Web NDL Authorities SPARQL API仕様書

国立国会図書館

2014年3月31日作成
2018年3月31日, 2013年3月31日改訂

目 次

1 Web NDLA SPARQL APIの概要

Web NDL Authorities（以下 Web NDLA）では、典拠の情報を RDF（Resource Description Framework）のデータとして格納しています。このデータは SPARQL（RDF 照会言語）によって検索することができます。

この章では、基本的な RDF のデータ構造と SPARQL による検索方法の概要を説明します。SPARQL に関する詳細は第 2 章、API からの返戻の詳細は第 3 章、Web NDLA の典拠レコード全体を俯瞰した検索例は第 4 章で取り上げます。

1.1 SPARQL 照会 API

1.1.1 基本的な SPARQL クエリ

Web NDLA の典拠データは、**SPARQL クエリ**を用いて自由に検索できます。次の例は、件名「図書館」の典拠 URI を調べるクエリです。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj rdfs:label "図書館"
}
```

このクエリを次の手順を用いて API 経由で Web NDLA に送ることで、検索結果が得られます。

1.1.2 リクエスト URI とパラメータ

Web NDLA への SPARQL による照会は、次の URI（エンドポイント）に対して行ないます。

```
http://id.ndl.go.jp/auth/ndla
```

対応するパラメータは表 1 の 2 つです。

表 1: Web NDLA API のパラメータ

パラメータ	値
query	SPARQL 照会言語によるクエリを URL エンコードしたもの
output	結果フォーマット（`xml	json	turtle*`）*turtle はDESCRIBE、CONSTRUCT のみ

XML 形式で結果を得るためのリクエストは、次のような形になります。

```
http://id.ndl.go.jp/auth/ndla?query={URL エンコードしたクエリ}&output=xml
```

前項のクエリ例を URL エンコードして `query` パラメータの値とすると、次の形になります。

```
http://id.ndl.go.jp/auth/ndla?query=PREFIX+rdfs%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F
2000%2F01%2Frdf-schema%23%3E%0D%0ASELECT+*+WHERE+%7B%0D%0A%09%3Fsubj+rdfs%3Alabel
+%22%E5%9B%B3%E6%9B%B8%E9%A4%A8%22%0D%0A%7D%0D%0A&output=xml
```

SPARQL クエリには、上記のように値を調べる SELECT のほか、条件に一致するデータの有無を調べる ASK、条件に合う値を用いて新たな RDF グラフを構築する CONSTRUCT、リソースに関する説明 RDF グラフを取得する DESCRIBE の 4 つの形態があります。

結果フォーマット（output）として turtle を指定できるのは DESCRIBE、CONSTRUCT クエリのみです。

SPARQL クエリの記述方法については第 2 章で詳しく取り上げます。4 つのクエリ形態については、2.8 で詳述します。

1.1.3 結果フォーマット

リクエストの結果は、クエリの形態に従って、変数バインディングリスト（SELECT）、RDF グラフ（DESCRIBE、CONSTRUCT）もしくは真偽値（ASK）が返されます。変数バインディングリストおよび真偽値の結果フォーマットは output パラメータの値により、表 2 のようになります。

表 2: Web NDLA API の output パラメータ

output パラメータ値	フォーマット
xml	SPARQL Query Results XML Format に従った XML
json	SPARQL 1.1 Query Results JSON Format に従った JSON

前節のリクエスト（output=xml）の結果、次の XML が得られます。

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="subj"/>
  </head>
  <results>
    <result>
      <binding name="subj">
        <uri>http://id.ndl.go.jp/auth/ndlsh/00573385</uri>
      </binding>
    </result>
  </results>
</sparql>
```

XML フォーマット、JSON フォーマットの詳細はそれぞれ 3.1、3.2 で詳述します。

RDF グラフの場合の結果フォーマットは、表 3 のとおりです。

表 3: Web NDLA API の RDF グラフフォーマット

output パラメータ値	フォーマット
xml	RDF/XML 形式
json	RDF/JSON 形式
turtle	Turtle 形式

1.2 典拠の RDF グラフと SPARQL

Web NDLA における典拠レコードは RDF のグラフとして表現されています。SPARQL では「グラフの中で検索したいパターン」を用いてクエリを記述します。RDF グラフ中のパターンを SPARQL クエリとして表す方法の概要を説明します。

1.2.1 典拠 RDF グラフの基本形

RDF は、あるものごと（典拠レコードなど＝主語）が何らかの属性や性質（述語）を持ち、述語には値がある（目的語）という、主語－述語－目的語の 3 者の関係（**RDF トリプル**）を用いて基本的な情報を表します。この 3 者は、ウェブ上で他の情報と組合せても曖昧さが生じないように、**URI**[1]を用いて名前付け（識別）します。また、目的語には文字列データを**リテラル**として与えることもあります。

RDF トリプルの集合を **RDF グラフ**と呼びます。RDF グラフにおいて、同じ URI を持つトリプルは連結され、情報が連鎖して行きます（図 1）[2]。

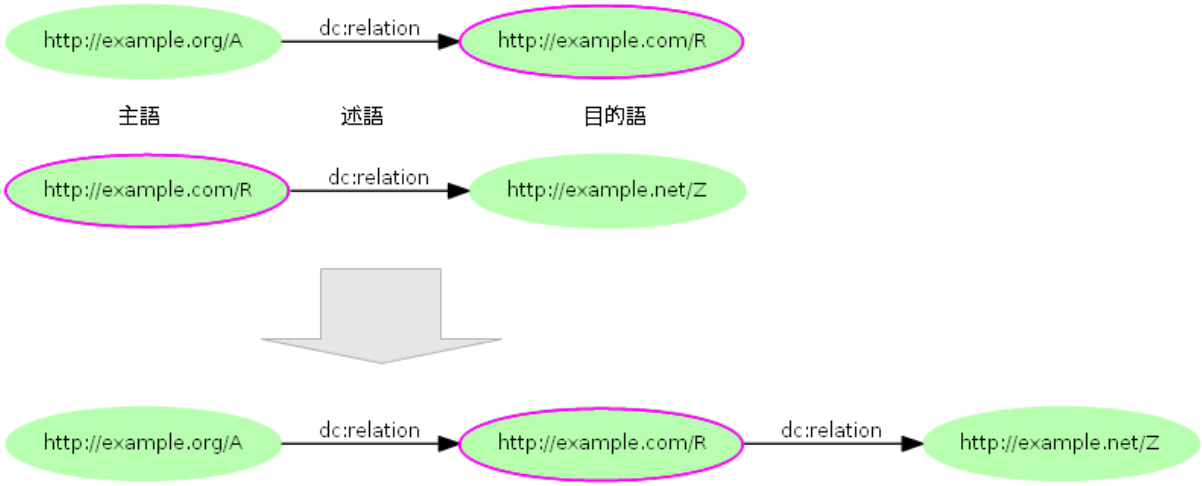

図 1: 共通の URI を介してグラフがつながる

典拠レコードはこの RDF で表現されています。各典拠には URI が与えられ、典拠間の関係（上位語、別名など）が RDF で記述されるため、典拠レコードの RDF がつながり、大きな RDF グラフを

[1]RDF での名前付けには URI を拡張して ASCII 以外の漢字なども含められる IRI（Internationalized Resource Identifier）を用いることができますが、ここではより一般的な用語である URI としています。本仕様書での URI はすべて IRI と読み替えることができます。

