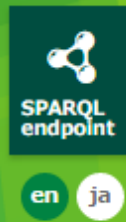


# RIKEN MetaDatabase

This database portal site is used to provide information on RIKEN's various life science databases to order to help researchers around the world make full use of RIKEN's research results.



[Databases](#) [Ontologies](#) [About RIKEN Meta Database](#)

## Javaのプログラムを作成してLODを 効率的に処理してみよう —超入門編—

小林 紀郎

情報基盤センター 計算工学応用開発ユニット

# なぜLODを処理するJavaプログラムを自分で開発すると良いのですか？

- LODは機械可読です（ただ、人間も読めます）
  - プログラムが容易に解釈（処理）可能
  - 標準化されているので専用プログラムは不要



プログラム開発のハードルは比較的低い

- 特にJavaでプログラミングすると良いことは？
  - LODのデータ処理がオブジェクト指向と親和性が高い
  - 高速に動作させるニーズに対応可能
  - 動作環境への依存が少ない

# LOD対応のライブラリーを使えばより効率的にデータ処理が可能です

- Javaのライブラリ Apache Jena を使ってみましょう
  - Apache Jenaは オープンソース
  - LOD (Semantic Web) の読み、書き、加工、公開ができる  
万能ライブラリ
  - Jenaで開発したプログラムを読み解くことで、LODデータ処理の本質的なところが理解可能
- このハンズオンでは、予めサンプルプログラムを用意してあります
  - Javaに詳しくない方でもわかりやすく工夫しています

# Apache Jenaの使い方

1. SPARQL Endpointにアクセスしてデータを取得し、そのデータを処理する
2. Apache Jena上のRDFデータを処理する
  - A) Apache Jenaでデータを作成して処理する
  - B) Apache JenaでRDFデータファイルを読み込み処理する

# 目次

1. Jenaを使う環境を整えましょう
2. RDFの復習
3. SPARQL Endpointにアクセスするプログラムを作ってみましょう
4. RDFデータを作ってみましょう
5. RDFデータを読み込んでみましょう
6. RDFデータを検索してみましょう

# 1.

Jenaを使う環境を整えましょう

# 必要なソフトウェア

- Java SE Development Kit 8 以降  
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Apache Jena 3 以降  
<https://jena.apache.org/>
- Eclipse 最新版  
<https://eclipse.org>

# サンプルプログラム

- Githubにサンプルプログラムが用意されています

<https://github.com/KnowledgeGraphJapan/Apache-Jena-Sample-Programs>

- GitでCloneするには以下のURLを指定してください

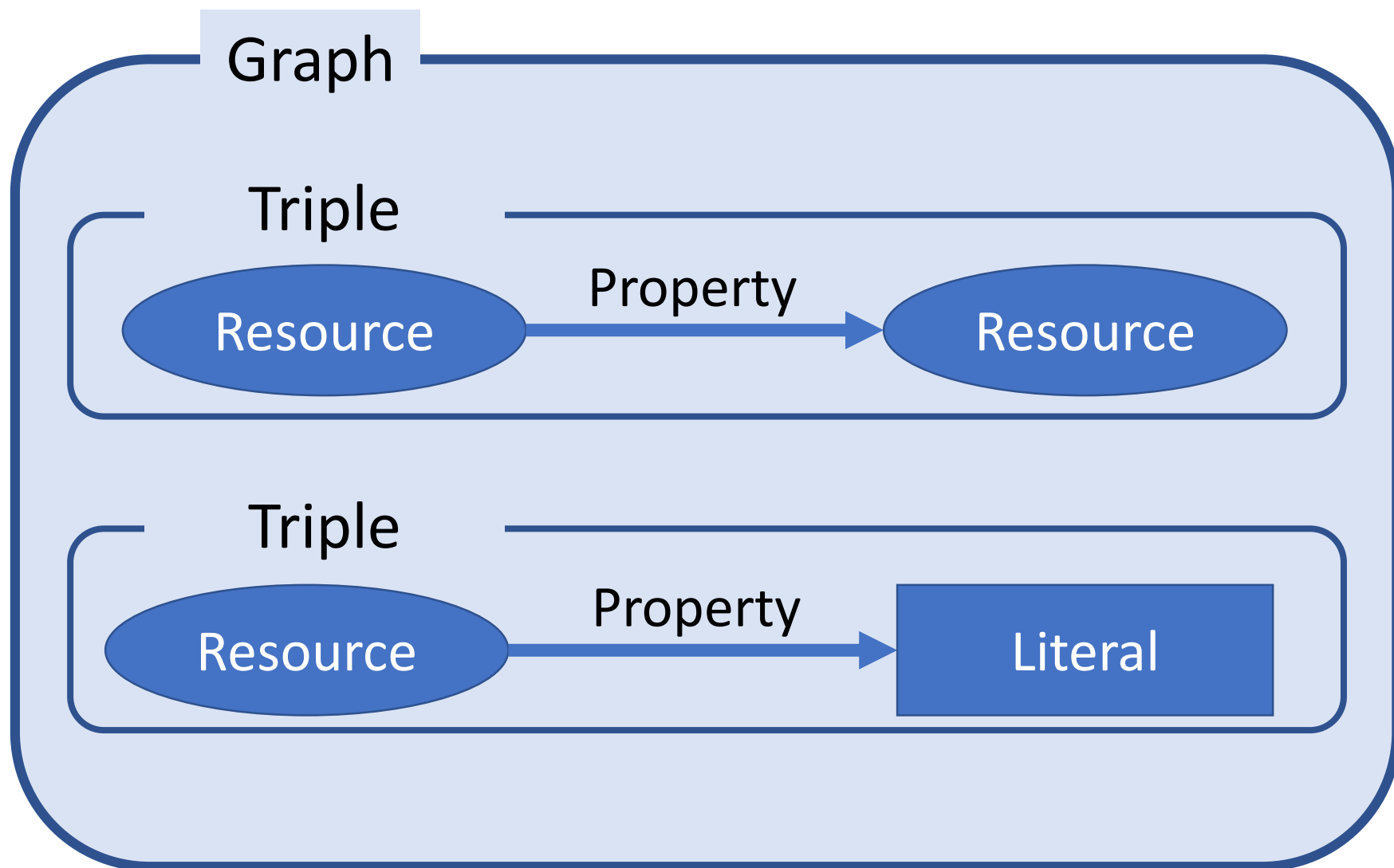
<https://github.com/KnowledgeGraphJapan/Apache-Jena-Sample-Programs.git>



