SPARQL で RDF データを検索する

SPARQL と Jena Toolkit によって、セマンティック Web を開く

Philip McCarthy
Developer
SmartStream Technologies Ltd

2005年 10月 01日

RSS のような RDF フォーマットで保存されるデータが増えてきたために、特定な情報を見つけるための簡単な方法が必要になってきました。強力な、新しいクエリー言語である SPARQL は、その要求に応えるものであり、これを利用することによって、RDF の山の中から必要なデータを容易に見つけることができます。SPARQL の機能を訪ねるツアーに出かけましょう。Jena Semantic Web Toolkit を使って、皆さん自身の Java アプリケーションから SPARQL クエリー行う方法を学びます。

RDF (Resource Description Framework) を利用すると、データの中央集中を避け、分散化することが可能となります。RDF モデル同士は容易に合体することができ、シリアル化した RDF は単純に、HTTP で交換することができます。アプリケーションは、Web 上にある複数の RDF データ・ソースに疎結合されます。例えば PlanetRDF.com では、コンテンツを RSS 1.0 フィードとして RDF で提供する作成者からの weblog を配信しています。作成者のフィードの URL 自体は、bloggers.rdf と呼ばれる RDF グラフに保持されています。

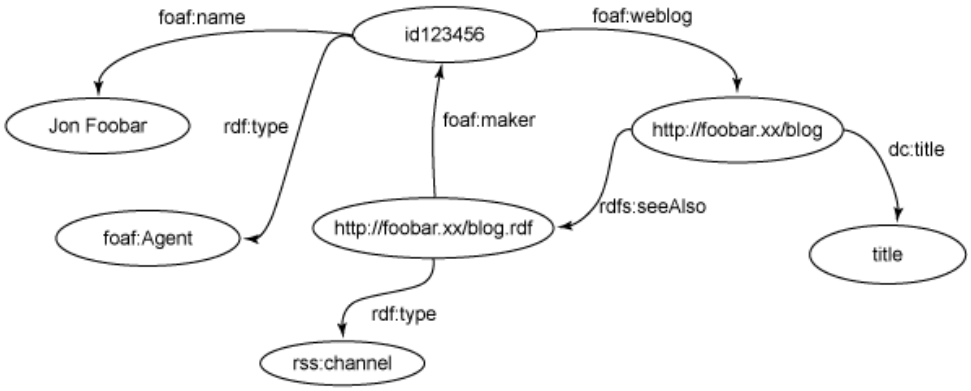
しかし、必要なデータを RDF グラフ内で見つけ、操作するには、どうすればよいのでしょう。SPARQL (SPARQL Protocol And RDF Query Language) が、現在、W3C のワーキング・ドラフトとして議論されています。SPARQL は、これまでの RDF クエリー言語 (rdfDB や RDQL、SeRQL など) の上に構築されており、SPARQL 自体が、幾つか貴重な新機能を含んでいます。この記事では、PlanetRDF を駆動する 3 つのタイプの RDF グラフ、(貢献者 (contributors) を記述する FOAF、その RSS 1.0 フィード、そして bloggers graph) を使って、データに対して SPARQL クエリーが行うことの幾つかを紹介します。SPARQL の実装は、様々なプラットフォームや言語用のものが存在しています。この記事では、Java プラットフォーム用の Jena Semantic Web Toolkit に焦点を当てることにします。

この記事では、読者が RDF に関して実際的な知識を持っていること、また Dublin Core や FOAF、RSS 1.0 などの RDF ボキャブラリーに慣れていることを想定しています。また、Jena Semantic Web Toolkit に関しても多少の経験があることも想定しています。こうした技術に慣れるためには、下記の参考文献にあるリンクを調べてください。

