohub/metabolights/20211210/MetaboLights_Studies.ttl"
val triples = spark.rdf(Lang.TURTLE)(path)

val q = QualityAssessmentOperations(triples)
val coverage_detail = Try(q.assessCoverageDetail()).getOrElse(" -- ")
```

need to compile sansa with a specific configuration :
sansa-rdf-spark/src/main/resources/metrics.conf

```
# This configuration file contains the settings for the assessment.
rdf.qualityassessment.dataset.prefixes=["https://www.ebi.ac.uk/metabolights/"]

#Class of subjects for which property value is checked
rdf.qualityassessment.dataset.subject="http://purl.org/isaterms/study"

#Property to be checked.
rdf.qualityassessment.dataset.property="http://www.w3.org/2004/02/skos/core#related"

rdf.qualityassessment.dataset.lowerBound=0
rdf.qualityassessment.dataset.upperBound=1

rdf.qualityassessment.dataset.shortUri.threshold = 55
```

https://github.com/p2m2/tp-big-data-scala-spark-sansa/blob/main/examples/QualityAssessmentMetabolights.scala
https://p2m2.github.io/tp-big-data-scala-spark-sansa/qualityassessment.md

Using parquet to store/restore triples

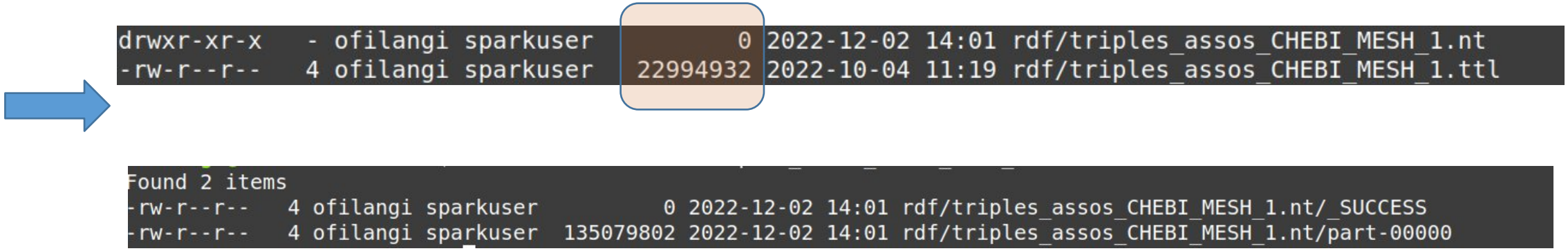
```
import org.apache.jena.graph.Triple
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{Encoder, Encoders}
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.riot.Lang

val chebi : String = « /rdf/ebi/chebi/13-Jun-2022/chebi.owl »
{
   import net.sansa_stack.rdf.spark.model.TripleOperations
   val triplesDs = spark.rdf(Lang.RDFXML)(chebi).toDS()
   // working on triplesDs
   // ...
}.write.save(« /rdf/datalake/chebi.sansa.parquet »)

val triplesDs = spark.read.parquet(« /rdf/datalake/chebi.sansa.parquet »)

//triplesDs.printSchema
//root
// |— value : binary (nullable = true)

implicit val triplesEncoder : Encoder[Triple] = Encoders.kryo(classOf[Triple])
val dataset = triplesDs.as[Triple]
```

Sansa I/O – Save in RDF format : N-Triples

```
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.riot.Lang
```

N-Triples / N-Quads sont des formats sérialisables (1 lignes -> 1 triplet/1quad)

```
triplesRDD.saveAsNTriplesFile(« rdf/some.nt »,mode=SaveMode.Overwrite)
```



Querying on data

Sansa Query - RDD

