Apache Jena Framework

Philippe Genoud &
Jérôme David

Introduction

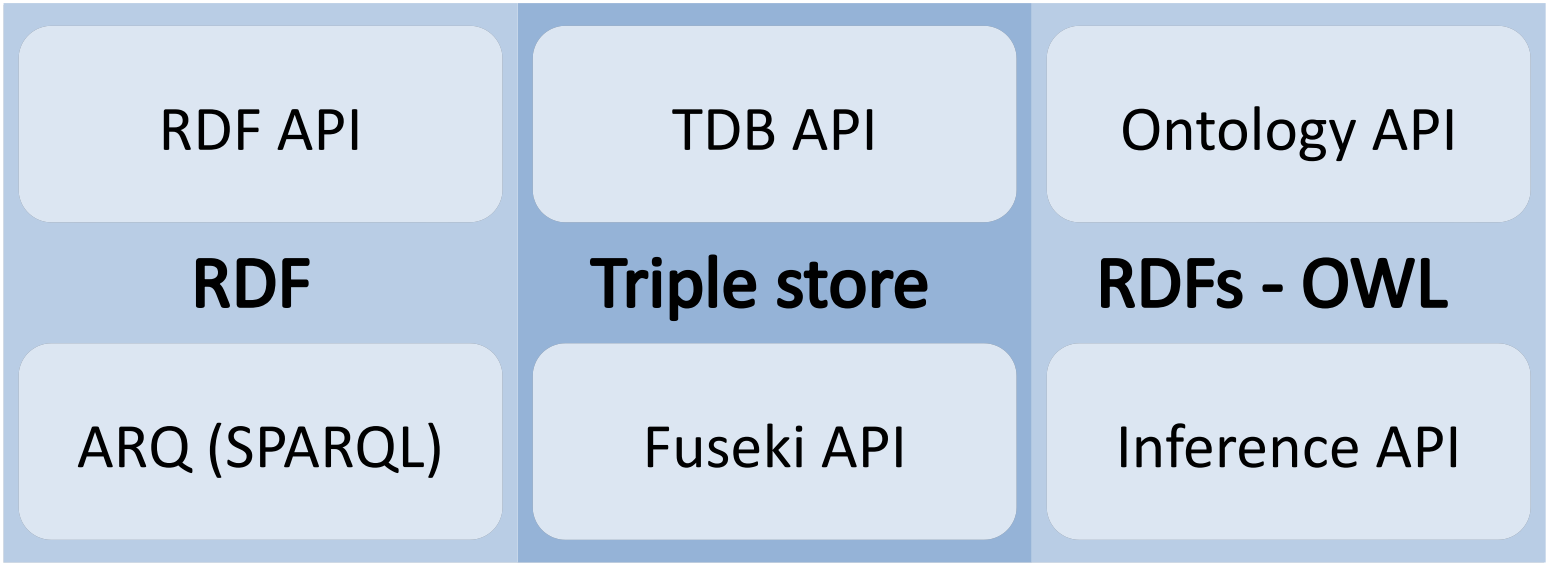
- What is Jena?
  - A free and open source Java framework for building Semantic Web and Linked Data applications.
  - developed at HP-Labs (Bristol-UK)
  - now an apache project : http://jena.apache.org/
    - November 2010: adopted by the Apache Software Foundation (incubation)
    - April 2012: graduated as a top-level project
    - Current version (on December 12, 2019) : 3.13

Jena APIs

Core API to create and read RDF graphs and serialize them in standard formats (RDF/XML, Turtle…)