[2]なお、グラフの中で連結の役割のみを担い、外部から直接アクセスする必要のない結節点は、URI で識別しない「空白ノード」とする場合もあります。空白ノードはグラフ内限定のローカル識別子によって連結されます。

形成します（図 2）。

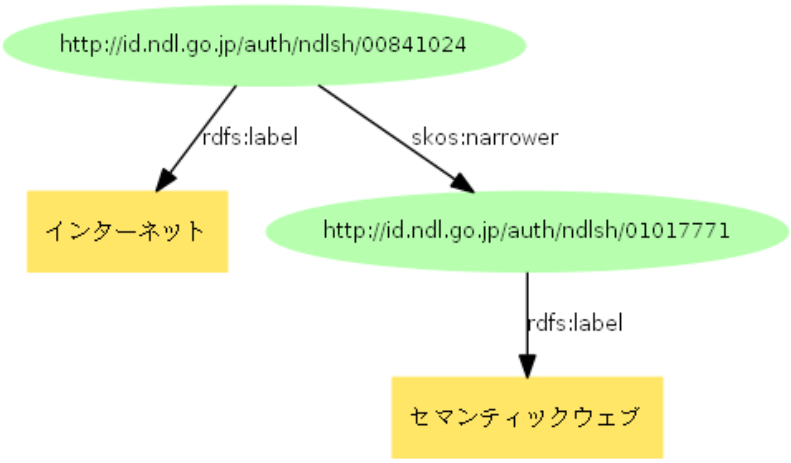

図 2: 「セマンティックウェブ」の標目 URI（http://…01017771）は「インターネット」（http://…00841024）の skos:narrower の目的語でもあるため、両者のグラフがつながる

また VIAF や LC authorities など URI を持つ外部の典拠も RDF によって参照しているため、Web NDLA の RDF グラフは国立国会図書館典拠を超えた広がりを持ち、LOD（Linked Open Data）クラウドの一部となっています（典拠レコード RDF グラフの全体像については第 4 章で説明します）。

1.2.2 部分グラフパターンと検索

SPARQL では、調べたい未知のデータ（変数）を含む RDF グラフの部分パターンを用いて、対象となる RDF グラフ全体からこのパターンに合致する部分を取り出し、変数に対応する値を結果として得ることができます。

例えば、ラベル（`rdfs:label`）が「インターネット」である典拠の下位語（`skos:narrower`）となる典拠を調べたいとき、この部分グラフパターンは図 3 のようになります。

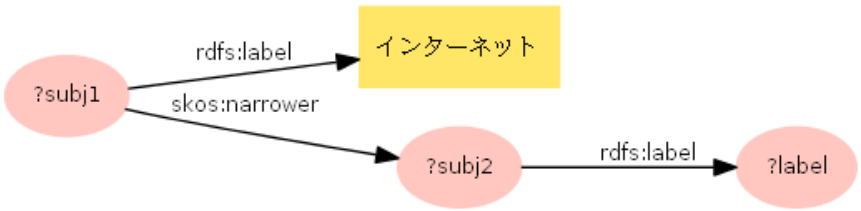

図 3: 下位語の典拠レコードを subj2、そのラベルを label という変数で表現した部分グラフパターン

この部分グラフパターンに合致するものを Web NDLA から取り出し、対応する変数を調べると表 4 の結果が得られます。

1.2.3 SPARQL クエリの記述

SPARQL クエリは、照会する部分グラフパターンを、Turtle に準ずる構文を用いて記述します。未知のノード（主語、目的語）やプロパティ（述語）は'?' で始まる変数で表現します。前項図 3 の部

表 4: 部分グラフパターンを用いて検索した結果

subj2	label
http://id.ndl.go.jp/auth/ndlsh/01017771	セマンティックウェブ
http://id.ndl.go.jp/auth/ndlsh/00969901	バーチャルプライベートネットワーク
http://id.ndl.go.jp/auth/ndlsh/00865280	イントラネット

分グラフパターンは次のように表されます。

```
?subj1 rdfs:label "インターネット" ;
    skos:narrower ?subj2 .
?subj2 rdfs:label ?label .
```

このパターンを用いて検索するためには、

1. 最初にグラフパターンで用いる接頭辞をキーワード PREFIX によって URI と対応付け[3]、
2. キーワード SELECT に続けて取得したい変数を空白区切りで列挙し（SQL の場合にカラム名を列挙するのと同様)、
3. キーワード WHERE に続けてパターン全体を{}で囲む

という形でクエリを構築します。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?subj2 ?label
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

ここでは表 4 の値を得るために SELECT に続けて 2 つの変数?subj2 ?label を列挙しています。グラフパターンで用いた変数すべての値を取得するときは、次のように変数名の代わりに*とすることもできます[4]。

```
SELECT * WHERE {
 ...
```

ASK、CONSTRUCT、DESCRIBE クエリの場合も、PREFIX 句[5]、WHERE 句は同様で、SELECT の部分をそれぞれの記述で置き換えます（2.8 参照)。

[3]以下の例では説明の要点を分かりやすくするために PREFIX 句を省略する場合がありますが、接頭辞を用いるときには PREFIX による対応付けが必要です。

[4]WHERE の前に改行がある例とない例を示していますが、SPARQL 構文では改行は空白文字として扱われるので、どちらでも違いはありません。

[5]SELECT などのキーワードとその対象となる記述をまとめて句（clause）と呼びます。

2 SPARQL照会言語

Web NDLA はARC2 ライブラリ[6]を用いてSPARQL Query Language for RDF のバージョン1.0[7]をサポートしています。本章では、Web NDLA で利用できるSPARQL クエリについて説明します。

2.1 基本構文

SPARQL ではRDF Turtle に準ずる構文によって検索対象とするグラフのパターンを記述します。グラフパターンはトリプルパターンの集合で構成されます。

2.1.1 トリプルパターン

トリプルパターンは、Turtle のトリプル記述要素に変数を加えた形で記述できます。変数は主語、述語、目的語のどの位置にも置くことができます。

- <>で囲んだ値は**URI**[8]となります。絶対URI のほか、BASE 句と組合せて相対URI を用いることもできます。
- URI に代えて、接頭辞:ローカル名の形の**接頭辞付名**を用いることができます。接頭辞はPREFIX 句でURI にマッピングしておく必要があります。
- ""もしくは'' で囲んだ値は**リテラル**となります。引用符に続けて@で言語コード、^^でデータ型を指定することができます。
- _:で始める接頭辞付名は**空白ノード**となります。[] で囲む形の空白ノードの省略記述も用いることができます[9]。
- **変数**は?もしくは$で始まる英数字です[10]。アンダーバー（_）を含めることはできますが、ハイフン（-）は使えません。変数はクエリ全体をスコープとします。

型（クラス）を表す述語rdf:type は、Turtle と同様にa の1 文字で簡略表記できます。他にRDF 構文語彙を用いる必要がなければ、この記法を利用することで、rdf:のためのPREFIX 句を省略できます。