```
import net.sansa_stack.query.spark.api.domain.ResultSetSpark
import net.sansa_stack.query.spark.sparqlify._
import net.sansa_stack.query.spark.ontop._
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.graph.Triple
import org.apache.jena.query.ResultSet
import org.apache.jena.rdf.model.Model
import org.apache.jena.riot.Lang
import org.apache.jena.sparql.core.Var
import org.apache.jena.sparql.engine.binding.Binding
import org.apache.spark.rdd.RDD

val query = "SELECT ?compound WHERE {
?compound <http://www.w3.org/2004/02/skos/core#related>
<http://id.nlm.nih.gov/mesh/D058573> .
 }"
```

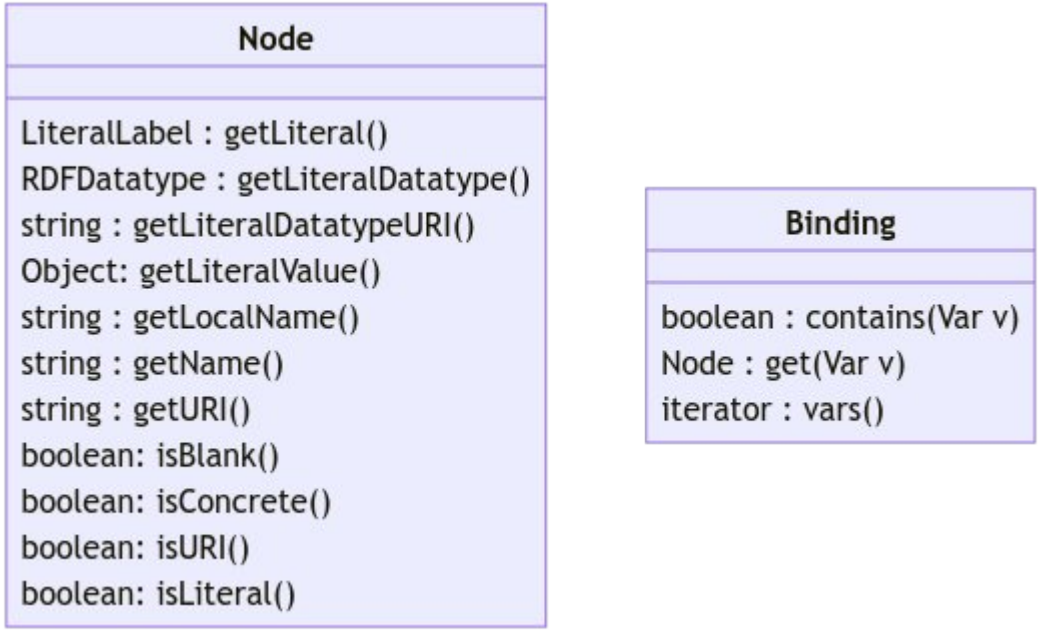
```
val path="./rdf/triples_assos_CHEBI_MESH_1.nt"
val triples = spark.rdf(Lang.NTRIPLES)(path)

 // Ontop
val queryEngineFactory = new QueryEngineFactoryOntop(spark)
// Sparqlify
// val queryEngineFactory = new QueryEngineFactorySparqlify(spark)

val qef1 = queryEngineFactory.create(triples)
val qe = qef1.createQueryExecution(query)
val result: ResultSetSpark = qe.execSelectSpark()

val resultBindings: RDD[Binding] = result.getBindings
```

2 sparql engines available :

- Ontop
- Sparqlify

The Jena « Binding »

```
RDFNode x  = binding.get("varName") ;       // Get a result variable by name.
```



Extracting results from SPARQL Query (1)

```
val query = "SELECT ?compound WHERE {
  ?compound <http://www.w3.org/2004/02/skos/core#related> <http://id.nlm.nih.gov/mesh/D058573> .
}"
```

Transformation using **createDataset : RDD[Binding] -> Dataset[String]**

```
val ds = spark.createDataset(resultBindings.map( binding => binding.get("compound").getURI))
```

Extracting results from SPARQL Query (2)

```
val query = "SELECT ?compound WHERE {
  ?compound <http://www.w3.org/2004/02/skos/core#related> <http://id.nlm.nih.gov/mesh/D058573> .
}"
```

Transformation using **createDataset : RDD[Binding] -> Dataset[Binding]**