A native high performance triple store to persist RDF data

API for handling OWL and RDFS ontologies to add extra semantics to RDF data

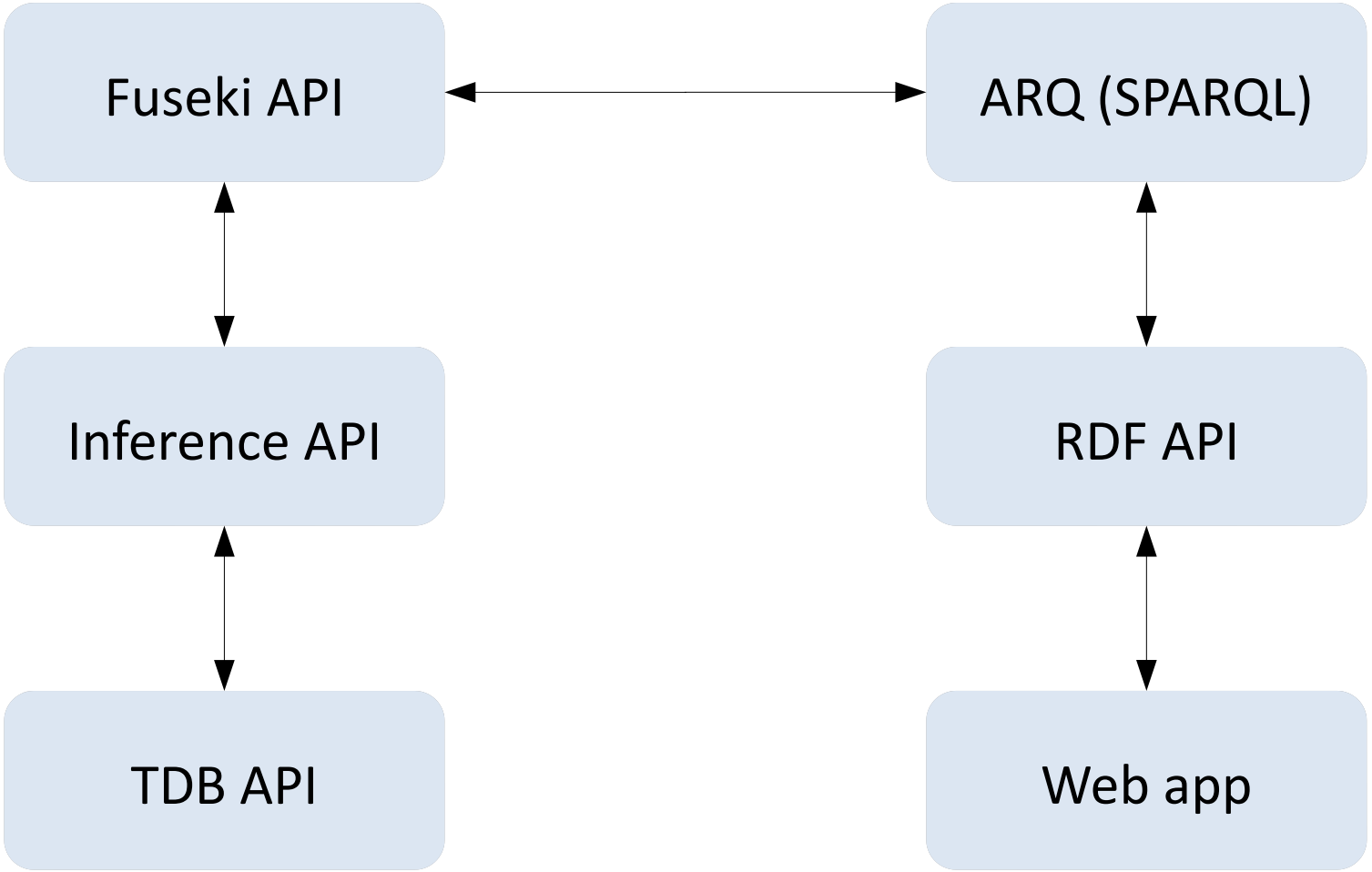


A query engine compliant with the latest SPARQL specification (1.1)

To expose RDF triples as a SPARQL end-point accessible over HTTP. Provides REST-style interaction with RDF data.

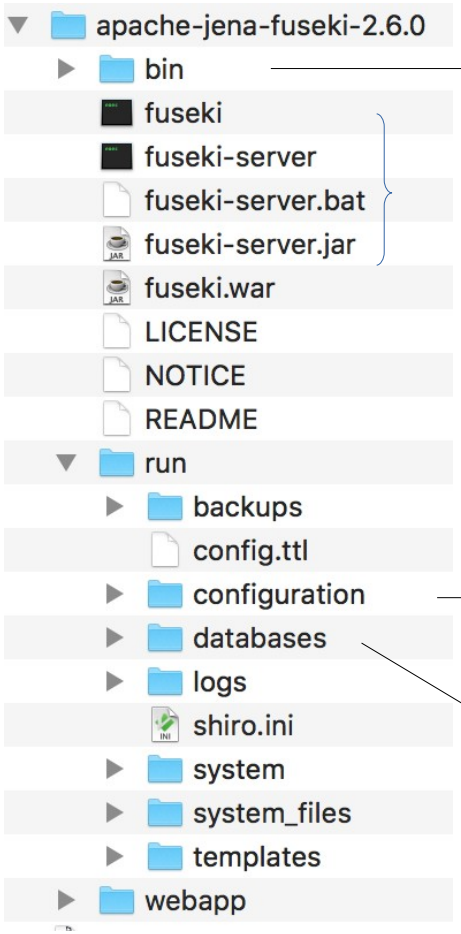
To reason over RDF data to expand and check it. Configure your own inference rules or use the built-in OWL and RDFS reasoners.

What we will try to do ?



Let's try Fuseki

- What the package contains ?



Commands (shell) to access sparql endpoint via HTTP (SOH)