次は、foaf:Person 型を持つことをa で簡略表記し、典拠の中で人物実体（4.1 参照）を表すリソースを調べる例です。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
    ?who a foaf:Person
}
```

[6] https://github.com/semsol/arc2
[7] http://www.w3.org/TR/rdf-sparql-query/
[8] SPARQL においても識別子はIRI を用いることができますが、一般的な用語であるURI としておきます。
[9] グラフパターンにおける空白ノードは、特定の空白ノードを表すものではなく、変数と同様の役割を持ちます。したがって、RDF グラフ中ではURI やリテラルであるノードの位置に空白ノードパターンを記述してもマッチします（ただし値を取り出すことはできません）。
[10] SPARQL 仕様では変数名に仮名漢字も用いることができるようになっていますが、Web NDLA では実装していません。

このクエリを a を用いずに書くと次のようになります。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
    ?who rdf:type foaf:Person
}
```

なお、クエリ内のリテラル、URI 以外の場所に#を置くと、それ以降行末までがコメントとして扱われます。またクエリ内では改行および TAB 文字は空白文字として扱われます。

2.1.2 リテラルのデータ型と言語タグ

リテラルは単純な文字列の他に、データ型（日付、整数など）を持ったり、言語タグを持つことがあります。文字列部分が同じでも、データ型、言語タグが違うと、SPARQL の検索では異なる値として扱われるので注意が必要です。

- データ型は、Web NDLA では用いていません。他のデータセットでは、たとえばレコード作成日を`"2013-12-15"^^xsd:date` のようなデータ型付リテラルで表現している場合がありますが、Web NDLA では`"2013-12-15"`の形です。
- 言語タグは、Web NDLA では「よみ」にのみ用いています。例えば「図書館」の場合、カナ読みを`"トショカン"@ja-Kana`、ローマ字読みを`"Toshokan"@ja-Latn` としており、言語タグによって読みの種類を区別できます。

2.1.3 グラフパターン

トリプルパターンを列挙することで**グラフパターン**を表現できます。グラフパターンを構成する複数のトリプルパターンはピリオド（.）で区切ります[11]。

Turtle と同様に、; を用いて同じ主語の反復を省略した述語－目的語のリストを記述できます。また同じく , を用いて同じ述語の反復を省略した目的語のリストを記述できます。

```
?subj rdfs:label ?label ;
    skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DK341> ,
        <http://id.ndl.go.jp/class/ndc9/694.5> .
```

上の例は、同じ変数`?subj` を主語とする 3 つのトリプルパターンです。さらに最後の行では述語`skos:relatedMatch` の反復を省略し、, で目的語を続けています。省略なしで記述すると次のようになります。

[11] Turtle と異なり、トリプルパターンの末尾にピリオドは必須ではありません。したがってグラフパターンの末尾は、ピリオドを省略することもできます。

```
?subj rdfs:label ?label .
?subj skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DK341> .
?subj skos:relatedMatch <http://id.ndl.go.jp/class/ndc9/694.5> .
```

2.1.4 フィルタ

変数値に対して FILTER 句で値の制約を設定できます。たとえば次のような記述で、ラベルに「夏目」を含む標目に対象を限定することができます（フィルタについては 2.6 で詳述します）。

```
?uri rdfs:label ?label .
FILTER regex(?label, "夏目")
```

2.2 グラフパターンのグループ

SPARQL でのパターンマッチの最小単位となるグラフパターンを**基本グラフパターン**（Basic Graph Pattern）と呼びます。フィルタを含む一連のトリプルパターンが基本グラフパターンを構成し、別のグラフパターンが出現したところで基本グラフパターンが区切られます。

1 つ以上のグラフパターンを{}でまとめたものを**グループグラフパターン**と呼びます。クエリの WHERE 句はキーワードに続く{}内全体が 1 つのグループグラフパターンとなります。

グループグラフパターンは入れ子にすることができます。

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    {?subj2 rdfs:label ?label }
}
```

上の例では、WHERE 句を構成するグラフパターン内に、1 つの基本グラフパターンと 1 つのグループグラフパターンが含まれ、内側のグループグラフパターンは 1 つの基本グラフパターンで構成されています。

2.2.1 グラフパターンとフィルタ

フィルタは、FILTER 句が含まれるグループグラフパターン全体を対象として結果の値を制約します。同じグループグラフパターン内であれば、FILTER 句をどこに記述しても結果は同じになります。したがって、

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
```

```
    ?subj2 rdfs:label ?label .
    FILTER regex(?label, "ネット")
}
```

と

```
WHERE {
    FILTER regex(?label, "ネット")
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

では同じ結果が得られます。

2.2.2 グラフパターンと空白ノード ID

空白ノード ID は基本グラフパターンをスコープとし、同じラベルを異なる基本グラフパターンで用いることはできません[12]。従って、

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower _:s2 .
    _:s2 rdfs:label ?label .
}
```

のように書くことはできますが、次の例はエラーになります。

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower _:s2 .
    {_:s2 rdfs:label ?label }
}
```

2 番目の例では、内側の{}によって基本グラフパターンが 2 つに分かれており、両者で同じ空白ノード ID の_:s2 を共有することはできないからです。

2.3 OPTIONAL による非必須条件

基本グラフパターンによるクエリでは、その中に含まれるすべての変数に値がマッチする必要があります。たとえば次の例では、「インターネット」の下位語（skos:narrower）であってもその関連語（skos:related）がない件名は?subj2 にマッチしません。

[12]空白ノードは変数と同様の働きをしますが、変数はクエリ全体をスコープとしている点が異なります。

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label ;
        skos:related ?subj3 .
    ?subj3 rdfs:label ?rels .
}
```

必須ではない変数を含む部分グラフをひとつのグループグラフパターンとし、キーワード OPTIONAL で必須パターンに連結することで、その変数をオプション扱いすることができます。

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    OPTIONAL {
        ?subj2 skos:related ?subj3 .
        ?subj3 rdfs:label ?rels .
    }
}
```

上のクエリでは、「インターネット」の下位語をすべて列挙し、さらに関連語を持つ場合のみその値も取得することができます。

2.3.1 複数の OPTIONAL

OPTIONAL パターンは複数連結することができます。

```
パターン 1 OPTIONAL {パターン 2} OPTIONAL {パターン 3}
```

このとき、これらのパターンは左結合され、次のようにグループ化されているのと同じように実行されます。

```
{パターン 1 OPTIONAL {パターン 2}} OPTIONAL {パターン 3}
```

2.3.2 OPTIONAL と FILTER

オプションのパターンは FILTER によって値を制約することもできます。次の例は、「インターネット」の下位語を調べ、さらにその下位語に「情報」を含む関連語があれば合わせて取り出す、というクエリになります。

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    OPTIONAL {
        ?subj2 skos:related ?subj3 .
        ?subj3 rdfs:label ?rels .
        FILTER regex(?rels, "情報")
    }
}
```

フィルタの対象はグループグラフパターンであるため、次の例のようにFILTERキーワードをOPTIONALパターンの外にだすと、WHERE句全体に対してフィルタリングを行ないます。

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    OPTIONAL {
        ?subj2 skos:related ?subj3 .
        ?subj3 rdfs:label ?rels .
    }
    FILTER regex(?rels, "情報")
}
```

この場合、すべての結果に対して?relsが「情報」を含むかどうか調べるため、下位語の関連語がない（?relsの値がない）ものは条件を満たさず、結果から省かれてしまいます。

2.4 UNIONによる代替条件

複数のグラフパターンのいずれかにマッチする結果を得るためには、グループグラフパターンをUNIONで結合します。

たとえば、国立国会図書館分類表（NDLC）のND633（電気通信応用 - データ通信）もしくは日本十進分類法第9版（NDC9）の547.483（データ通信網）のいずれかに該当する件名を調べるためには、次のクエリを用いることができます。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
prefix xl: <http://www.w3.org/2008/05/skos-xl#>
prefix ndl: <http://ndl.go.jp/dcndl/terms/>
```

```
SELECT *
WHERE {
    {
        ?subj
            skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/ND633> ;
            rdfs:label ?dcndl .
    } UNION {
        ?subj
            skos:relatedMatch <http://id.ndl.go.jp/class/ndc9/547.483> ;
            rdfs:label ?ndc9 .
    }
}
```