# 2.

## RDFの復習

# RDFを構成するデータ要素



# 目的語がResourceとなるトリプル



URI

URI

URI

<http://riken.jp>

`foaf:homepage`

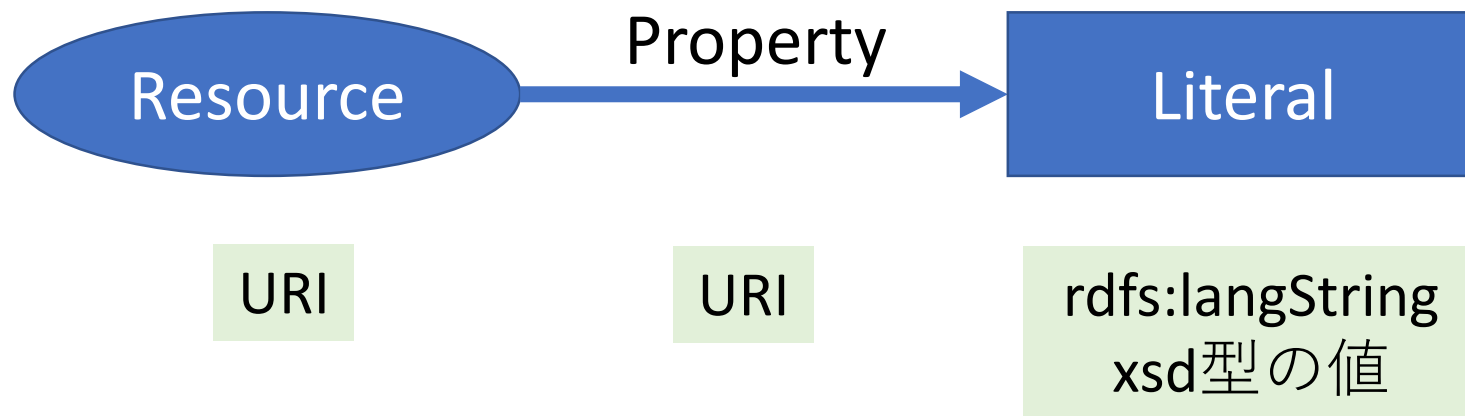
<http://www.riken.jp>



<http://xmlns.com/foaf/0.1/homepage>

foaf は prefixで、<http://xmlns.com/foaf/0.1/>

# 目的語がLiteralとなるトリプル



`http://riken.jp`

`rdfs:label`

`"RIKEN"@en`



`http://www.w3.org/2000/01/rdf-schema#label`

`rdfs` は prefixで、  
`http://www.w3.org/2000/01/rdf-schema#`

# 3.

## SPARQL Endpointにアクセスする プログラムを作ってみましょう

プログラム:

`jp.riken.accc.lod.symposium.sample.SPARQLSearcherForEndpoint`

# SPARQL Endpointに対する検索

```
String endpointURI = "http://metadb.riken.jp/sparql";
```

```
StringBuffer sb = new StringBuffer();
```

```
sb.append("SELECT (count(*) AS ?numTriple)¥n");
```

```
sb.append("WHERE {¥n");
```

```
sb.append("  ?s ?p ?o.¥n");
```

```
sb.append("}");
```

```
String queryString = sb.toString();
```

```
Query query = QueryFactory.create(queryString);
```

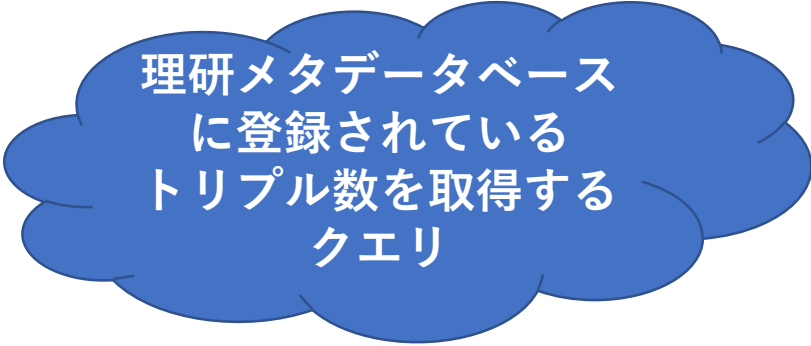
```
QueryExecution qe
```

```
    = QueryExecutionFactory.sparqlService(endpointURI, query);
```

```
ResultSet results = qe.execSelect();
```