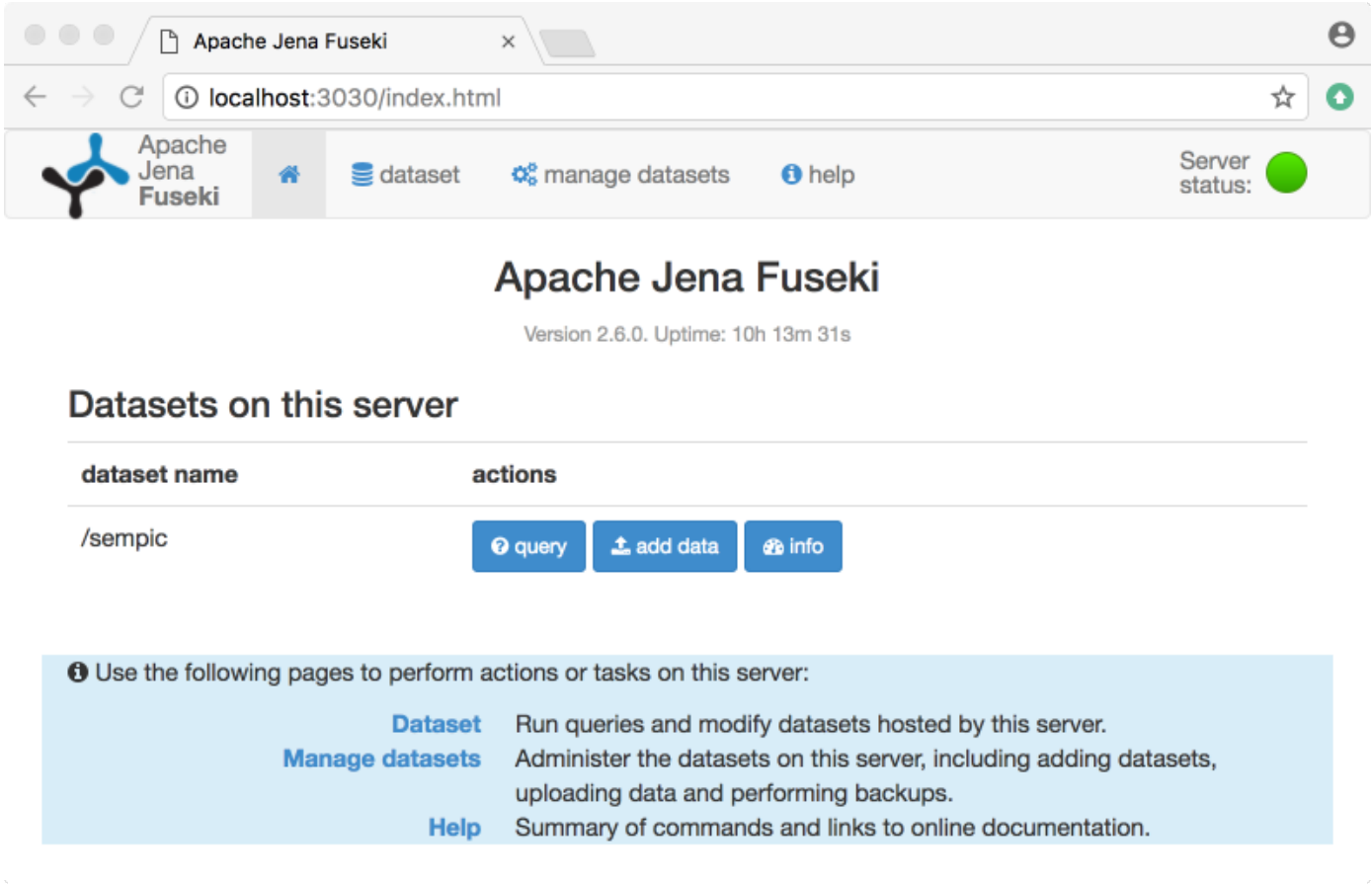
Commands to launch the server

Configuration files for each dataset. Config file is written in RDF :

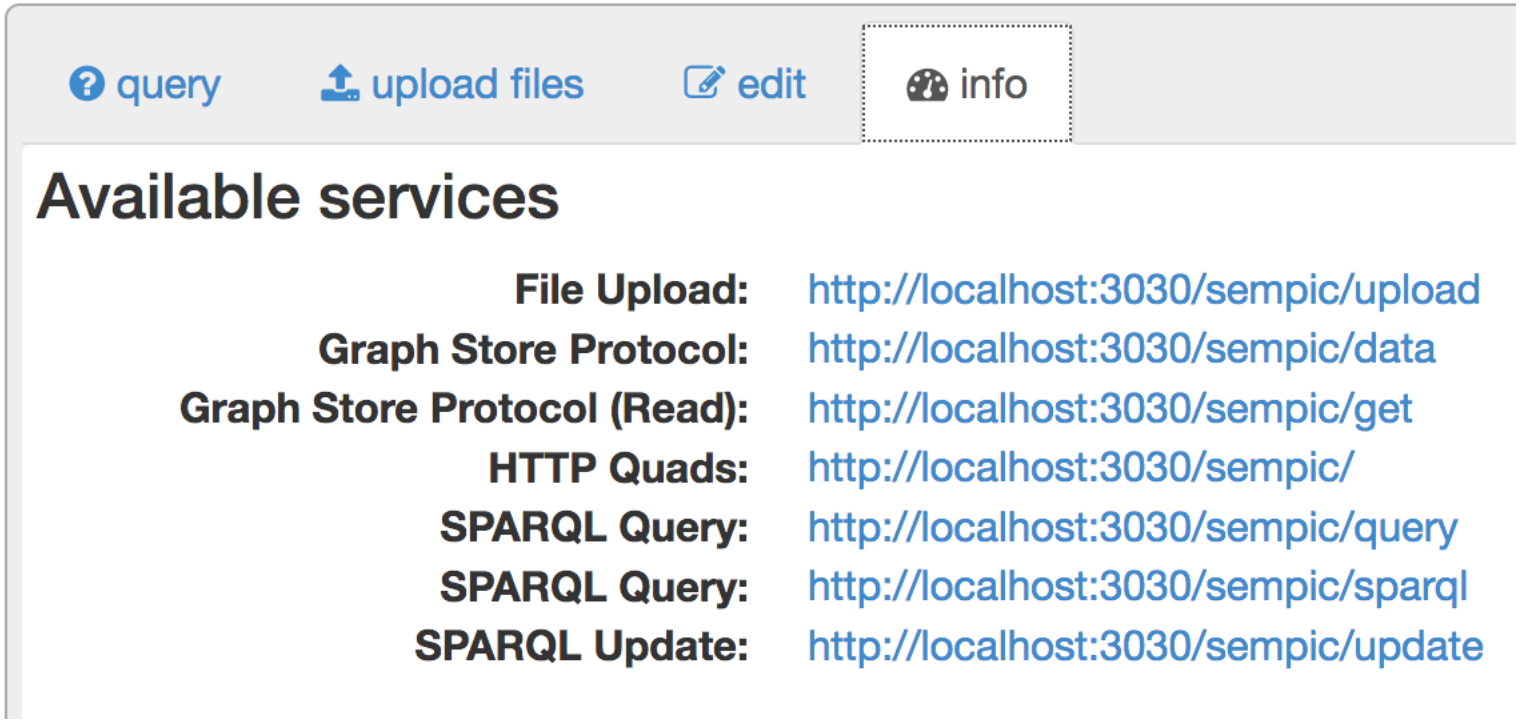
TDB files for each persistent dataset (could be store elsewhere, if needed)