2.5 RDF データセットとグラフ

SPARQL クエリは、RDF グラフの集まりである **RDF データセット**に対して実行されます。

RDF データセットに含まれるグラフは、URI による名前で識別します。データセットには名前をもたない**既定グラフ**が常に 1 つだけあり、オプションで**名前付きグラフ**を含めることができます。

基本グラフパターンのマッチは、データセットの中のいずれかのグラフを対象に行ないます。検索対象となるグラフを**アクティブなグラフ**と呼び、GRAPH キーワード（2.5.1）で指定します。対象グラフを指定しない場合は、既定グラフがアクティブなグラフとなります。

Web NDLA のデータセットは表 5 の 2 つの名前付きグラフで構成されています。

表 5: Web NDLA データセットのグラフ

データ内容	グラフ URI
件名典拠	http://id.ndl.go.jp/auth/ndlsh
名称典拠	http://id.ndl.go.jp/auth/ndlna

また、この 2 つのグラフを合わせたもの（典拠全体）を既定グラフとして扱います。

2.5.1 GRAPH キーワード

WHERE 句の中で GRAPH に続けて URI を指定すると、アクティブなグラフを変更することができます。このアクティブなグラフに対して検索するパターンを、グループグラフパターンとして{}に囲んで続けます。

```
SELECT * WHERE {
    GRAPH <http://id.ndl.go.jp/auth/ndlna> {
        ?s rdfs:label "インターネット"
    }
}
```

上の例では、名称典拠にグラフを限定して「インターネット」を検索しています（団体名としての「インターネット」がマッチします）。

GRAPH キーワードに続く URI を変数にすると、グループグラフパターンがマッチするグラフ名を取得することができます。

```
SELECT * WHERE {
    GRAPH ?g {
        ?s rdfs:label "インターネット"
    }
}
```

このクエリから、件名典拠、名称典拠それぞれに「インターネット」があることが分かります。

表 6: グラフ名を変数にした「インターネット」検索結果

g	s
http://id.ndl.go.jp/auth/ndlsh	http://id.ndl.go.jp/auth/ndlsh/00841024
http://id.ndl.go.jp/auth/ndlna	http://id.ndl.go.jp/auth/ndlna/001144835

GRAPH キーワードを複数用いて、複数のグラフに対してそれぞれ検索を行なうことができます。また同じ変数を複数のグラフにまたがって用いることもできます。

次の例では、件名典拠、名称典拠のグラフに対して同じ変数?label を用いることで、両者で同じラベルが用いられている標目を調べています。

```
SELECT * WHERE {
    GRAPH <http://id.ndl.go.jp/auth/ndlsh> {
        ?sh rdfs:label ?label
    }
    GRAPH <http://id.ndl.go.jp/auth/ndlna> {
        ?na rdfs:label ?label
    }
}
```

2.5.2 FROM 句

WHERE 句の前に FROM <グラフ URI>を置いて、既定グラフを設定することができます。また FROM NAMED <グラフ URI>とすると、そのグラフを名前付きグラフとしてデータセットに組み込みます。これらはそれぞれ複数記述する（複数のグラフを対象に検索する）ことができます。

FROM 句は、Web NDLA においては対象グラフを絞り込む働きをします[13]。次の例は名称典拠を対象として「インターネット」を検索することになり、GRAPH キーワードを用いた時と同じ結果が得られます。

[13]サービスによっては、FROM 句を用いて外部のグラフを取り込んでデータセットに加えられるものもありますが、Web NDLA では外部データの取り込みは行ないません。また Web NDLA では、FROM でも FROM NAMED でも、いずれもグラフは既定グラフと名前付きグラフの両方に加えられます。

```
SELECT *
FROM <http://id.ndl.go.jp/auth/ndlna>
WHERE {
    ?s rdfs:label "インターネット"
}
```

2.6 フィルタによる絞り込み

グラフパターンにマッチする結果を、FILTER によって絞り込むことができます。FILTER キーワードに続けて変数を含む式を与え、その式を評価した結果が偽になるときは、マッチ（変数の組合せ）全体を最終結果から取り除きます。

```
?book ex:price ?price .
FILTER (?price < 2000)
```

この例では、グラフパターンにマッチする変数の組合せのうち、?price の値が 2000 以上であるものが取り除かれ、それ以外の組合せが結果として返されます。

2.6.1 比較と演算

一般的なプログラミング言語と同様に、等しい、あるいは大きいなどの比較を表 7 の、また論理的な関係を表 8 の演算子によって行ないます。式は () で囲んで記述します。

表 7: SPARQL の比較演算子

演算子	意味（真である条件）
`A = B`	A と B が等しい
`A != B`	A と B が等しくない
`A > B`	A は B より大きい
`A < B`	A は B より小さい
`A >= B`	A は B 以上
`A <= B`	A は B 以下

表 8: SPARQL の論理演算子

演算子	意味（真である条件）
`A || B`	A、B のいずれかが真（OR）
`A && B`	A、B のいずれも真（AND）
`! A`	A ではない（NOT）

比較の評価は、A、B のデータ型に応じて行ないます。A が 10、B が 5 であるとき、`A > B` の評価は、これらが数値型であれば真になりますが、文字列型である場合は偽になります。A と B は同じ型

でなければなりません。Web NDLA ではリテラル値はすべてデータ型をもたない文字列ですが、変数値、比較対象ともに数値に変換可能な場合、自動的に数値変換して比較します[14]。

値が数値型であれば、四則演算を用いることもできます。次の例では最大値と最小値の差が 10 以上である場合に限定した絞り込みを行なっています。

```
?what ex:height ?max ;
    ex:low ?min .
FILTER (?max - 10 >= ?min)
```

2.6.2 テスト演算子

値がリテラルである、空白ノードであるなどを評価するために表 9 の演算子が用意されています。

表 9: SPARQL のテスト演算子

演算子	意味（真である条件）
bound(A)	変数 A にマッチする値がある
isIRI(A)	A は IRI である
isURI(A)	A は URI である
isBLANK(A)	A は空白ノードである
isLITERAL(A)	A はリテラルである

OPTIONAL パターンと bound() による絞込の否定を組み合わせると、特定のプロパティを「持たない」グラフを検索することができます。例えば次の例では、生年を持つが没年を持たない人物（存命中の人物）を調べることができます。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rda: <http://RDVocab.info/ElementsGr2/>
SELECT * WHERE {
    ?who a foaf:Person; foaf:name ?name ;
        rda:dateOfBirth ?bdate .
    OPTIONAL {
        ?who rda:dateOfDeath ?ddate.
    }
    FILTER (!bound(?ddate))
}
```

没年を OPTIONAL としないと、グラフパターンとしては?ddate が必須であり、かつ FILTER で?ddate にマッチするものを除いてしまうため、結果が空になることに注意してください[15]。

[14] 一般には、データ型が異なる場合は、2.6.3 の STR() 演算子を用いて単純文字列にそろえるなどしてから比較します。

[15] SPARQL 1.1 では、FILTER NOT EXISTS ?who rda:dateOfDeath ?ddate とすることで否定の絞り込みを直接行なうことができます。

2.6.3 値取得演算子

真偽値以外の値を得るために、表 10 の演算子が用意されています。

表 10: SPARQL の値取得演算子

演算子	値
`str(A)`	A を単純文字列に変換した値
`lang(A)`	リテラル A の言語タグの値
`datatype(A)`	リテラル A のデータ型 URI

例えば、あるグラフパターンにマッチする標目のローマ字読みを取り出したいときは、次のように `lang()` を用いて絞り込みます。