```
import org.apache.spark.sql.{Encoder, Encoders}

implicit val compoundEncoder: Encoder[Binding] = Encoders.kryo(classOf[Binding])
val ds = spark.createDataset(resultBindings)
```

Extracting results from SPARQL Query (3)

```
val query = "SELECT ?compound WHERE {
  ?compound <http://www.w3.org/2004/02/skos/core#related> <http://id.nlm.nih.gov/mesh/D058573> .
}"
```

Transformation using **createDataset : RDD[Binding] -> Dataset[<UserClass>]**

```
import org.apache.spark.sql.{Encoder, Encoders}

case class Compound(uri : String)

implicit val compoundEncoder: Encoder[Compound] = Encoders.kryo(classOf[Compound])
val ds = spark.createDataset(
            resultBindings.map( binding => Compound(uri=binding.get("compound").getURI))
           )
```

Sansa Query - Dataset / DataFrame

```
import net.sansa_stack.rdf.spark.io.RDFReader
import net.sansa_stack.rdf.spark.model.TripleOperations // to
reach `toDS()` functionnality
import org.apache.jena.riot.Lang
import org.apache.jena.graph.Triple
import org.apache.spark.sql.Dataset
import net.sansa_stack.ml.spark.featureExtraction.SparqlFrame
import net.sansa_stack.query.spark.SPARQLEngine
import org.apache.spark.sql.DataFrame

val query = """SELECT ?compound WHERE {
?compound <http://www.w3.org/2004/02/skos/core#related>
<http://id.nlm.nih.gov/mesh/D058573> .
 }"""
```

```
val path="./rdf/triples_assos_CHEBI_MESH_1.nt"
val triplesDataset = spark.rdf(Lang.NTRIPLES)(path).toDS()

val sparqlFrame = new SparqlFrame()
      .setSparqlQuery(query)
      .setQueryExcecutionEngine(SPARQLEngine.Sparqlify)
                                    // SPARQLEngine.Ontop is not supported !

val resultsDF : DataFrame = sparqlFrame.transform(triplesDataset)


resultsDF.printSchema
/*
root
 |-- compound: string (nullable = false)
*/
resultsDF.select("compound").show(2,false)
/*
+----------------------------------------+
|compound                                |
+----------------------------------------+
|http://purl.obolibrary.org/obo/CHEBI_17375 |
|http://purl.obolibrary.org/obo/CHEBI_174690|
+----------------------------------------+
*/
```

Sansa Inference

Supported Reasoning Profiles

- RDFS
- OWL Horst

Inference：ForwardRuleReasonerRDFS

https://www.w3.org/TR/rdf11-mt/#patterns-of-rdfs-entailment-informative

RDFS entailment patterns.

	If S contains:	then S RDFS entails recognizing D:
rdfs1	any IRI aaa in D	aaa `rdf:type rdfs:Datatype .`
rdfs2	aaa `rdfs:domain` xxx . yyy aaa zzz .	yyy `rdf:type` xxx .
rdfs3	aaa `rdfs:range` xxx . yyy aaa zzz .	zzz `rdf:type` xxx .
rdfs4a	xxx aaa yyy .	xxx `rdf:type rdfs:Resource .`
rdfs4b	xxx aaa yyy.	yyy `rdf:type rdfs:Resource .`
rdfs5	xxx `rdfs:subPropertyOf` yyy . yyy `rdfs:subPropertyOf` zzz .	xxx `rdfs:subPropertyOf` zzz .
rdfs6	xxx `rdf:type rdf:Property .`	xxx `rdfs:subPropertyOf` xxx .
rdfs7	aaa `rdfs:subPropertyOf` bbb . xxx aaa yyy .	xxx bbb yyy .
rdfs8	xxx `rdf:type rdfs:Class .`	xxx `rdfs:subClassOf rdfs:Resource .`
rdfs9	xxx `rdfs:subClassOf` yyy . zzz `rdf:type` xxx .	zzz `rdf:type` yyy .
rdfs10	xxx `rdf:type rdfs:Class .`	xxx `rdfs:subClassOf` xxx .
rdfs11	xxx `rdfs:subClassOf` yyy . yyy `rdfs:subClassOf` zzz .	xxx `rdfs:subClassOf` zzz .
rdfs12	xxx `rdf:type rdfs:ContainerMembershipProperty .`	xxx `rdfs:subPropertyOf rdfs:member .`
rdfs13	xxx `rdf:type rdfs:Datatype .`	xxx `rdfs:subClassOf rdfs:Literal .`

Inference : ForwardRuleReasonerRDFS