単純な SPARQL クエリーを分解する

まず、PlanetRDF の bloggers.rdf モデルを見てみましょう。これは非常に単純であり、FOAF と Dublin Core ボキャブラリーを使って、各ブログ貢献者の名前やブログ・タイトル、URL、RSS フィード記述などを提供します。図 1 は、単一の貢献者に対する基本的なグラフ構造を示しています。完全なモデルでは単純に、集約するブログそれぞれに対して、この構造を繰り返します。

図 1. bloggers.rdf での、単一の貢献者に対する基本的なグラフ構造



では、この bloggers モデルに対する、非常に単純な SPARQL クエリーを見てみましょう。このクエリーをリスト 1 に示しますが、これはつまり、「Jon Foobar という名前の人物によるブログの URL を探せ」と言っています。

リスト 1. 貢献者のブログを見つけるための SPARQL クエリー

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?url
FROM  <bloggers.rdf>
WHERE  {
    ?contributor foaf:name "Jon Foobar" .
    ?contributor foaf:weblog ?url .
}
```

クエリーの第 1 行は単純に、FOAF 名前空間に対する `PREFIX` を定義しており、参照する度に全部をタイプし直さなくて済むようになっています。`SELECT` 文は、クエリーが何を返すべきか (この場合は、`url` という名前の変数) を規定しています。SPARQL 変数には、`?` あるいは `$` という接頭辞が付きます。この 2 つは交換可能ですが、この記事では、`?` を使うことにします。`FROM` はオプションの文であり、使用すべきデータセットの URI を提供します。ここでは、単にローカル・ファイルを指していますが、Web 上のどこかにあるグラフの URL を示すこともできます。最後に `WHERE` 文は、Turtle ベースの構文を使って表現された、3 つセット (triple) パターンの連続から成っています。こうした 3 つセットを合わせて、グラフ・パターン と呼びます。

クエリーは、グラフ・パターンの 3 つセットと、モデルの一致を調べます。モデルのノードに一致したグラフ・パターンの変数のバインディングは、それぞれクエリー・ソリューション となり、`SELECT` 文の中に名前を挙げられた変数の値が、クエリー結果の一部となります。

この例では、`WHERE` 文のグラフ・パターンにある最初の 3 つセットが、「Jon Foobar」の `foaf:name` プロパティーを持つノードに一致し、`contributor` という名前の変数にバインドされま

す。bloggers.rdf モデルでは、`contributor` が、図 1 の最上部にある `foaf:Agent` 空白ノードと一致します。グラフ・パターンの 2 番目の 3 つセットは `contributor` の `foaf:weblog` プロパティーのオブジェクトに一致します。これは `url` 変数にバインドされ、クエリー・ソリューションを形成します。

Jena で SPARQL を使う

Jena での SPARQL サポートは現在、ARQ というモジュールを利用して行われます。ARQ クエリー・エンジンは、SPARQL をサポートすることに加え、RDQL、あるいは ARQ 自体のクエリー言語で表現されたクエリーを構文解析することができます。ARQ は活発な開発が行われており、まだ標準の Jena ディストリビューションの一部にはなっていません。しかし、Jena の CVS リポジトリーから、あるいは自己完結のダウンロードとして、入手することができます。

ARQ を動くようにするのは簡単です。単純に最新の ARQ ディストリビューション (参考文献にリンクがあります) を入手し、解凍し、環境変数 `ARQROOT` を ARQ ディレクトリーを指すようにするだけです。また、read をリストアし、ARQ bin ディレクトリーの内容に対してパーミッションを実行する必要があるかも知れません。bin ディレクトリーはコマンドラインから ARQ を呼び出すラッパー・スクリプトを含んでいるため、実行パスに bin ディレクトリーを追加すると便利です。全て準備が出来たかどうかを確認するために、コマンドラインから `sparql` を呼び、使用メッセージが見えることを確認します。これらの全ステップはリスト 2 に説明されています。リスト 2 では、UNIX ライクのプラットフォームで、あるいは Windows の下での Cygwin で作業していることを想定しています。(ARQ には、Windows の下で使うための `.bat` スクリプトも付属していますが、その使い方は、この例での使い方と少し異なっています。)

リスト 2. Jena ARQ を使うために環境を設定する