```
PREFIX xl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT * WHERE {
    ?uri xl:prefLabel [ ndl:transcription ?yomi ] ;
    #他のグラフパターン
    FILTER (lang(?yomi) = "ja-Latn")
}
```

2.6.4 正規表現

文字列に対しては、正規表現演算子 `regex()` を用いて絞り込みを行なうことができます。文字列の部分一致も正規表現演算子によって調べます。

Web NDLA では表 11 の記法を使うことができます。[16]。

正規表現演算子は、`regex(対象, 正規表現)` の形で用います。文字列の部分一致の場合は、正規表現に検索したい部分文字列を直接記述します。

たとえばラベルに「図書館」を含む標目を検索する場合は次のような絞り込みを行ないます。

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER regex(?label, "図書館")
}
```

次のようにすると、アルファベット大文字 3 字の標目を調べることができます。

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER regex(?label, "^[A-Z]{3}$")
```

[16]SPARQL の正規表現は、XQuery/XPath の正規表現構文に従うことになっていますが、現在 Web NDLA でバックスラッシュによるエスケープ、メタ文字はサポートしていません。

表 11: Web NDLA の正規表現

文字	機能
.	任意の 1 文字にマッチ
*	直前のパターンの 0 回以上の繰り返し
+	直前のパターンの 1 回以上の繰り返し
?	直前のパターンがオプション（0 回もしくは 1 回）
^	文字列の先頭にマッチ
$	文字列の末尾にマッチ
()	パターンのグループ化
|	グループ内でのパターンの選択（OR）
{}	直前のパターンの繰り返し数指定
[...]	文字クラス
[^...]	文字クラスの否定

```
}
```

正規表現の 3 番目の引数として、マッチ方法を制御するフラグを与えることができます。現在 Web NDLA では、大小文字を区別しない"i"フラグをサポートしています。次の例は"internet"、"Internet"、"INTERNET"のいずれにもマッチするクエリです。

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER regex(?label, "internet", "i")
}
```

FILTER の条件式は論理演算子で連結することができます。次のようにすると、「インターネット」もしくは「Internet」を含む標目が得られます。

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER ( regex(?label, "インターネット") || regex(?label, "Internet") )
}
```

なお論理積&&による結合は、FILTER 句を 2 つ書くことと同等です。

2.7 結果セットの制御

結果セットの取得数を制約したり、結果を並べ替えたりすることができます。

2.7.1 LIMIT と OFFSET

WHERE 句に続けて LIMIT 句を置くと、取得する結果の最大数を設定できます。また OFFSET 句を置くと、結果セットの指定値以降の結果を得ることができます。

次の例は標目の最初の 10 件を取得します。

```
SELECT * WHERE {
    ?uri rdfs:label ?label
} LIMIT 10
```

次の例は、標目の 6 番目から 10 件を取得します（OFFSET は先頭からスキップする数を示します）。LIMIT と OFFSET はどちらを先に記述しても構いません。

```
SELECT * WHERE {
    ?uri rdfs:label ?label
} LIMIT 10 OFFSET 5
```

現在、Web NDLA は取得結果の上限を 100 としています。このため、LIMIT を 100 以上にしても 1 回のクエリで最大 100 件しか結果を取得できません。101 件目以降の結果を取り出すためには、OFFSET 句を用いてください。

```
SELECT * WHERE {
    ?uri rdfs:label ?label
} OFFSET 100
```

2.7.2 並べ替え

結果を並べ替えるためには、WHERE 句のあとに、ORDER BY に続けて並べ替えキーとする変数を空白区切りで列挙します。変数名を DESC() で囲むと降順、ASC() で囲むと昇順です。変数名のみを書くと昇順として扱われます。

```
SELECT * WHERE {
    ?uri rdfs:label ?label ;
        dct:modified ?moddate .
} ORDER BY ?moddate
```

LIMIT、OFFSET と組合せ、並べ替えた上で必要な部分だけを取り出すこともできます。この場合、ORDER BY を先に記述し、そのあとに LIMIT、OFFSET を続けます。

2.7.3 重複値の制御

グラフパターンによっては、変数に同じ値がマッチする結果が複数みつかる場合があります。このとき、SELECT に続けて DISTINCT を指定すると、重複を取り除いた結果を得ることができます。

```
SELECT DISTINCT ?type WHERE {
    ?s a ?type .
}
```

DISTINCT の代わりに REDUCED を指定すると「重複を省くものの完全な重複チェックはしない」という形での絞り込みを行ないます。どちらでもほとんどの場合で同じ結果となりますが、大規模な結果セットに対する完全な重複チェックは負荷が高いため、REDUCED の方が速く答えが得られます。

2.7.4 集計

結果の数え上げやグループ分けといった集計機能は SPARQL 1.1 で導入されたものですが、Web NDLA（ARC2 ライブラリ）でもその一部を利用できます。

結果の数をカウントするためには、SELECT での変数選択時に、COUNT() 関数で結果数を数え上げ、AS によって別の変数に割り当てます。たとえば Web NDLA に 1960 年生まれの人物の典拠が何件あるかを調べるためには次のクエリを用います。

```
SELECT (COUNT(?who) AS ?howmany) WHERE {
    ?who rda:dateOfBirth "1960" .
}
```

WHERE 句に続けて GROUP BY で変数を指定すると、結果を変数の値によってグループ化できます。これを用いて、グループごとの集計を行うことが可能です。例えば、1900 年以降に生まれた人物の典拠が年ごとに何件あるかを調べるためには、次のようにします。

```
SELECT ?byear (COUNT(?who) AS ?howmany) WHERE {
    ?who rda:dateOfBirth ?byear .
    FILTER (?byear >= 1900)
} GROUP BY ?byear
ORDER BY ?byear
```

集計に用いることができる関数は COUNT() の他に、MAX()、MIN()、さらに変数が数値ならば AVG()、SUM() があります。例えば次のクエリでは、人物典拠の生年が最も古いものと最も新しいものが調べられます。

```
SELECT (MIN(?byear) AS ?past) (MAX(?byear) AS ?recent) WHERE {
    ?who rdf:dateOfBirth ?byear .
}
```

2.8 クエリの形態と結果

SPARQL クエリには SELECT、ASK、CONSTRUCT、DESCRIBE の 4 つの形態があります。

2.8.1 SELECT

SELECT クエリは、グラフパターンにマッチした変数のセットを取得します。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?subj2 ?label
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

上の SELECT クエリを用いて Web NDLA に照会を行なうと、XML 結果フォーマットの場合、次のような結果が得られます。

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="subj2"/>
    <variable name="label"/>
  </head>
  <results>
    <result>
      <binding name="subj2">
        <uri>http://id.ndl.go.jp/auth/ndlsh/00969901</uri>
      </binding>
      <binding name="label">
        <literal>バーチャルプライベートネットワーク</literal>
      </binding>
    </result>
    <result>
      <binding name="subj2">
        <uri>http://id.ndl.go.jp/auth/ndlsh/00865280</uri>
      </binding>
      <binding name="label">
        <literal>イントラネット</literal>
      </binding>
    </result>
    <result>
      <binding name="subj2">
        <uri>http://id.ndl.go.jp/auth/ndlsh/01017771</uri>
      </binding>
      <binding name="label">
```

```
        <literal>セマンティックウェブ</literal>
      </binding>
    </result>
  </results>
</sparql>
```