```
val prefixDeclStr =
   """@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
     |@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
     |@prefix owl: <http://www.w3.org/2002/07/owl#> .
     |@prefix ab: <http://atelierbigdata-inrae-sete/2023/01#> .
     |
     |ab:Femme rdfs:subClassOf ab:Humain .
     |ab:Anne rdf:type ab:Femme .
     |ab:Anne ab:married ab:Shakespeare .
     |ab:married rdf:type owl:SymmetricProperty .
  """.stripMargin
//....
val triples: RDD[Triple] = sc.parallelize(toTriples(turtle))
val parallelism : Int = 1
```

```
val reasoner=new ForwardRuleReasonerRDFS(spark.sparkContext,parallelism)
val inferredTriples = reasoner.apply(triples)

triples.count()
inferredTriples.count()
triples.collect().map(println)
inferredTriples.collect().map(println)
```

```
scala> triples.collect().map(println)
http://atelierbigdata-inrae-sete/2023/01#Femme @rdfs:subClassOf http://atelierbigdata-inrae-sete/2023/01#Humain
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type http://atelierbigdata-inrae-sete/2023/01#Femme
http://atelierbigdata-inrae-sete/2023/01#Anne @http://atelierbigdata-inrae-sete/2023/01#married http://atelierbigdata-inrae-sete/2023/01#Shakespeare
http://atelierbigdata-inrae-sete/2023/01#married @rdf:type owl:SymmetricProperty
res2: Array[Unit] = Array((), (), (), ())

scala> inferredTriples.collect().map(println)
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type http://atelierbigdata-inrae-sete/2023/01#Femme
http://atelierbigdata-inrae-sete/2023/01#Femme @rdfs:subClassOf http://atelierbigdata-inrae-sete/2023/01#Humain
owl:SymmetricProperty @rdf:type rdfs:Resource
http://atelierbigdata-inrae-sete/2023/01#married @rdf:type rdfs:Resource
http://atelierbigdata-inrae-sete/2023/01#Anne @http://atelierbigdata-inrae-sete/2023/01#married http://atelierbigdata-inrae-sete/2023/01#Shakespeare
http://atelierbigdata-inrae-sete/2023/01#Shakespeare @rdf:type rdfs:Resource
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type rdfs:Resource
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type http://atelierbigdata-inrae-sete/2023/01#Humain
http://atelierbigdata-inrae-sete/2023/01#Femme @rdf:type rdfs:Resource
http://atelierbigdata-inrae-sete/2023/01#married @rdf:type owl:SymmetricProperty
http://atelierbigdata-inrae-sete/2023/01#Humain @rdf:type rdfs:Resource
res3: Array[Unit] = Array((), (), (), (), (), (), (), (), (), ())
```

https://github.com/p2m2/tp-big-data-scala-spark-sansa/blob/main/examples/RDFSReasonerShakespeare.scala

Inference : ForwardRuleReasonerOWLHorst

OWL Horst is limited to RDFS (subClassOf, subSpropertyOf, domain and range) plus what was popular in the past as OWL Lite : sameAs, equivalentClass, equivalentProperty, SymmetricProperty, TransitiveProperty, inverseOf, FunctionalProperty, InverseFunctionalProperty. There is also partial support for : intersectionOf, someValuesFrom, hasValue, allValuesFrom.

https://stackoverflow.com/questions/63163024/horst-pd-compared-to-owl2-rl

Inference : ForwardRuleReasonerOWLHorst