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

```
qe.close();
```



理研メタデータベース  
に登録されている  
トリプル数を取得する  
クエリ

表形式で  
結果を出力



# SPARQL Endpointに対する検索

- SELECT句の変数に応じた値の取得も可能
- SPARQLクエリ: SELECT (count(\*) AS ?numTriple)...

```
ResultSet results = qe.execSelect();  
while(results.hasNext()) {  
    QuerySolution solution = results.next();  
    RDFNode node = solution.get("numTriple");  
    System.out.println(node.toString());  
}
```

# 4.

## RDFデータを作ってみましょう

プログラム:

`jp.riken.accc.lod.symposium.sample.RDFGenerator`

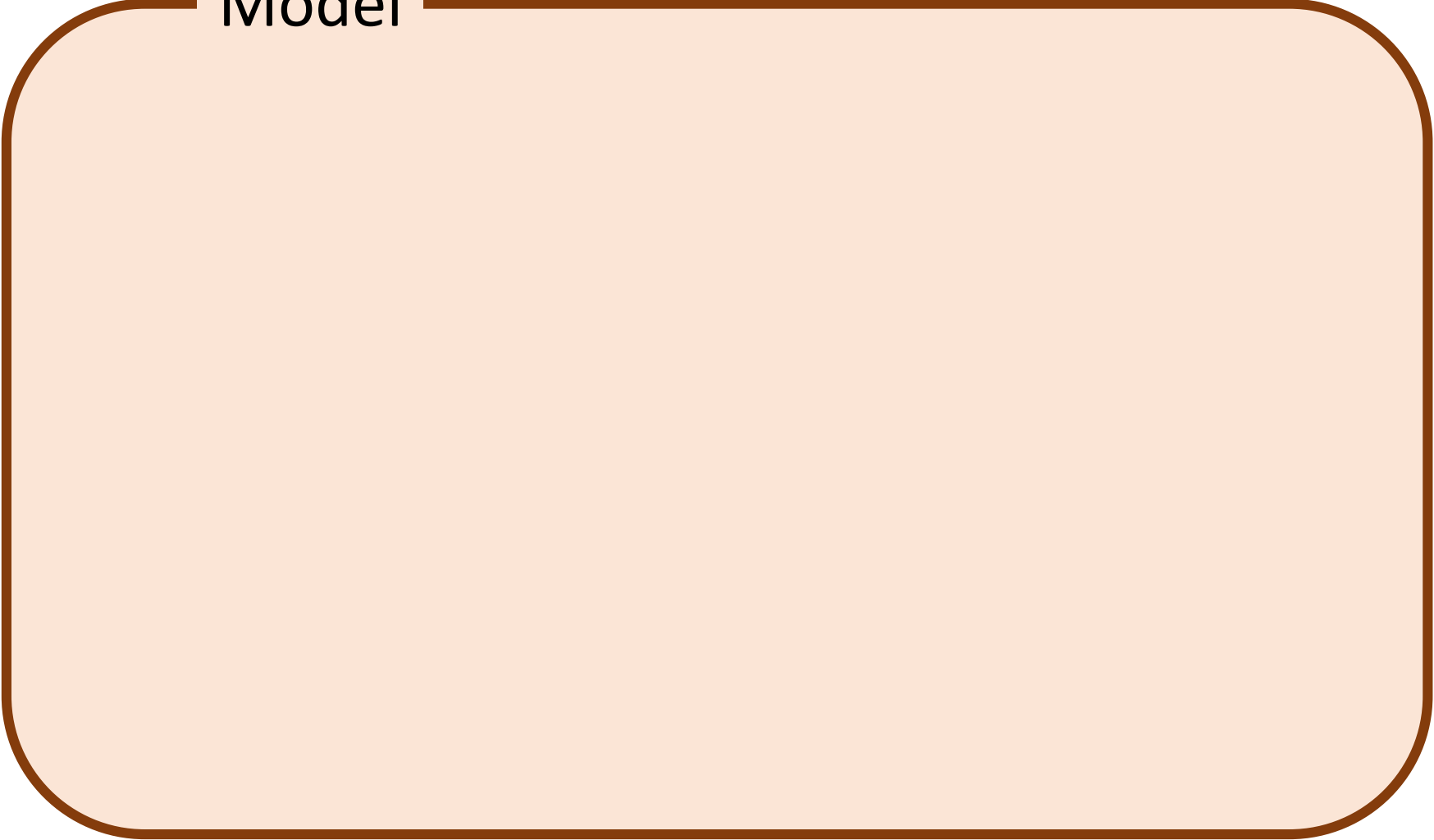


# RDFデータを作る手順

1. 空のモデル(Model: RDF Graphに相当)を作成する
2. トリプルを作成する
  1. 主語リソース(Resource)を作る
  2. プロパティ(Property)を作る
  3. 目的語リソースあるいはリテラル(Literal)を作る
  4. トリプルを作る
3. ファイルに出力する