<head>要素には結果セットに含まれる変数名が列挙され、<results>要素内にはマッチした変数の組がそれぞれ<result>要素として列挙されます。結果フォーマットの詳細は 3.1 を参照してください。

2.8.2 ASK

パターンに合致する部分グラフがあるかどうかを調べる（値は取得せず、合致するかどうかの真偽値を得る）場合は、ASK クエリを用います。変数値は取得しないので、ASK キーワードに直接 WHERE 句の内容を続けます。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
ASK WHERE {
    ?subj1 rdfs:label "インターネット"
}
```

上の ASK クエリを用いて Web NDLA に照会を行なうと、XML 結果フォーマットの場合、次のような結果が得られます。

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head></head>
  <boolean>true</boolean>
</sparql>
```

合致する部分グラフがない場合は、<boolean>要素の値として false が返されます。

2.8.3 CONSTRUCT

グラフパターンに合致する変数値を用いて別の RDF グラフを構築します。新しいグラフのパターンを CONSTRUCT 句に記述し、条件となる WHERE 句をそのあとに続けます。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX ex: <http://example.org/terms#>
CONSTRUCT {
    ?subj1 ex:下位語 ?label .
}
WHERE {
```

```
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

上の CONSTRUCT クエリを用いて Web NDLA に照会を行なうと、Turtle 結果フォーマットの場合、次のような結果が得られます。

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns0: <http://example.org/terms#> .

<http://id.ndl.go.jp/auth/ndlsh/00841024>
    ns0:下位語 "バーチャルプライベートネットワーク" ,
        "イントラネット" ,
        "セマンティックウェブ" .
```

CONSTRUCT クエリによって、元のグラフを異なるモデルのグラフに変換することが可能です。なお、この例のように結果の接頭辞はクエリで指定したものと異なる場合があります。結果フォーマットを XML とすると、RDF/XML 形式で結果が得られます。

2.8.4 DESCRIBE

DESCRIBE クエリは、リソースに関する説明を取得します。Web NDLA では、そのリソース URI を主語にするトリプル、および空白ノードを介してつながっている構造化記述のトリプルからなる「説明グラフ」が得られます。

対象リソースは直接 URI で指定できます。次の DESCRIBE クエリを用いて Web NDLA に照会を行なうと、URI に拡張子.ttl もしくは.rdf を加えた場合と同じ RDF が得られます。

```
DESCRIBE <http://id.ndl.go.jp/auth/ndlsh/00841024>
```

また、次のように WHERE 句を用いてグラフパターンの変数にマッチする URI を取り出し、そのリソースに関する RDF グラフを得ることもできます。複数の URI がマッチする場合、また取得対象として複数の変数を列挙した場合は、それらの URI に関する「説明グラフ」を併合したグラフが得られます。

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
DESCRIBE ?subj2
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
```

```
}
```

変数値がリテラルである場合は、「説明グラフ」は空になります。

3　APIパラメータと結果フォーマット

APIのリクエストは、1.1.2に示した通り、クエリをURLエンコードした文字列をqueryパラメータ値とし、結果フォーマットをoutputパラメータ値として、次のURLに送ります。

```
http://id.ndl.go.jp/auth/ndla
```

SELECTクエリの結果の変数バインディングリスト、およびASKクエリの結果の真偽値は、outputパラメータの値に応じてXMLもしくはJSONの結果フォーマットで返されます。

3.1　XML結果フォーマット

outputパラメータ値をxmlとした場合は、SPARQL Query Results XML Format[17]に従って結果を返します。

このフォーマットは、2.8.1の例に示した通り名前空間http://www.w3.org/2005/sparql-results#のsparql要素内にheadとresults要素（ASKクエリの場合はboolean要素）を持つXMLです。

head要素内には、クエリ結果の全ての変数が、クエリでの記述順にvariable要素として並びます。name属性値が変数名を示します。ASKクエリの場合はhead要素は空になります。

```
<head>
  <variable name="subj2"/>
  <variable name="label"/>
</head>
```

results要素内には、結果の変数セットがそれぞれresult要素として列挙されます。result要素内には値が結び付けられた変数ごとにbinding要素が置かれ、name属性値で変数名を示します。値がURI、リテラル、空白ノードのいずれかに応じて<uri>、<literal>、<bnode>を子要素にし、その内容として変数値が示されます。

リテラル値が言語タグを持つ場合は、<literal>要素にxml:lang属性が加わります。データ型がある場合はdatatype属性にデータ型URIが示されます。

```
<results>
  <result>
    <binding name="subj2">
      <uri>http://id.ndl.go.jp/auth/ndlsh/00969901</uri>
    </binding>
    <binding name="label">
      <literal>バーチャルプライベートネットワーク</literal>
    </binding>
  </result>
  ...
</results>
```

[17] http://www.w3.org/TR/rdf-sparql-XMLres/

ASK クエリの場合は、2.8.2 の例のように、results 要素の代わりに<boolean>要素が置かれ、真（true）もしくは偽（false）がその内容として返されます。

3.2 JSON 結果フォーマット

output パラメータ値を json とした場合は、SPARQL 1.1 Query Results JSON Format[18]に従って結果を返します。

このフォーマットは、最上位オブジェクト内に head および results（ASK クエリの場合は boolean）プロパティ値としてそれぞれオブジェクトを持つ JSON です。2.8.1 のクエリの結果を JSON フォーマットで受け取ると、次のようなものとなります。

```
{
  "head": {
    "vars": [
      "subj2",
      "label"
    ]
  },
  "results": {
    "bindings": [
      {
        "subj2": {
          "type": "uri",
          "value": "http://id.ndl.go.jp/auth/ndlsh/00969901"
        },
        "label": {
          "type": "literal",
          "value": "\u30d0\u30fc\u30c1\u30e3\u30eb\u30d7\u30e9..."
        }
      },
      ....
    ]
  }
}
```

head プロパティ値のオブジェクトは vars プロパティを持ち、変数名の配列がその値となります。

results プロパティ値のオブジェクトは bindings プロパティを持ち、結果の変数セットオブジェクトの配列がその値となります。各変数セットオブジェクトは、値が結び付けられた変数名をプロパティとし、その値オブジェクトに URI、空白ノード、リテラルを示す type プロパティと、値を示す value プロパティを持つ、という構成になっています。

なお、上例のように value プロパティの値は、ASCII 以外の文字が\u + Unicode 番号の形にエスケープされます。

[18]http://www.w3.org/TR/sparql11-results-json/

4　典拠レコードの RDF グラフと総合的な検索例

Web NDLA 典拠レコードのモデル（グラフ構造）を説明し、そのグラフに対して、ここまでで説明した SPARQL の要素を組合せて検索を行なう例を紹介します。

4.1　個人名、家族名、団体名の典拠

名称典拠の中でも個人名、家族名、団体名は、典拠レコードを表すリソースと、人物や団体などの実体に対応するリソースを区別しています。

典拠レコードは標目、別名、出典、関連リンクなどのプロパティを持ちます。標目、別名は読みとセットにするために空白ノードを介して構造化しています。

実体リソースは人物の生没年や団体の設立年、来歴などを記述しています。典拠レコードと実体リソースは `foaf:primaryTopic` によって結び付けられています（図 4）。