```
val prefixDeclStr =
   """@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
     |@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
     |@prefix owl: <http://www.w3.org/2002/07/owl#> .
     |@prefix ab: <http://atelierbigdata-inrae-sete/2023/01#> .
     |
     |ab:Femme rdfs:subClassOf ab:Humain .
     |ab:Anne rdf:type ab:Femme .
     |ab:Anne ab:married ab:Shakespeare .
     |ab:married rdf:type owl:SymmetricProperty .
  """.stripMargin
//....
val triples: RDD[Triple] = sc.parallelize(toTriples(turtle))
val parallelism : Int = 1
```

```
val reasoner=new ForwardRuleReasonerOWLHorst(spark.sparkContext,parallelism)
val inferredTriples = reasoner.apply(triples)

triples.count()
inferredTriples.count()
triples.collect().map(println)
inferredTriples.collect().map(println)
```

```
scala> triples.collect().map(println)
http://atelierbigdata-inrae-sete/2023/01#Femme @rdfs:subClassOf http://atelierbigdata-inrae-sete/2023/01#Humain
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type http://atelierbigdata-inrae-sete/2023/01#Femme
http://atelierbigdata-inrae-sete/2023/01#Anne @http://atelierbigdata-inrae-sete/2023/01#married http://atelierbigdata-inrae-sete/2023/01#Shakespeare
http://atelierbigdata-inrae-sete/2023/01#married @rdf:type owl:SymmetricProperty
res2: Array[Unit] = Array((), (), (), ())

scala> inferredTriples.collect().map(println)
http://atelierbigdata-inrae-sete/2023/01#Shakespeare @http://atelierbigdata-inrae-sete/2023/01#married http://atelierbigdata-inrae-sete/2023/01#Anne
http://atelierbigdata-inrae-sete/2023/01#Femme @rdfs:subClassOf http://atelierbigdata-inrae-sete/2023/01#Humain
http://atelierbigdata-inrae-sete/2023/01#Anne @http://atelierbigdata-inrae-sete/2023/01#married http://atelierbigdata-inrae-sete/2023/01#Shakespeare
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type http://atelierbigdata-inrae-sete/2023/01#Femme
http://atelierbigdata-inrae-sete/2023/01#Anne @rdf:type http://atelierbigdata-inrae-sete/2023/01#Humain
http://atelierbigdata-inrae-sete/2023/01#married @rdf:type owl:SymmetricProperty
http://atelierbigdata-inrae-sete/2023/01#Femme @rdfs:subClassOf http://atelierbigdata-inrae-sete/2023/01#Humain
res3: Array[Unit] = Array((), (), (), (), (), (), ())
```

https://github.com/p2m2/tp-big-data-scala-spark-sansa/blob/main/examples/OwlReasonerShakespeare.scala

Runtime environment / packaging

Runtime environment / packaging

- **spark-shell**
  - *Interactive mode to evaluate, test or to apply corrective actions on production*

- **SBT/ Application Scala**
  - *for execution in batch mode that can be integrated into a workflow*

Sansa

- Sansa is a library (JAR) which have to be in the classpath of the jvm

```
>wget https://github.com/SANSA-Stack/SANSA-Stack/releases/download/v0.8.5_ExPAD/sansa-ml-spark_2.12-0.8.0-RC3-SNAPSHOT-jar-with-dependencies.jar
>git clone https://github.com/SANSA-Stack/SANSA-Stack.git
>cd SANSA-Stack
>sh ./dev/make_spark_dist.sh
```

INSTALLATION : **Compilation JAR** -> https://p2m2.github.io/tp-big-data-scala-spark-sansa/prerequisites.html

Java 12

Spark-shell : Environnement d'exécution