```
$ export ARQROOT=~/ARQ-0.9.5
$ chmod +rx $ARQROOT/bin/*
$ export PATH=$PATH:$ARQROOT/bin
$ sparql
 Usage: [--data URL] [exprString | --query file]
```

コマンドラインから SPARQL クエリーを実行する

これで、SPARQL クエリー (リスト 3 を見てください) を実行する準備ができました。ここでは、リスト 1 のデータセットとクエリーを使います。このクエリーは、使用すべきグラフの指定に `FROM` キーワードを使うので、`sparql` コマンドに対してクエリー・ファイルの位置を提供すればよいだけです。ただし、グラフはクエリーの `FROM` 文にある相対 URL を使って規定されるため、クエリーはグラフを含むディレクトリーから実行する必要があります。

リスト 3. sparql コマンドを使って簡単なクエリーを実行する

```
$ sparql --query jon-url.rq
----------------------------
| url                      |
============================
| <http://foobar.xx/blog>  |
----------------------------
```

場合によると、クエリーに `FROM` 文を含まないようにした方が妥当です。そうすれば、クエリーが実行された時に、グラフがクエリーに渡されるようになります。アプリケーション・コードから

SPARQL を使用する場合には、データセットをクエリーにバインドしないようにするのは良い習慣です。それによって、例えば同じくエリーが、別のグラフで再利用できるようになります。グラフは、オプション付きのコマンドライン、`sparql --data URL` の実行時に規定されます (ここで `URL` はグラフの位置です) 。この URL は、ローカル・ファイルの位置、あるいはリモート・グラフの Web アドレスです。

Jena API で SPARQL クエリーを実行する

独立のクエリーを実行する場合には、コマンドラインの `sparql` ツールも便利ですが、Java アプリケーションでは直接、Jena の SPARQL 機能を使用することができます。`com.hp.hpl.jena.query` パッケージの中にあるクラスを利用して、Jena で SPARQL クエリーを作成し、実行することができます。`QueryFactory` を使えば、一番単純です。`QueryFactory` には、ファイル、あるいは `String` から、テキスト型クエリーを読むための様々な `create()` メソッドがあります。`create()` メソッドは、構文解析されたクエリーをカプセル化した、`Query` オブジェクトを返します。

次のステップは、クエリーを 1 回実行することを表すクラスである、`QueryExecution` のインスタンスを作ることです。`QueryExecution` を得るためには、`QueryExecutionFactory.create(query, model)` を呼び、実行すべき `Query` と、実行対象となる `Model` を渡します。クエリーに対するデータはプログラム的に提供されるため、このクエリーには `FROM` 文が必要ありません。

`QueryExecution` には、幾つか異なる実行方法があり、それぞれ異なるタイプのクエリーを行います (囲み記事、「他のタイプの SPARQL クエリー」を見てください)。単純な `SELECT` クエリーの場合、`execSelect()` を呼ぶと、`ResultSet` が返ります。`ResultSet` を使うと、クエリーで返された `QuerySolution` それぞれに対して繰り返しを行うことができ、バインドされた変数の値それぞれにアクセスできるようになります。一方 `ResultSetFormatter` は、クエリー結果を様々なフォーマットで出力するために使います。

リスト 4 は、こうしたステップを簡単にまとめる方法を示しています。これは、bloggers.rdf に対してクエリーを実行し、その結果をコンソールに出力します。

リスト 4. Jena の API を使って簡単なクエリーを実行する