Let's try Fuseki

- Launch the server
  - ./fuseki start or ./fuseki-server start
- And go to http://localhost:3030



Fuseki: services and supported protocols



Querying Fuseki from Jena

- ARQ allows to query RDF from
  - Jena models (i.e. RDF Graph) loaded into memory or via TDB
  - **Remote HTTP endpoint (for instance Fuseki)**
- Jena provides `RDFConnection` API
  - Package `org.apache.jena.rdfconnection`
  - It allows to use : SPARQL Query, SPARQL Update and Graph Store Protocol services
  - It supports transactions
  - Documentation : https://jena.apache.org/documentation/rdfconnection/

RDFConnection - SPARQL SELECT

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        QueryExecution qe = cnx.query("SELECT DISTINCT ?s WHERE {?s ?p ?o}");
        ResultSet rs = qe.execSelect();
        while (rs.hasNext()) {
            QuerySolution qs = rs.next();
            System.out.println(qs.getResource("s"));
        }

        cnx.close();
    }
}
```

Shorter syntax with Java 8 (lambda expressions)

```
cnx.querySelect("SELECT DISTINCT ?s WHERE {?s ?p ?o}", qs -> {
            System.out.println(qs.getResource("s"));
        });
```

RDFConnection - SPARQL CONSTRUCT

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        Model res = cnx.queryConstruct("CONSTRUCT {?s a ?t} WHERE {?s a ?t}");
        res.write(System.out);

        cnx.close();
    }
}
```

The type Model is the Jena type for representing RDF graphs

RDFConnection - SPARQL UPDATE

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        cnx.begin(ReadWrite.WRITE);
        cnx.update("UPDATE INSERT DATA {<http://test.org/dudule> a <ttp://test.org/bidule>");
        cnx.update("UPDATE DELETE WHERE {<http://test.org/dudue> ?p ?o");
         cnx.commit();