```
spark-shell \
 --name TP \
 --master yarn \
 --conf "spark.yarn.appMasterEnv.JAVA_HOME=/usr/lib/jvm/jdk-12.0.2+10/" \
 --conf "spark.executorEnv.JAVA_HOME=/usr/lib/jvm/jdk-12.0.2+10/" \
 --conf "spark.serializer=org.apache.spark.serializer.KryoSerializer"  \
 --conf "spark.sql.crossJoin.enabled=true"   \
 --conf "spark.kryo.registrator=net.sansa_stack.rdf.spark.io.JenaKryoRegistrator"  \
 --conf "spark.kryoserializer.buffer.max=2000" \
 --conf spark.sql.shuffle.partitions="300" \
 --executor-memory 4G \
 --num-executors 4  \
 --jars /usr/share/java/sansa-stack-spark_2.12-0.8.0-RC3-SNAPSHOT-jar-with-dependencies.jar
```

java 12

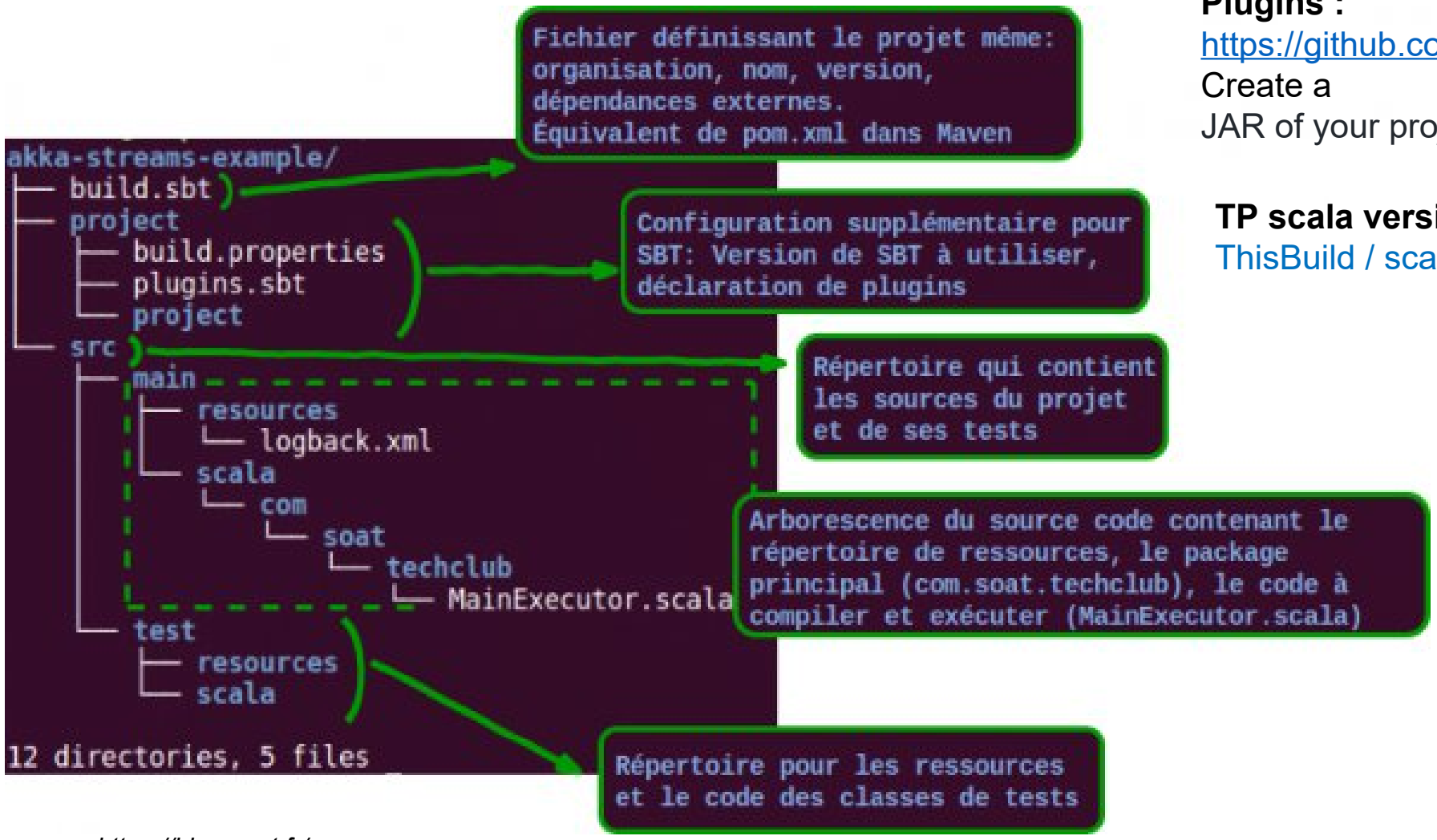
Stratégie de sérialisation des classes que Jena Implémente

JAR SANSA

SPARQL Engine use

option spark-shell
—conf « spark.kryo.registrator=net.sansa_stack.rdf.spark.io.JenaKryoRegistrator
,net.sansa_stack.query.spark.ontop.OntopKryoRegistrator,
net.sansa_stack.query.spark.sparqlify.KryoRegistratorSparqlify »

SBT / Application



source : https://blog.soat.fr/

Plugins :
https://github.com/sbt/sbt-assembly
Create a
JAR of your project with all of its dependencies

TP scala version :
ThisBuild / scalaVersion := "2.12.16"

Main : Environnement d'exécution

build.sbt 1