```
// Open the bloggers RDF graph from the filesystem
InputStream in = new FileInputStream(new File("bloggers.rdf"));
// Create an empty in-memory model and populate it from the graph
Model model = ModelFactory.createMemModelMaker().createModel();
model.read(in,null); // null base URI, since model URIs are absolute
in.close();
// Create a new query
String queryString =
 "PREFIX foaf: <http://xmlns.com/foaf/0.1/> " +
 "SELECT ?url " +
 "WHERE {" +
 "      ?contributor foaf:name \"Jon Foobar\" . " +
 "      ?contributor foaf:weblog ?url . " +
 "      }";
Query query = QueryFactory.create(queryString);
// Execute the query and obtain results
QueryExecution qe = QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
// Output query results
ResultSetFormatter.out(System.out, results, query);
// Important - free up resources used running the query
qe.close();
```

より複雑なクエリーを書く

ここまでは、簡単な SPARQL クエリーを実行するための、2 つの方法を見てきました。1 つはコマンドラインの `sparql` ユティリティーを使い、もう 1 つは Jena API で Java コードを使う方法です。このセクションでは、SPARQL の機能をさらに紹介し、もっと複雑なクエリーが行えることを説明します。

RDF は多くの場合、半構造化データ (semi-structured data) を表すために使われます。これは、同じタイプを持つ、モデル中の 2 つのノードが、異なるプロパティー・セットを持つかも知れないことを意味します。例えば、FOAF モデルでの、ある人物の記述には、e メール・アドレスしか無いかも知れません。あるいは、実際の名前に加えて、IRC ニックネーム、その人が写った写真の URL 等々、を含んでいるかも知れません。

クエリー結果を洗練する

SPARQL には、クエリー結果を洗練されたものにするために、`DISTINCT` や `LIMIT`、`OFFSET`、`ORDER BY` などのキーワードがあります。これらは SQL での同じキーワードと、ほぼ同じような動作をします。`DISTINCT` は、`SELECT` クエリーとの組み合わせで、`SELECT DISTINCT` という形でのみ使用できます。これによって、重複したクエリー・ソリューションが結果セットから取り除かれ、残ったソリューションは、それぞれ固有なものとなります。その他のキーワードはどれも、クエリーの `WHERE` 文の後に置かれます。`LIMITn` は、クエリーから返される結果の数を n に制限し、一方 `OFFSETn` は、最初の n 個の結果を省略します。`ORDER BY ?var` は、例えば語彙的に var がストリング・リテラルであれば、結果を ?var の自然順でソートします。また、`ASC[?var]` と `DESC[?var]` は、ソート方向を規定するために使われます。

もちろん、クエリーの中で `DISTINCT` や `LIMIT`、`OFFSET`、`ORDER BY` を組み合わせることもできます。例えば、`ORDER BY DESC[?date] LIMIT 10` を使って、RSS フィードの中にある、最も新しい 10 のエントリーを探すことができます。

リスト 5 は、Turtle 構文で表現した、非常に小さな FOAF グラフを示しています。これには 4 人の架空人物が含まれていますが、それぞれの人物の記述には、別々のプロパティー・セットが入っています。

リスト 5. 4 人の人物を記述した FOAF グラフ

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name         "Jon Foobar" ;
    foaf:mbox         <mailto:jon@foobar.xx> ;
    foaf:depiction    <http://foobar.xx/2005/04/jon.jpg> .
_:b foaf:name         "A. N. O'Ther" ;
    foaf:mbox         <mailto:a.n.other@example.net> ;
    foaf:depiction    <http://example.net/photos/an-2005.jpg> .
_:c foaf:name         "Liz Somebody" ;
    foaf:mbox_sha1sum "3f01fa9929df769aff173f57dec2fe0c2290aeea"
_:d foaf:name         "M Benn" ;
    foaf:depiction    <http://mbe.nn/pics/me.jpeg> .
```

オプション一致 (optional matches)

例えば、リスト 5 のグラフで記述された全ての人物の名前と、その人物の写真へのリンクがある場合には、そのリンクも返すクエリーを書きたい、としましょう。グラフ・パターンに `foaf:depiction` を含む `SELECT` クエリーは、3 つのソリューションのみを見つけます。Liz Somebody には `foaf:name` がありますが、`foaf:depiction` プロパティーがありません。クエリーに一致するためには両方が必要なため、Liz Somebody はソリューションになりません。