# 1. 空のモデルを作る

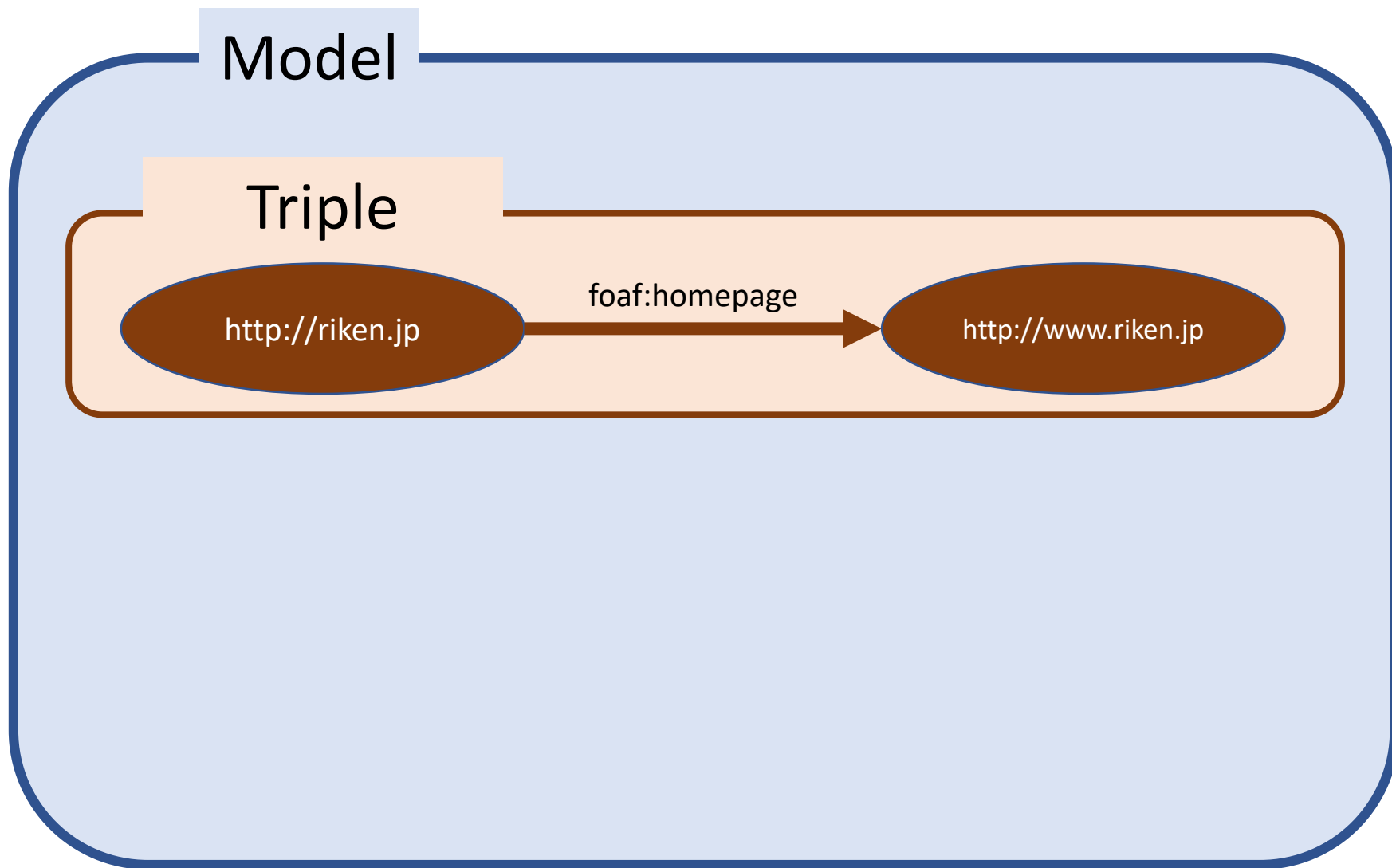
Model



# 1. 空のモデルを作る

```
import org.apache.jena.rdf.model.Model;  
import org.apache.jena.rdf.model.ModelFactory;  
  
// 空のモデルを作る  
// 空のRDF graphを作ることに対応  
Model model =  
    ModelFactory.createDefaultModel();
```

## 2. トリプルを作成する



## 2. トリプルを作成する



1. 主語となるリソース $s$ を作成 `model.createResource`
2. プロパティ $p$ を作成 `model.createProperty`
3. 目的語 $o$ を作成
  - 3.a. リソースの場合 `model.createResource`
  - 3.b. リテラルの場合 `model.createLiteral`
4. トリプルを作成 `s.addProperty( $p,o$ )`

## 2. トリプルを作成する

### 1. 主語リソースを作成する

```
import org.apache.jena.rdf.model.Resource;
```

```
Resource subj = model.createResource(  
    "http://riken.jp");
```

- 1. で作成したmodelを使ってリソースを作成します
- createResourceの引数はURI文字列です

## 2. トリプルを作成する

### 2. プロパティを作成する

```
import org.apache.jena.rdf.model.Property;  
  
Property prop = model.createProperty(  
    "http://xmlns.com/foaf/0.1/homepage");
```