```
lazy val root = (project in file("."))
  .settings(
    name := "tp-big-data-scala-spark-sansa",

                libraryDependencies ++= Seq(
      ("org.apache.spark" %% "spark-sql"  % sparkVersion % "provided,test")
        .exclude("com.fasterxml.jackson","databind"),
      ("net.sansa-stack" %% "sansa-rdf-spark" % "0.8.0-RC3")
        .exclude("org.apache.avro","avro-mapred")
        .exclude("org.apache.hadoop","hadoop-common") % "test,provided",
      ("net.sansa-stack" %% "sansa-ml-spark" % "0.8.0-RC3")
        .exclude("org.apache.avro","avro-mapred")
        .exclude("org.apache.hadoop","hadoop-common") % "test,provided",
      ("net.sansa-stack" %% "sansa-inference-spark" % "0.8.0-RC3")
        .exclude("org.apache.avro","avro-mapred")
        .exclude("org.apache.hadoop","hadoop-common") % "test,provided",
      "com.github.scopt" %% "scopt" % "4.1.0"
    ),
    Compile / mainClass := Some("fr.inrae.bigdata.tp.Main"),
```

Main.scala 2

```
val spark = SparkSession
  .builder()
  .appName("tp-bigdata-rdf-sansa")
  .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
  .config("spark.sql.crossJoin.enabled", "true")
  .config("spark.kryo.registrator","net.sansa_stack.rdf.spark.io.JenaKryoRegistrator")
  .config("spark.kryoserializer.buffer.max.mb","1800")
  .getOrCreate()
```

sbt asembly 3

4

```
spark-submit \
 --name TP \
 --master yarn \
 --conf "spark.yarn.appMasterEnv.JAVA_HOME=/usr/lib/jvm/jdk-12.0.2+10/" \
 --conf "spark.executorEnv.JAVA_HOME=/usr/lib/jvm/jdk-12.0.2+10/" \
 --executor-memory 4G \
 --num-executors 4  \
 --jars /usr/share/java/sansa-stack-spark_2.12-0.8.0-RC3-SNAPSHOT-jar-with-dependencies.jar <path/tp.jar>
```

https://github.com/p2m2/tp-big-data-scala-spark-sansa/blob/main/build.sbt

- Documentation
  - http://sansa-stack.github.io/SANSA-Stack/
  - https://jena.apache.org/documentation/javadoc.html

- Cours
  - https://github.com/p2m2/tp-big-data-scala-spark-sansa/
  - Programmation fonctionnelle (Scala) sur Spark (6 x 1/2 journées)
    - https://nextcloud.inrae.fr/s/4RDPFrJYP2RfKdg
  - Administration cluster Hadoop/Spark (4 x 1/2 journées)
    - https://nextcloud.inrae.fr/s/YM845N383cQWCEX