ここで、SPARQL の `OPTIONAL` キーワードという助けが利用できます。オプション・ブロック (Optional blocks) が追加のグラフ・パターンを定義し、ソリューションは、一致しなくても拒否されず、一致する可能性がある場合にはグラフにバインドされるのです。リスト 6 は、リスト 5 の FAOF データにある各人物の `foaf:name` を見つけてから、オプションとして、付随する `foaf:depiction` を見つけるクエリーを示しています。

リスト 6. オプション・ブロックで FOAF データをクエリーする

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?depiction
WHERE { ?person foaf:name ?name .
        OPTIONAL {
            ?person foaf:depiction ?depiction .
        } .
      }
```

リスト 7 は、リスト 6 でのクエリーを実行した結果を示しています。全クエリー・ソリューションが人物の名前を含んでいますが、オプションのグラフ・パターンは、`foaf:depiction` プロパティーが存在する場合にのみバインドされます。存在しない場合には、単純にソリューションから省略されます。この点に関しては、このクエリーは、SQL での左外部結合 (left outer join) に似ています。

リスト 7. リスト 6 でのクエリーの結果

```
-------------------------------------------------------------
| name           | depiction                                |
=============================================================
| "A. N. O'Ther" | <http://example.net/photos/an-2005.jpg> |
| "Jon Foobar"   | <http://foobar.xx/2005/04/jon.jpg>      |
| "Liz Somebody" |                                          |
| "M Benn"       | <http://mbe.nn/pics/me.jpeg>            |
-------------------------------------------------------------
```

オプション・ブロックは、リスト 6 に示した単一の 3 つセットだけではなく、任意のグラフ・パターンを含むことができます。オプション・パターンがクエリー・ソリューションの一部を成すためには、オプション・ブロックでのクエリー・パターン全体が一致する必要があります。クエリーが複数のオプション・ブロックを持つ場合には、それぞれのブロックが独立に動作します。それぞれが、ソリューションから省略されたり、ソリューションの中に入ったりします。また、オプション・ブロックはネストすることもできます。その場合、内部のオプション・ブロックは、外部のオプション・ブロックのパターンがグラフに一致する場合のみ、考慮の対象となります。

代替一致 (alternative matches)

FOAF グラフは、人物の e メール・アドレスを使って、その人を固有識別します。人によっては、プライバシーの観点から、e メール・アドレスのハッシュコードを使いたがります。プレーン・テキストの e メール・アドレスは `foaf:mbox` プロパティーを使って表現されますが、e メール・アドレスのハッシュコードは、`foaf:mbox_sha1sum` プロパティーを使って表現されます。両者は通常、人物の FOAF 記述では、相互排他的です。そうした場合には、SPARQL の代替一致 (alternative matches) 機能を使って、どちらのプロパティーであっても、得られる方のプロパティーを返すクエリーを書くことができます。

代替一致の定義は、複数の代替グラフ・パターンを記述し、それらを UNION キーワードを使って組み合わせることによって行います。リスト 8 に示すクエリーは、リスト 5 の FOAF グラフにある各人物の名前と共に、foaf:mbox、あるいは foaf:mbox_sha1sum を見つけます。M Benn は、foaf:mbox プロパティーも foaf:mbox_sha1sum プロパティーも持っていないため、クエリー・ソリューションではありません。代替一致では OPTIONAL グラフ・パターンとは対照的に、どれか 1 つが、完全に、クエリー・ソリューションと一致する必要があります。UNION での分岐の両方が一致する場合には、2 つのソリューションが生成されます。

リスト 8. 代替一致でのクエリーと、その結果

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?mbox
WHERE {
  ?person foaf:name ?name .
  {
      { ?person foaf:mbox ?mbox } UNION { ?person foaf:mbox_sha1sum ?mbox }
  }
}
-------------------------------------------------------------------------
| name                 | mbox                                           |
=========================================================================
| "Jon Foobar"         | <mailto:jon@foobar.xx>                         |
| "A. N. O'Ther"       | <mailto:a.n.other@example.net>                 |
| "Liz Somebody"       | "3f01fa9929df769aff173f57dec2fe0c2290aeea" |
-------------------------------------------------------------------------
```