- 1. で作成したmodelを使ってプロパティを作成します
- createPropertyの引数はURI文字列です

## 2. トリプルを作成する

### 3. 目的語リソースを作成する

```
Resource obj = model.createResource(  
    “http://www.riken.jp”);
```

- 1. で作成したmodelを使ってリソースを作成します
- createResourceの引数はURI文字列です



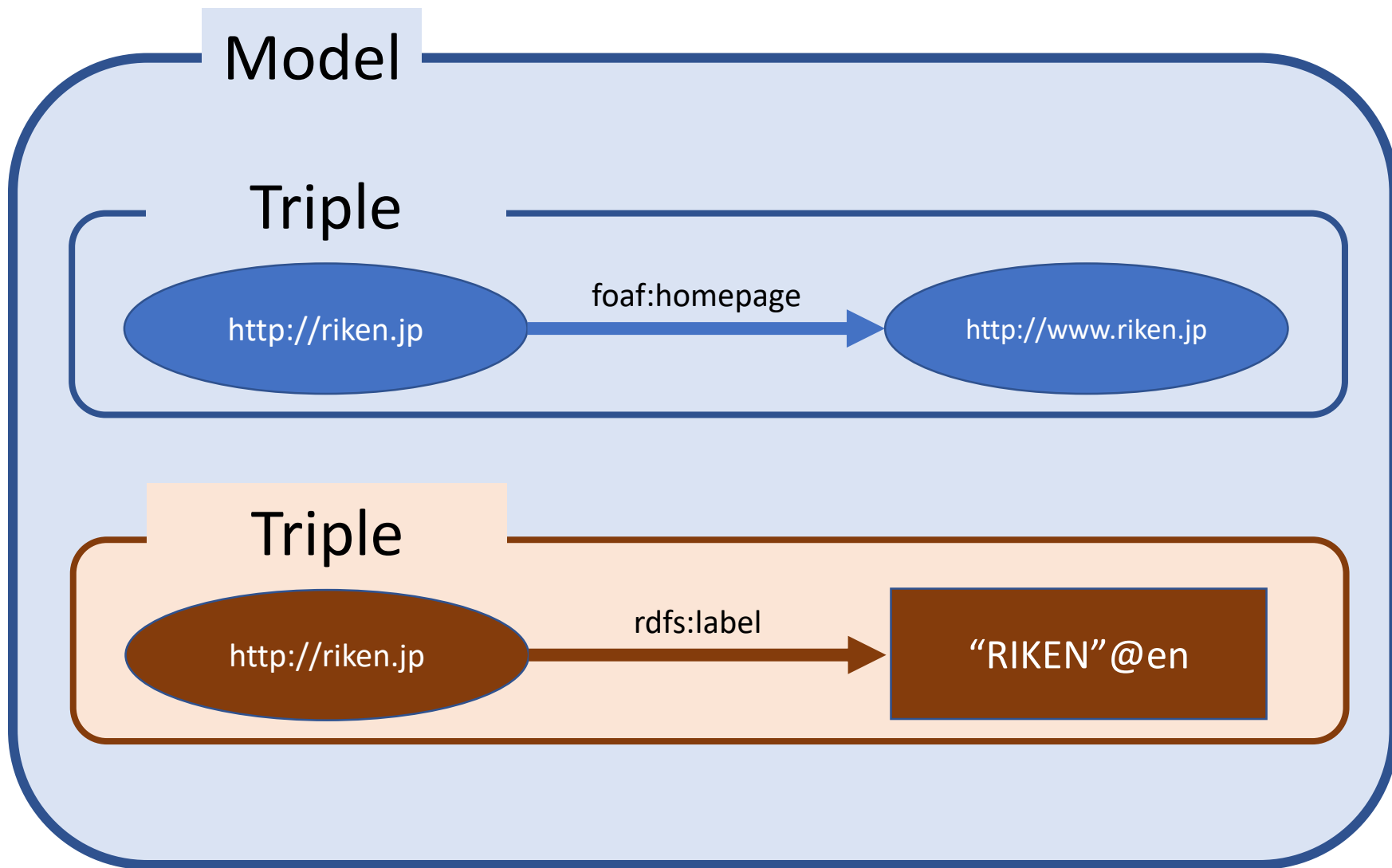
## 2. トリプルを作成する

### 4. トリプルを作成する

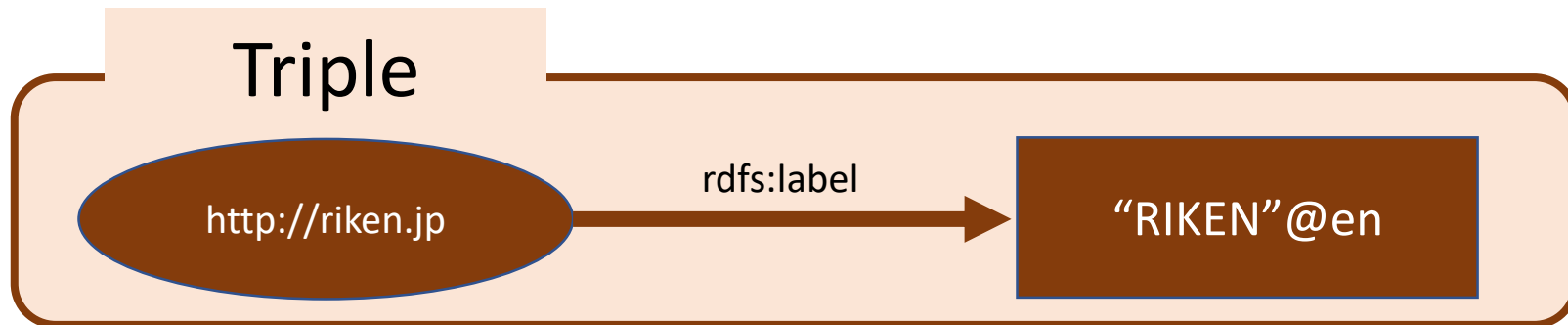
```
subj.addProperty(prop, obj);
```

- 主語subjに述語propと目的語objを関連づけてトリプルを作成します
- addPropertyの引数は述語Propertyと目的語Resourceです

## 2. トリプルを作成する



## 2. トリプルを作成する



1. 主語となるリソース $s$ を作成 `model.createResource`
2. プロパティ $p$ を作成 `model.createProperty`
3. 目的語 $o$ を作成
  - 3.a. リソースの場合 `model.createResource`
  - 3.b. リテラルの場合 `model.createLiteral`
4. トリプルを作成 `s.addProperty( $p, o$ )`

## 2. トリプルを作成する

### 2. プロパティを作成する

```
Property propLabel = model.createProperty(  
    "http://www.w3.org/2000/01/rdf-schema#label");
```

- 1. で作成したmodelを使ってプロパティを作成します
- createPropertyの引数はURI文字列です

## 2. トリプルを作成する

### 3. 目的語リテラルを作成する

```
import org.apache.jena.rdf.model.Literal;
```

```
Literal lit = model.createLiteral(  
    "RIKEN", "en");
```