        cnx.close();
    }
}
```

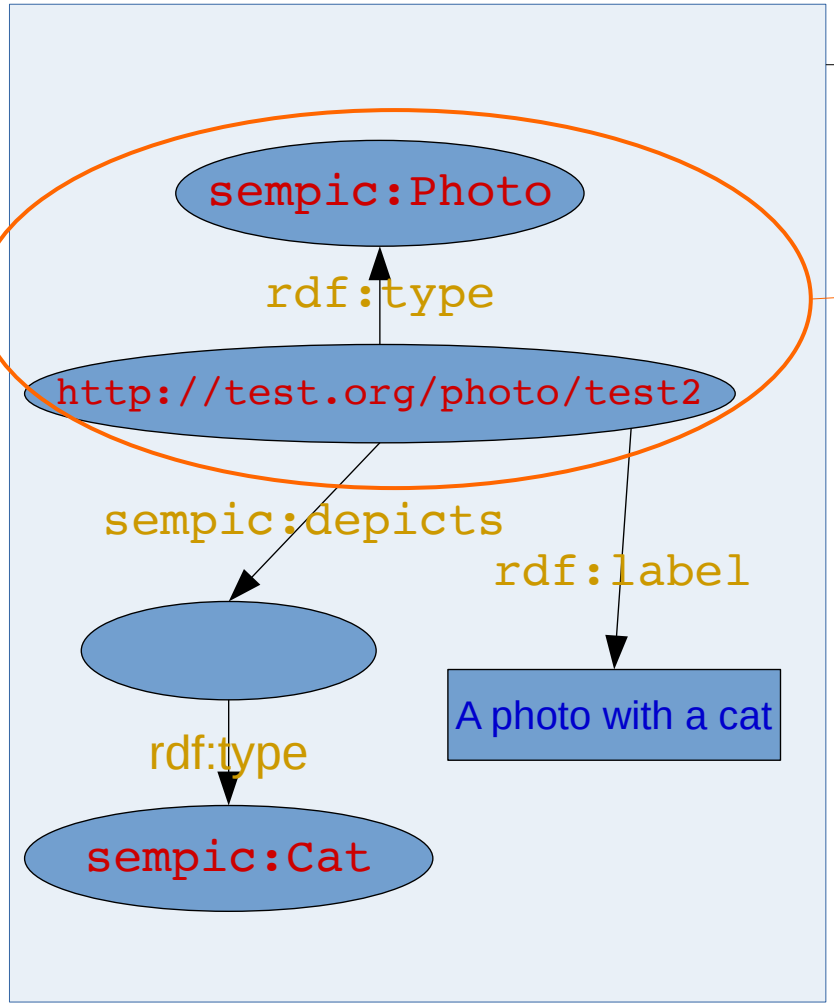
Jena provides transactions.
Use cnx.abort() if you want to abort(rollback) the transaction

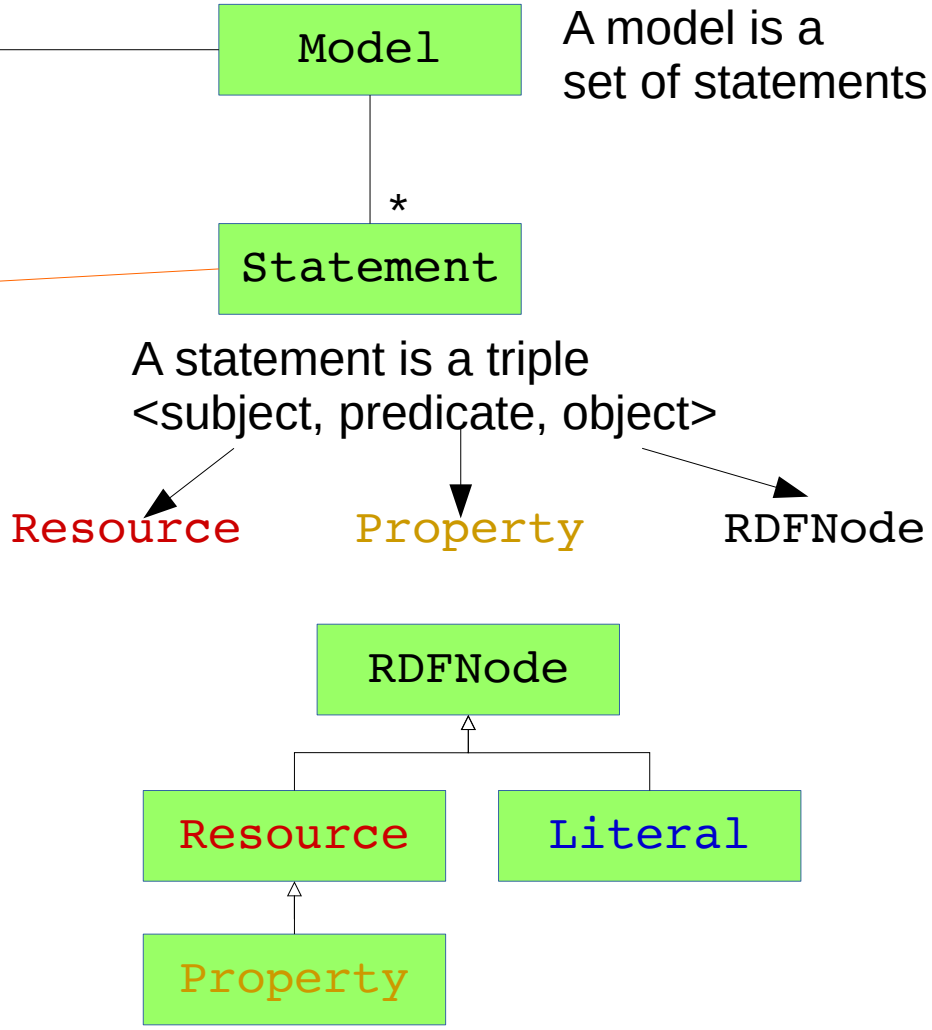
RDFConnection - Graph Store Protocol

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        cnx.begin(ReadWrite.WRITE);
        cnx.update("UPDATE INSERT DATA {<http://test.org/dudule> a <<ttp://test.org/bidule>");
        cnx.update("UPDATE DELETE WHERE {<http://test.org/dudue> ?p ?o");
         cnx.commit();

        cnx.close();
    }
}
```