値に対する制限

SPARQL での FILTER キーワードは、バインドされた変数値に対して制限を課すことによって、クエリー結果を制限します。値に対する、こうした制限は、ブール値に対して評価される論理表現であり、場合によっては論理演算子 && や論理演算子 || と組み合わせられます。例えば、名前のリストを返すクエリーであれば、与えられた正規表現に一致する名前のみを返すように、フィルターによって修正することができます。あるいは、リスト 9 に示すように、2 つの日付の間に公開される RSS フィードにあるアイテムであれば、アイテムの公開日に制限を加えるフィルターを使って見つけることができます。リスト 9 はまた、SPARQL の XPath 流のキャスト機能がどのように使われるか (ここでは、date 変数を XML スキーマの dateTime 値にキャストします) 、また、リテラルの date ストリングと同じデータ・タイプを、^^xsd:dateTime によって規定する方法も示しています。これによって、クエリーで使われるのは標準のストリング比較ではなく、日付の比較であることが保証されるようになります。

リスト 9. フィルターを使って、2005 年 4 月に公開された RSS フィード・アイテムを検索する

```
PREFIX rss:   <http://purl.org/rss/1.0/>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX  dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?item_title ?pub_date
WHERE {
      ?item rss:title ?item_title .
      ?item dc:date ?pub_date .
      FILTER xsd:dateTime(?pub_date) >= "2005-04-01T00:00:00Z"^^xsd:dateTime &&
             xsd:dateTime(?pub_date) < "2005-05-01T00:00:00Z"^^xsd:dateTime
      }
```

複数のグラフを扱う

ここまでに示したクエリーは全て、単一の RDF グラフから成るデータセットのみを扱いました。SPARQL の用語を使うと、こうしたクエリーは、バックグラウンド・グラフ (background graph) に対して実行したことになります。バックグラウンド・グラフというのは、クエリーの `FROM` 文や `sparql` コマンドの `--data` スイッチによって、あるいは Jena の API を使う場合に、モデルを `QueryExecutionFactory.create()` に渡すことによって規定されるものを言います。

他のタイプの SPARQL クエリー

SPARQL は、この記事で使用している SELECT クエリーの他にも、3 つのクエリー・タイプをサポートしています。ASK は、データセットの中にクエリーのグラフ・パターンと一致するものが何かある場合、単純に「yes」を返し、無い場合には「no」を返します。DESCRIBE は、グラフ・パターンの中に一致があるノードに関連した情報を含むグラフを返します。例えば、`DESCRIBE ?person WHERE { ?person foaf:name "Jon Foobar" }` は、Jon Foobar に関するモデルからの 3 つセットを含むグラフを返します。最後に CONSTRUCT は、各クエリー・ソリューションに対するグラフ・パターンを出力するために使われます。これによって、クエリー結果から直接、新しい RDF グラフが作られるようになります。RDF グラフに対する CONSTRUCT クエリーは、ある意味で、XML データの XSL 変換のようなものと考えることができます。

SPARQL ではバックグラウンド・グラフに加えて、任意の数の名前付きグラフ (named graphs) をクエリーすることができます。バックグラウンド・グラフは URI によって規定され、クエリー内では、それぞれ別々です。名前付きグラフの使い方を検証する前に、どうやって名前付きグラフをクエリーに入れるかを説明しましょう。名前付きグラフもバックグラウンド・グラフと同様、クエリー自体の中で、`FROM NAMED <URI>` を使って規定することができます (`URI` がグラフを規定します) 。あるいは、`--named URL` を付けた `sparql` コマンドで名前付きグラフを提供することもできます (URL がグラフの位置を示します)。最後に、Jena の `DataSetFactory` クラスを使って、名前付きグラフをプログラム的にクエリーするように規定する方法もあります。