- 言語付きの文字列をmodelを使って作成します
- createLangLiteralの引数は名前の文字列と言語の文字列です

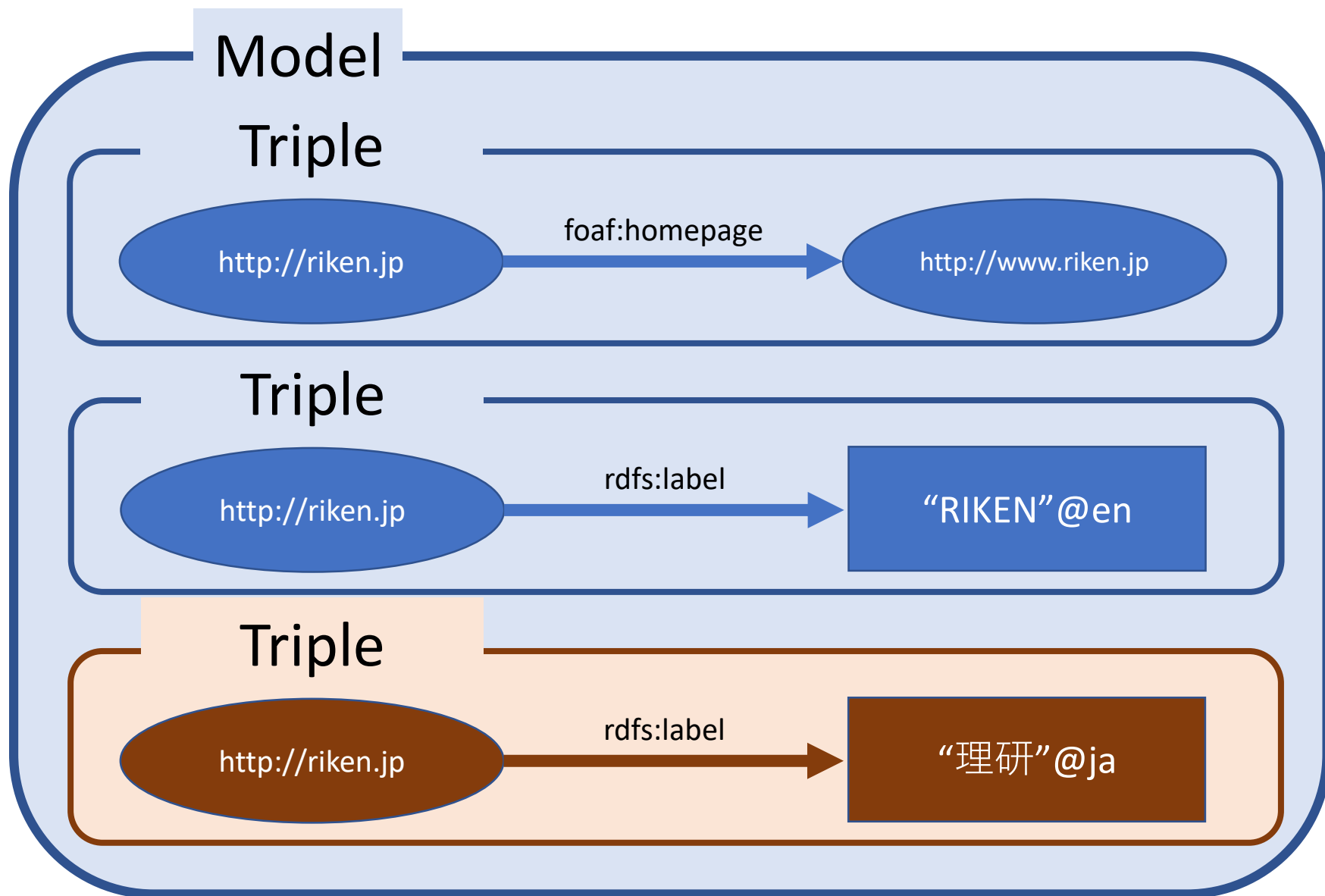
## 2. トリプルを作成する

### 4. トリプルを作成する

```
subj.addLiteral(propLabel, lit);
```

- 主語subjに述語propと目的語litを関連づけてトリプルを作成します
- addLiteralの引数は述語Propertyと目的語Literalです

## 2. トリプルを作成する (演習)



## 2. トリプルを作成する

### 3. 目的語リテラルを作成する

```
Literal litJa =  
    model.createLiteral(“理研”, “ja”);
```

- 言語付きの文字列をmodelを使って作成します
- createLangLiteralの引数は名前の文字列と言語の文字列です



## 2. トリプルを作成する

### 4. トリプルを作成する

```
subj.addLiteral(propLabel, litJa);
```

- 主語subjに述語propと目的語litを関連づけてトリプルを作成します
- addLiteralの引数は述語Propertyと目的語Literalです

# 多様なLiteralデータ型にも対応

- `subj.addLiteral(prop, b);`    `b:boolean`値
  - `subj.addLiteral(prop, c);`    `c:char`値
  - `subj.addLiteral(prop, d);`    `d:double`値
  - `subj.addLiteral(prop, f);`    `f:float`値
  - `subj.addLiteral(prop, l);`    `l:long`値
  - `subj.addLiteral(prop, s);`    `s:string`値
- など..

これらのデータ型についてはリテラルを作成しなくてもトリプルが作成できます。

参考:

<https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Resource.html>

# 整数型リテラルを追加



```
Property propEst =  
    model.createProperty("http://riken.jp/establishedIn");  
subj.addLiteral(propEst, 1917);
```

# 3. ファイルに出力

そのまえに、標準出力に出力

```
model.write(System.out);           // XML形式  
model.write(System.out, "N-TRIPLES"); // N-Triple形式
```

ファイルに出力

```
FileOutputStream fos = new FileOutputStream(new File(file));  
model.write(fos, "N-TRIPLES");  
fos.close();
```

# 最後に

```
model.close();