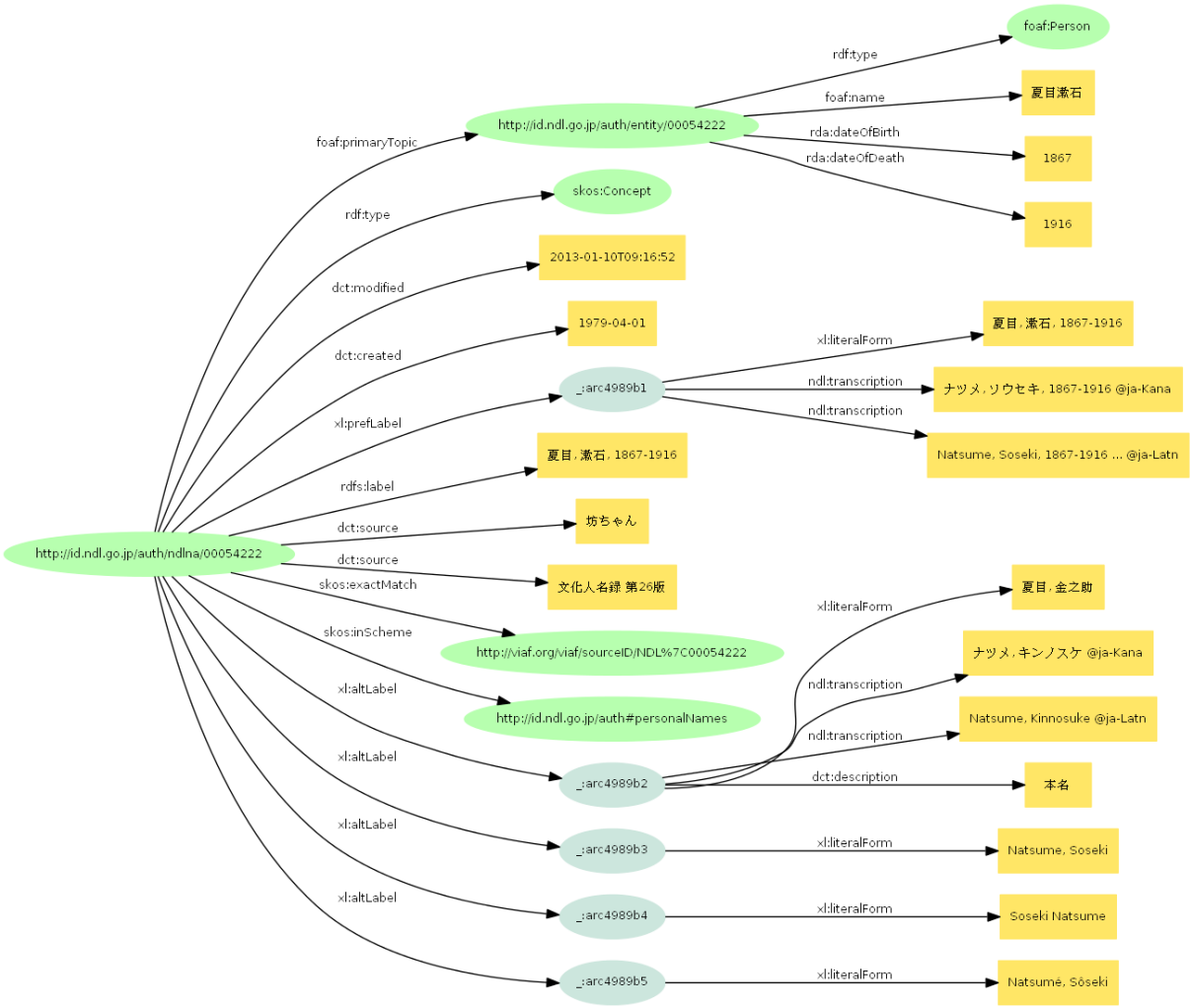

図 4:「夏目漱石」の名称典拠グラフ。図左の.../ndlna/00054222 が典拠レコード、図上の.../entity/00054222 が実体リソースを表す。

4.1.1 ある人名の生没年、標目形、標目形カナヨミを調べる

人名が分かる場合は、その値を foaf:name の目的語とし、それ以外の知りたい値を変数として SELECT クエリを組み立てます。

```
PREFIX rda: <http://RDVocab.info/ElementsGr2/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT * WHERE {
    ?auth
        foaf:primaryTopic ?entity ;
        xl:prefLabel [
            xl:literalForm ?preflabel ;
            ndl:transcription ?yomi ] .
    ?entity
        rda:dateOfBirth ?birth ;
        rda:dateOfDeath ?death ;
        foaf:name "夏目漱石".
    FILTER (lang(?yomi) = "ja-Kana")
}
```

標目や rdfs:label の値は、典拠のラベルとして「夏目, 漱石, 1867-1916」のようになっているため、姓名を連結した人名での検索には実体に与えた foaf:name を用いることに注意してください。

標目の読み（ndl:transcription）はカナとローマ字の 2 つがあるため、FILTER で言語タグを指定します。

4.1.2 11 世紀生まれの人物とその VIAF を調べ、生年順に並べる

「1001〜1100 年に生まれた」という条件を、FILTER の論理演算を用いて表現します[19]。VIAF には典拠レコードから skos:exactMatch でリンクしています。変数?birth は、FILTER による絞込に加え、並べ替えにも用います。

```
PREFIX rda: <http://RDVocab.info/ElementsGr2/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT * WHERE {
    ?auth
        foaf:primaryTopic [
            rda:dateOfBirth ?birth ;
            foaf:name ?name ] ;
```

[19] rda:dateOfBirth の値は文字列（単純リテラル）であるため、FILTER での数値比較を行なう場合、xsd:integer などの数値型に変換する必要があります。Web NDLA の SPARQL 1.0 エンドポイントではこの変換なしでも検索可能ですが、SPARQL 1.1 エンドポイントでは変換なしでは結果が得られません。

```
        skos:exactMatch ?viaf .
    FILTER (xsd:integer(?birth) >= 1001 && xsd:integer(?birth) <= 1100)
} ORDER BY ?birth
```

検索結果は最大 100 件までしか取得できないので、100 件を超える場合は 2.7.1 の OFFSET を用いて繰り返し検索してください。

4.2 地名、統一タイトル、普通件名、細目の典拠

これらは前節のようなリソースの使い分けはなく、典拠リソースを主語にしたフラットな構造です（標目、同義語は読みをセットにするため、名称典拠の場合と同様に構造化しています）。件名では上位、下位、関連語や分類を示すため、名称典拠よりもプロパティは増えています（図 5）。

4.2.1 ある件名典拠の上位語、下位語などを調べる

件名典拠の場合は","の有無などを気にする必要はないので、シンプルな rdfs:label の目的語に件名を与え上位語、下位語などを変数とします。次は「インターネット」の上位語を調べる例です。