名前付きグラフは、SPARQL クエリーの中で、グラフの URI あるいは変数名と共に、`GRAPH` キーワードと組み合わせて使います。このキーワードの後には、対象となるグラフに一致するグラフ・パターンが続きます。

特定なグラフで一致を見つける

グラフの URI (あるいは、既にグラフの URI にバインドされた変数) で `GRAPH` キーワードが使われると、そのグラフ・パターンは、その URI によって識別されるグラフの全てに適用されます。指定されたグラフで一致が見つかると、その一致がクエリー・ソリューションの一部となります。リスト 10 では、名前付き FOAF グラフが 2 つ、クエリーに渡されます。クエリー・ソリューションは、両方のグラフで見つかる人々の名前です。2 つの FOAF グラフそれぞれで、人物を表すノードは空白ノードであり、これらの空白ノードを含むグラフ内にしかスコープがないことには注意してください。つまり、同じ人物ノードが、クエリー中の名前付きグラフの両方に存在することはできず、従って別々の変数 (`x` と `y`) を使って表す必要があるのです。

リスト 10. 名前付き FOAF グラフ 2 つの中で記述された人物を見つけるクエリー

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM NAMED <jon-foaf.rdf>
FROM NAMED <liz-foaf.rdf>
WHERE {
  GRAPH <jon-foaf.rdf> {
    ?x rdf:type foaf:Person .
    ?x foaf:name ?name .
  } .
  GRAPH <liz-foaf.rdf> {
    ?y rdf:type foaf:Person .
    ?y foaf:name ?name .
  } .
}
```

パターンを含むグラフを見つける

GRAPH にはもう一つの使い方として、バウンドされていない変数で GRAPH を探す使い方があります。この場合には、クエリー中にある、名前付きグラフのそれぞれに対してグラフ・パターンが適用されます。そのパターンが、名前付きグラフの 1 つに一致すれば、そのグラフの URI は GRAPH の変数にバインドされていることになります。リスト 11 の GRAPH 文は、クエリーに与えられた、名前付き FOAF グラフにある人物ノードそれぞれに一致します。一致した人物名は name 変数にバインドされており、graph_uri 変数は、そのパターンに一致したグラフの URI にバインドされます。リスト 11 には、このクエリーの結果も示してあります。A. N. O'Ther という人物は、jon-foaf.rdf と liz-foaf.rdf の両方で記述されているため、1 つの名前が 2 回一致しています。

リスト 11. 別々の人物を記述するグラフがどれかを判定する

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?graph_uri
FROM NAMED <jon-foaf.rdf>
FROM NAMED <liz-foaf.rdf>
WHERE {
    GRAPH ?graph_uri {
      ?x rdf:type foaf:Person .
      ?x foaf:name ?name .
    }
}
-------------------------------------------------------
| name                | graph_uri                     |
=======================================================
| "Liz Somebody"      | <file://.../jon-foaf.rdf>     |
| "A. N. O'Ther"      | <file://.../jon-foaf.rdf>     |
| "Jon Foobar"        | <file://.../liz-foaf.rdf>     |
| "A. N. O'Ther"      | <file://.../liz-foaf.rdf>     |
-------------------------------------------------------
```

XML フォーマットでのクエリー結果

SPARQL では、クエリー結果を XML として、「SPARQL Variable Binding Results XML Format」として知られる単純なフォーマットで返すことができます。この、スキーマ定義のフォーマットは、XML ツールやライブラリーと RDF クエリーとの間の橋渡しとして動作します。