```

# 出力されたファイルを見てみましょう

- エディタなどのテキストファイルを閲覧するためのソフトウェアを用いて、出力されたファイルを見てください。
- RDFは機械可読ですが、人間も読めます。
- 簡単なRDFなら、プログラムを使わなくても手書きでも作れます。（お勧めはしません）

<http://riken.jp> <http://riken.jp/establishedIn>  
"1917"^^<http://www.w3.org/2001/XMLSchema#long> .

<http://riken.jp> <http://www.w3.org/2000/01/rdf-schema#label> "理研"@ja .

<http://riken.jp> <http://www.w3.org/2000/01/rdf-schema#label> "RIKEN"@en .

<http://riken.jp> <http://xmlns.com/foaf/0.1/homepage> <http://www.riken.jp> .

# 5.

RDFデータを読み込んでみましょう

プログラム:

`jp.riken.accc.lod.symposium.sample.RDFReader`



# RDFファイルを読み込む

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

- 空モデルを作成した後、readメソッドでファイルを読み込むだけです
- *file* は文字列で、ファイル名、URLなどが指定可能

# 6.

## RDFデータを検索してみましよう

プログラム:

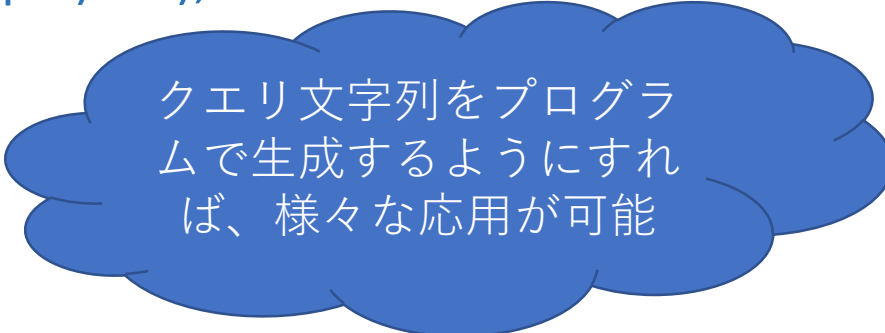
`jp.riken.accc.lod.symposium.sample.SPARQLSearcherForEndpoint`

`jp.riken.accc.lod.symposium.sample.SPARQLSearcherForModel`

`jp.riken.accc.lod.symposium.sample.JenaSearcher`

# SPARQL Endpointに対する検索(再)

```
String endpointURI = "http://metadb.riken.jp/sparql";
StringBuffer sb = new StringBuffer();
sb.append("SELECT (count(*) AS ?numTriple)¥n");
sb.append("WHERE {¥n");
sb.append("  ?s ?p ?o.¥n");
sb.append("}");
String queryString = sb.toString();
Query query = QueryFactory.create(queryString);
QueryExecution qe
    = QueryExecutionFactory.sparqlService(endpointURI, query);
ResultSet results = qe.execSelect();
ResultSetFormatter.out(System.out, results, query);
qe.close();
```