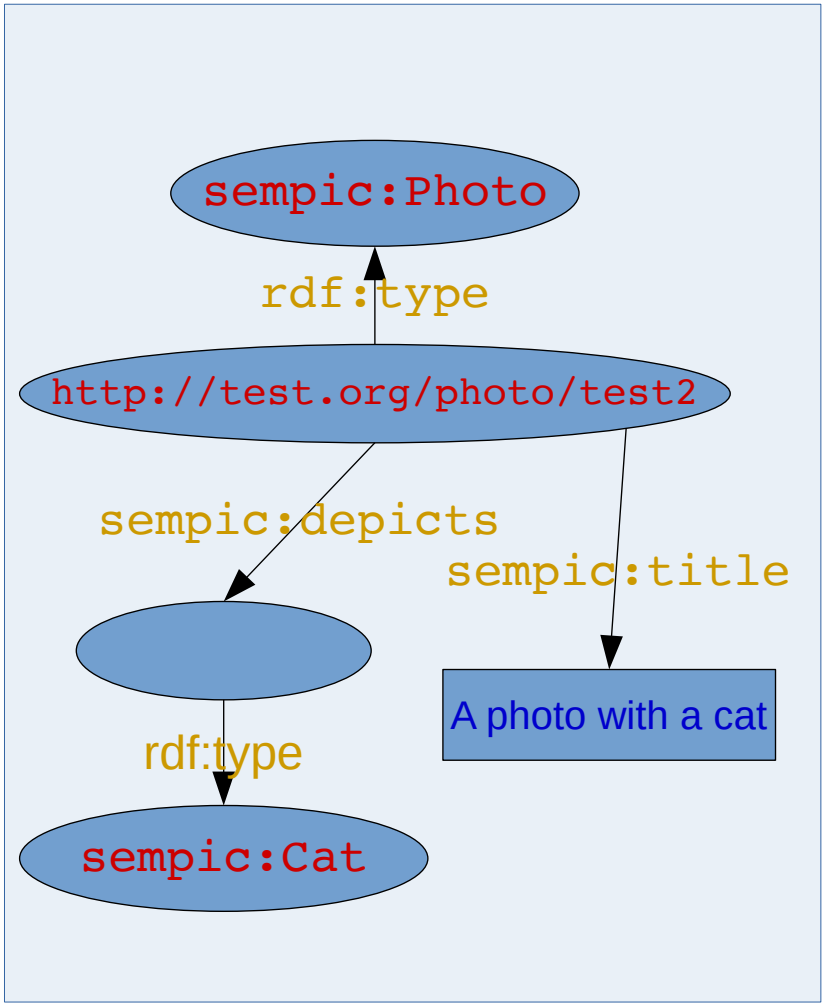
Core RDF API

package org.apache.jena.rdf.model





Core RDF API: create a model




```
// Create an empty RDF model
Model m = ModelFactory.createDefaultModel();
// create and add a named (i.e. with an URI)
// resource of type photo
// this adds statement:
// [<.../test2> rdf:type sempic:photo]
Resource photo = m.createResource(
                 "http://test.org/photo/test2",
                 Photo.type);
// this adds statement:
// [<.../test2> sempic:title "A photo with a cat"]
photo.addLiteral(Photo.title, "A photo with a cat");
// create and add an anonymous resource
// of type cat
// this adds statement:
// [_:generatedID rdf:type sempic:cat]
Resource cat = m.createResource(Types.cat);
// this adds statement:
// [<.../test2> sempic:depicts _:generatedID]
photo.addProperty(Photo.depicts, cat);
```

Core RDF API
package `org.apache.jena.rdf.model`

- interface **`Model`**
  - represents a RDF Graph
  - a model is a set of **`Statements`** (triples).
- methods are provided for
  - creating resources, properties and literals
  - creating the **`Statements`** (triples) which link them,
  - for adding **`Statements`** to and removing them from a model,
  - for querying a model
  - set operations for combining models.

Core RDF API
package `org.apache.jena.rdf.model`

- Factory design pattern to create models
  - class **`ModelFactory`**
    - `static Model createDefaultModel()`
      - answer a new memory based Model (RDF Graph)
    - `static ModelMaker createFileModelMaker(String root)`
      - answer a `ModelMaker` that constructs memory-based Models that are backed by files
    - `static OntModel createOntologyModel()`
      - answer a new ontology model which will process in-memory models of ontologies expressed the default ontology language (OWL)
    - …

Core RDF API
package org.apache.jena.rdf.model

- `Model` method to create a `Statement`
  - `Statement createStatement(Resource s, Property p, RDFNode o)`

Creating a new Statement does not add it to the set of statements in the model

- `Model` methods to add it newly created `Statements`
  - `Model add(Statement stmt)`
  - `Model add(List<Statement> statements)`
  - `Model add(Statement[] statements)`
  - `Model add(StmtIterator iter)`

- `Statement` accessor methods
  - `Resource getSubject()`
  - `Property getPredicate()`
  - `RDFNode getObject()`

Core RDF API
package `org.apache.jena.rdf.model`

- `Resource` methods to create triples with the resource (this) as subject.
  - `Resource addProperty(…)`
    - `addProperty(Property p, RDFNode o)`
    - `addProperty(Property p, java.lang.String o)`
    - …
  - `Resource addLiteral(…)`
    - `addLiteral(Property p, Literal o)`
    - `addLiteral(Property p, double d)`
    - …

newly created Statement are automatically added to the set of statements in the model the Resource belongs to.

Core RDF API

package `org.apache.jena.rdf.model`

- **`addProperty`** and **`addLiteral`** methods return the **`Resource`** (this)

  → facilitates creation of multiple triples with the same subject

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

- vCard : a standard file format for electronic business cards
  - e.g. vCards are often attached to e-mail messages
  - http://www.w3.org/TR/vcard-rdf/ → RDF vocabulary corresponding to vCard

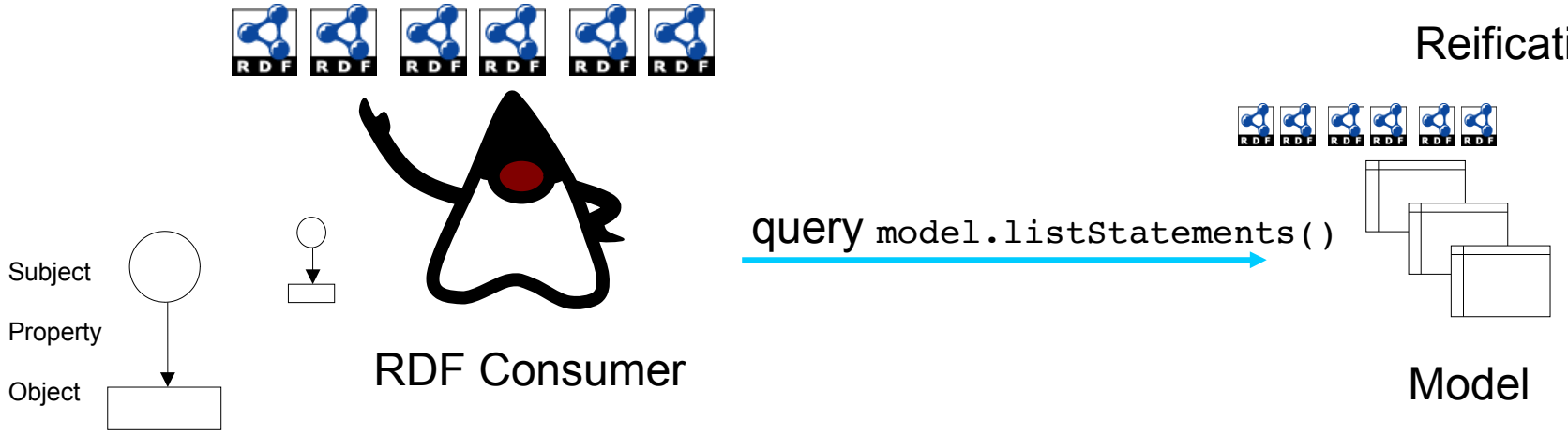
- **`org.apache.jena.vocabulary`** : package containing constant classes with predefined constant objects for classes and properties defined in well known vocabularies.
  - **`org.apache.jena.vocabulary.VCARD`** : vCard
    - static Property NAME
    - .
  - **`org.apache.jena.vocabulary.RDF`** : The standard RDF vocabulary
    - static Property type
    - ...
  - **`org.apache.jena.vocabulary.DC`** : Dublin Core