この機能には、幾つかの使い方が考えられます。SPARQL クエリーの結果を、XSLT を通して Web ページや RSS フィードに変換したり、XPath を使って結果にアクセスしたり、あるいは

結果文書を SOAP クライアントや AJAX クライアントに返すこともできます。クエリー結果を XML として出力するためには、`ResultSetFormatter.outputAsXML()` メソッドを使うか、あるいはコマンドラインで `--results rs/xml` を規定します。

背景データと名前付きグラフを組み合わせる

クエリーが、名前付きグラフと組み合わせた背景データを使う場合もあります。リスト 12 は、PlanetRDF.com からライブ集約された RSS フィードを背景データとして使い、私独自の FOAF プロファイルを含む名前付きグラフと組み合わせて、背景データをクエリーします。考え方としては、私が知っている blogger がポストしたアイテムの内、最新の 10 アイテムを見つけることによって、カスタム化したフィードを作る、ということです。クエリーの最初の部分では、FOAF ファイルの中で私を表すノードを見つけてから、(グラフによると) 私が知っている人物の名前を見つけます。2 番目の部分では、こうした人達によって作られたアイテムを、RSS フィードの中で探します。最後に、見つかったセットはアイテムの作成日でソートされ、10 アイテムのみを結果として返すように制限されます。

リスト 12. カスタム化した、ライブの PlanetRDF フィードを取得する

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rss:  <http://purl.org/rss/1.0/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>

SELECT ?title ?known_name ?link

FROM <http://planetrdf.com/index.rdf>
FROM NAMED <phil-foaf.rdf>

WHERE {
        GRAPH <phil-foaf.rdf> {
          ?me foaf:name "Phil McCarthy" .
          ?me foaf:knows ?known_person .
          ?known_person foaf:name ?known_name .
        } .

        ?item dc:creator ?known_name .
        ?item rss:title ?title .
        ?item rss:link ?link .
        ?item dc:date ?date.
      }

ORDER BY DESC[?date] LIMIT 10
```

このクエリーは単純に、タイトルと名前と URL のリストを返すだけですが、もっと複雑なクエリーを使えば、一致したアイテムの中にある全データを抽出することもできます。SPARQL の XML 結果フォーマット (囲み記事「XML フォーマットでのクエリー結果」を見てください) と、XSL スタイルシートと組み合わせて使うことによって、ブログ・アイテムの HTML ビューをカスタム化したり、さらには、別の RSS フィードを作ったりすることもできます。

まとめ

この記事で取り上げた例によって、皆さんは SPARQL の基本機能や構文を理解でき、また SPARQL が RDF アプリケーションにもたらす利点も理解できたと思います。また、オプション一致や代替一致を補助として使うことによって、半構造化されている、実世界の RDF グラフの持つ特質を SPARQL で取り扱う方法を学びました。名前付きグラフを使った例では、SPARQL で複数のグラフ

を組み合わせることによって、クエリーのオプションが広がることを示しました。さらに、Jena の API を使えば、Java コードから SPARQL クエリーを実行することが簡単であることも見てきました。

SPARQL には他にも、この記事では書ききれないほど、様々なことを行うことができます。下記の参考文献を活用して、さらに SPARQL について学んで下さい。SPARQL の仕様を見れば、SPARQL の組み込み機能や演算子、クエリー形式や構文について学ぶことができ、また、SPARQL クエリーの例も数多く挙げられています。

もちろん、SPARQL について学ぶために一番良いのは、自分でクエリーを書いてみることです。皆さんも Web から幾つか RDF データを取得し、Jena ARQ モジュールをダウンロードし、実験を始めてみてください。

著者について

Philip McCarthy

Philip McCarthy は J2EE そしてフロントエンドの技術を専門とする Web Developer (Web 開発者) です。Orange での Consumer Internet Applications での実績を含む 4 年間の Java プログラミングの経験を誇ります。現在、彼は SmartStream Technologies にて金融関連の Web アプリケーションを開発中です。