クエリ文字列をプログラムで生成するようになれば、様々な応用が可能

# SPARQL Endpointに対する検索(再)

- SELECT句の変数に応じた値の取得も可能
- SPARQLクエリ: SELECT (count(\*) AS **?numTriple**)...

```
ResultSet results = qe.execSelect();
while(results.hasNext()) {
    QuerySolution solution = results.next();
    RDFNode node = solution.get("numTriple");
    System.out.println(node.toString());
}
```

# SPARQLで検索する

Modelに対してSPARQL検索する

```
Query query = QueryFactory.create(queryString);  
QueryExecution qe  
    = QueryExecutionFactory.create(query, model);  
ResultSet results = qe.execSelect();  
ResultSetFormatter.out(System.out, results, query);  
qe.close();
```

- `queryString`はSPARQLクエリの文字列です

プログラム:

- `jp.riken.accc.lod.symposium.sample.SPARQLSearcherForModel`

# SPARQLで検索する

ホームページを探しましょう

```
SELECT ?hp ?label
```

```
WHERE{
```

```
  ?s <http://http://xmlns.com/foaf/0.1/homepage> ?hp.
```

```
  ?s <http://www.w3.org/2000/01/rdf-schema#label> ?label.
```

```
}
```

- 上記クエリをqueryStringとして実装

# 実行結果

```
<http://riken.jp> <http://xmlns.com/foaf/0.1/homepage> <http://www.riken.jp> .  
<http://riken.jp> <http://www.w3.org/2000/01/rdf-schema#label> "RIKEN"@en .  
<http://riken.jp> <http://www.w3.org/2000/01/rdf-schema#label> "理研"@ja .  
<http://riken.jp> <http://riken.jp/establishedIn> "1917"^^<http://www.w3.org/2001/XMLSchema#long> .
```

```
-----  
SELECT ?hp ?label  
WHERE {  
  ?s <http://xmlns.com/foaf/0.1/homepage> ?hp.  
  ?s <http://www.w3.org/2000/01/rdf-schema#label> ?label.  
}
```

```
-----  
-----  
| hp           | label       |  
=====
```

<http://www.riken.jp>   "RIKEN"@en
<http://www.riken.jp>   "理研"@ja

```
-----
```

# Jenaのメソッドを使って高速処理

- SPARQLはクエリを解釈し実行するので、高速処理には不向きです。
- Jenaに備わっているメソッドを使うとより高速にデータ処理が行えます。
- ただ、少しコードが複雑になります。

プログラム:

- `jp.riken.accc.lod.symposium.sample.JenaSearcher`



# 全トリプルを取得

```
StmtIterator it = model.listStatements();
while( it.hasNext() ) {
    Statement st = it.next();
    Resource subj = st.getSubject();
    Property prop = st.getPredicate();
    RDFNode objNode = st.getObject();
    if( objNode.isResource() ) {
        // resource
    }else{
        if( objNode.isLiteral() {
            // literal
        }
    }
}
```

# 全主語を取得

```
ResIterator rit = model.listSubjects();  
while( rit.hasNext() ) {  
    Resource res = rit.next();  
    System.out.println(res.getURI());  
}
```

# 主語を指定し全トリプルを取得

```
Resource res = model.getResource("http://riken.jp");  
StmtIterator sit  
    = model.listStatements(  
        res, (Property)null, (RDFNode)null);  
while( sit.hasNext() ) {  
    Statement st = sit.next();  
    Property prop = st.getPredicate();  
    RDFNode objNode = st.getObject();  
    ...  
}
```

# 主語と述語を指定し全目的語を取得

```
Resource res = model.getResource("http://riken.jp");
Property prop = model.getProperty(
    "http://www.w3.org/2000/01/rdf-schema#label");
NodeIterator nit =
    model.listObjectsOfProperty(res, prop);
while( nit.hasNext() ) {
    RDFNode objNode = nit.next();
    System.out.println(objNode.toString());
}
```

# 他にもデータ取得方法があります

- 以下のModelのドキュメントを参照してください  
<https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Model.html>
- 目的語を指定してトリプルを取得
- プロパティを指定してトリプルを取得  
など

# 理研でもJenaでツール開発しています

- 理研メタデータベース
  - ユーザーインターフェース
  - ExcelテーブルからRDFに変換するツール
  - 統計情報を取りまとめるツール
- LOD Surfer (DBCLS, 阪大との共同開発)  
(旧SPARQL Builder)
  - 公開Endpointのデータ構造を取得、取りまとめるツール
  - データのつながりを探索する機能

などなど...

# まとめ

- Apache Jenaのプログラミング 入門編を解説
  - JenaはLODのデータ処理を行う万能ツール
  - 高速動作が要求されるツール類開発に好適
  - LOD (RDF)データの生成、検索、加工を中心に解説
  - 応用や細かな解説は一切省略しました
- ご自身のご興味、データに合わせてプログラムを改良してください
- LODを楽しく使えるツールを開発してください

# 演習

- DBpediaに対してSPARQL検索を行い、面白いデータ表を作成してください。

Jenaでプログラミングしてもいいですし、SPARQLのみで実現しても構いません。

参考: Linked Open Data ハッカソン関西 SPARQLクエリ集

<http://wp.lodosaka.jp/tool/sparqlquery/>

クエリ集をそのまま実行しても良い演習ですが

例えば、大阪府を兵庫県に変えたり

自治体数を駅数に変えたり

複数のクエリを組み合わせてみたり

いろいろ工夫してみてください。