Core RDF API

package `org.apache.jena.rdf.model`

- **`StatementIterator`** to access statements asserted in a RDF Model
  - **`Model`** methods to get a **`StatementIterator`**
    - **`StmtIterator listStatements()`**: List all statements of the model
    - **`StmtIterator listStatements(Resource s, Property p, RDFNode o)`**:List all the statements matching a pattern

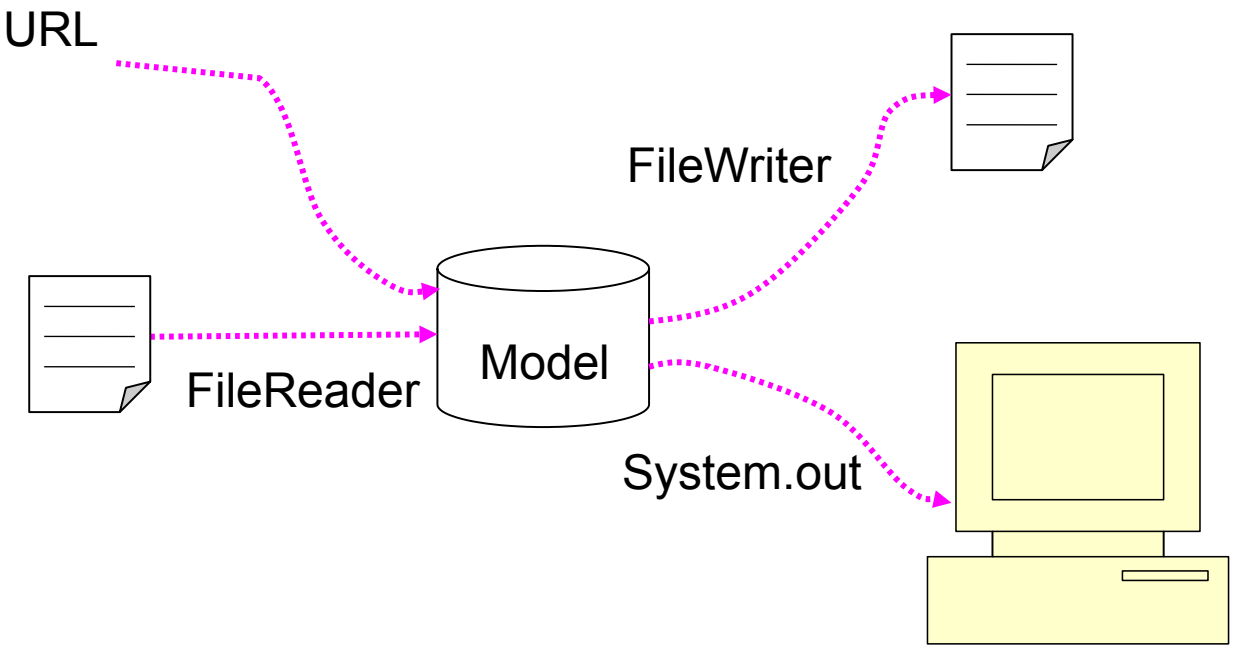
```
for(StatementIterator it = model.listStatements();it.hasNext();) {
    Statement stmt = it.nextStatement();
   System.out.println(stmt.getSubject().getLocalName());
}
```



animation from *http://fr.slideshare.net/yuhana/070517-jena* C.F.Liao (廖峻鋒)

Core RDF API
package `org.apache.jena.rdf.model`

- **`read`** and **`write`** methods of **`Model`** interface
  - e.g. **`write(java.io.OutputStream out, String lang, String base)`**
    - `lang` : output format "RDF/XML" (default), "RDF/XML-ABBREV" , "N3", "TURTLE", "N-TRIPLE"
    - `base` : The base uri for relative URI calculations. null means use only absolute URI's



Jena API for SPARQL

- **ARQ** : query engine for Jena that supports the SPARQL RDF Query language.

- javadoc
  - http://jena.apache.org/documentation/javadoc/arq/index.html

- **org.apache.jena.query** package for ARQ API

ARQ (SPARQL) API
package `org.apache.jena.query`

Key classes

- **`Query`** represents the application query.
  - **`QueryFactory`** : to create `Query` instances
- **`QueryExecution`** - represents one execution of a query.
  - **`QueryExecutionFactory`** - a place to get `QueryExecution` instances
- **`Dataset`**, a collection of named graphs and a default graph (background graph or unnamed graph).
  - **`DatasetFactory`** - a place to make datasets

ARQ (SPARQL) API
package `org.apache.jena.query`

Key classes

- For SELECT queries:
  - **`QuerySolution`** - A single solution to the query
  - **`ResultSet`** - An iterator over all the the `QuerySolutions`.
  - **`ResultSetFormatter`** : turns a `ResultSet` into various forms
    - text,
    - Model (an RDF graph)
    - plain XML
    - …

ARQ (SPARQL) API
package org.apache.jena.query

SELECT queries
① Preparing the request

```
import org.apache.jena.query.* ;

Model model = ... ;

String queryString = " ...SELECT... " ;

Query query = QueryFactory.create(queryString) ;

QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

try

} f

}
```

import for ARQ classes
better to have explicit imports
`import org.apache.jena.query.Query`

construct the SPARQL query String

construct the SPARQL query

produce an instance of `QueryExecution`

can be reduced to one step in some common cases:

```
QueryExecution qexec =
      QueryExecutionFactory.create(queryString,
model);
```

ARQ (SPARQL) API
package org.apache.jena.query

SELECT queries
② Executing the request and exploiting the results

```
import org.apache.jena.query.* ;

Model model = ... ;

String queryString = " ...SELECT... " ;

Query query = QueryFactory.create(queryString) ;

QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

try {
    ResultSet results = qexec.execSelect();

    for ( ; results.hasNext() ; ) {
      QuerySolution soln = results.nextSolution() ;
      RDFNode x = soln.get("varName") ;
      Resource r = soln.getResource("varR") ;
      Literal l = soln.getLiteral("varL") ;
    }
} finally {
   qexec.close() ;
}
```

it's an ARQ Resultset not a java.sql.ResultSet !

execute the query

Iteration loop to traverse **QuerySolutions**

`ResultSet` presents the results of a SELECT query in table-like manner.

Each row (**QuerySolution**) corresponds to a set of bindings which fulfill the conditions of the query.

SELECT queries

exploiting the results

```
for ( ; results.hasNext() ; ) {
   QuerySolution soln = results.nextSolution() ;
   RDFNode n = soln.get("varName") ;
   Resource r = soln.getResource("varR") ;
   Literal l = soln.getLiteral("varL") ;
}
```

Instead of a loop to deal with each row in the result set,
the application can call an operation of the `ResultSetFormatter`.

process results to produce a simple text presentation

```
ResultSetFormatter fmt = new ResultSetFormatter(results, query) ;
fmt.printAll(System.out) ;
```

or simply

```
ResultSetFormatter.out(System.out, results, query) ;
```

ARQ (SPARQL) API
package org.apache.jena.query

CONSTRUCT and DESCRIBE queries

- CONSTRUCT

```
import com.hp.hpl.jena.query.* ;

Model model = ... ;

String queryString = " ...CONSTRUCT... " ;

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
Model resultModel = qexec.execConstruct() ;
qexec.close() ;
```

CONSTRUCT queries return a single RDF graph

- DESCRIBE

```
import com.hp.hpl.jena.query.* ;

Model model = ... ;

String queryString = " ...DESCRIBE... " ;

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
Model resultModel = qexec.execDescribe() ;
qexec.close()
```

DESCRIBE queries return a single RDF graph

ASK queries

- ASK

```
import com.hp.hpl.jena.query.* ;

Model model = ... ;

String queryString = " ...ASK... " ;

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
boolean result = qexec.execAsk() ;
qexec.close() ;
```

ASK queries return a boolean

Datasets

- A SPARQL query is made on a **dataset** : a default graph and zero or more named graph

```
String dftGraphURI = "file:default-graph.ttl" ;
List namedGraphURIs = new ArrayList() ;
namedGraphURIs.add("file:named-1.ttl") ;
namedGraphURIs.add("file:named-2.ttl") ;

Query query = QueryFactory.create(queryString) ;
Dataset dataset = DatasetFactory.create(dftGraphURI, namedGraphURIs) ;
QueryExecution qExec = QueryExecutionFactory.create(query, dataset) ;
try {
   ...
}
finally {
   qExec.close() ;
}
```

- Already existing models can also be used

```
DataSet dataSource = DatsetFactory.createMem() ; // Creates an in-memory, modifiable Dataset
dataSource.setDefaultModel(model) ;
dataSource.addNamedModel("http://example/named-1", modelX) ;
dataSource.addNamedModel("http://example/named-2", modelY) ;
QueryExecution qExec = QueryExecutionFactory.create(query,
dataSource) ;
```

ARQ (SPARQL) API
package org.apache.jena.query

Remote Query

- Create a QueryExecution that will access a SPARQL service over HTTP

```
String queryStr = "select distinct ?Concept where {[] a ?Concept} LIMIT
10";
Query query = QueryFactory.create(queryStr);

// Remote execution.
QueryExecution qexec =
QueryExecutionFactory.sparqlService("http://dbpedia.org/sparql", query);
// Set the DBpedia specific timeout.
((QueryEngineHTTP)qexec).addParam("timeout", "10000") ;

// Execute.
ResultSet rs = qexec.execSelect();
ResultSetFormatter.out(System.out, rs, query);
qexec.close();
```