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj
        rdfs:label "インターネット" ;
        skos:broader ?broader .
    ?broader rdfs:label ?label .
}
```

4.2.2 ある代表分類に属する件名典拠を調べる

NDLC の「DM225」は URI として表現されます。この値を skos:relatedMatch の目的語としてクエリを組み立てます。

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj
        skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DM225> ;
        rdfs:label ?label .
}
```

NDLC の「DK341」かつ NDC9 の「694.5」に属する件名を調べる場合は、グラフパターンが skos:relatedMatch を 2 つ持つようにします（NDC9 も URI として表現されます）。

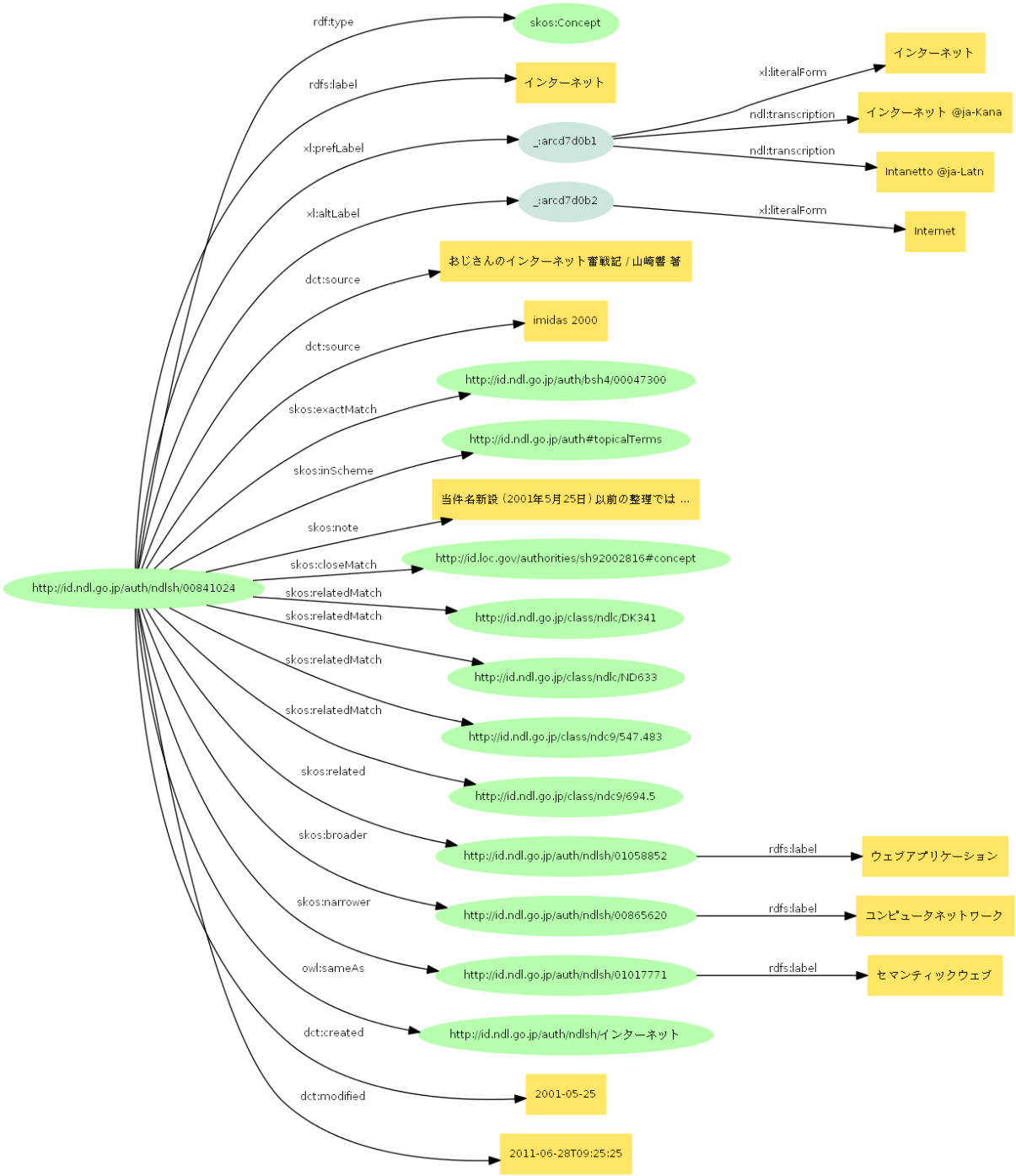

図 5: 「インターネット」の件名典拠グラフ。下位、関連語の一部を省略

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj
        skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DK341> ,
            <http://id.ndl.go.jp/class/ndc9/694.5> ;
        rdfs:label ?label .
}
```

5 SPARQL 1.1

2017 年度に Virtuoso を用いた SPARQL 1.1 エンドポイントを試験公開しました。従来のエンドポイントもそのまま利用できます。

5.1 リクエスト URI とパラメータ

新エンドポイントの URI は次のとおりです。

```
http://id.ndl.go.jp/auth/ndla/sparql
```

対応するパラメータは表 12 の 2 つです。

表 12: Web NDLA API のパラメータ

パラメータ	値
query	SPARQL 1.1 照会言語によるクエリを URL エンコードしたもの
format	結果フォーマット（`xml` | `json` | `turtle` | `csv` | `html`）

- `SELECT` および `ASK` クエリの場合は、`xml`、`json`、`csv` はそれぞれの SPARQL Query Results Formats に基づいた結果が返されます。`turtle` を指定すると、Virtuoso 独自の RDF による結果が返されます。
- `DESCRIBE`、`CONSTRUCT` クエリの場合は、`xml` は RDF/XML、`json` は JSON-LD の Expanded Form[20]、`csv` は各トリプルが 1 行となった CSV として返されます。`turtle` は通常の Turtle によるグラフです。

1 回のクエリで得られる結果の最大数は 1000 件です。

5.2 SPARQL 1.1 の機能とクエリ例

SPARQL 1.1 を用いることで可能になる主な機能を紹介します。詳細は SPARQL 1.1 Query Language[21]（以下 SPARQL1.1 仕様書）をご覧ください。

5.2.1 否定表現

フィルタ表現に `FILTER NOT EXISTS` を用いることで、結果セットからそのフィルタに該当するものを除いた結果を得ることができます（SPARQL1.1 仕様書 § 8）。
次の例は、読みを持たない標目を検索するクエリです。

[20] コンテクストを用いず、プロパティも含む URI がすべて展開された形
[21] http://www.w3.org/TR/sparql11-query/

```
PREFIX xl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT * WHERE {
    ?id xl:prefLabel ?xl .
    ?xl xl:literalForm ?label .
    FILTER NOT EXISTS {?xl ndl:transcription ?yomi }
} LIMIT 10
```

5.2.2 プロパティパス

トリプルパターンのプロパティ部分に、SPARQL1.1 仕様書§9 に示されるパス構文を用いることで、グラフ構造を短縮表現したクエリを記述できます。

次の例は、*によるプロパティの反復を用いて、下位標目をまとめて辿るクエリです。

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?sub ?label WHERE {
    ?top rdfs:label "図書館" ;
        skos:narrower* ?sub .
    ?sub rdfs:label ?label .
}
```

5.2.3 変数割り当て

キーワード BIND を用いて、グラフ中に存在しない値を変数に割り当てることができます（SPARQL1.1 仕様書§10）。

次の例は、各下位標目の NDC を調べ、その URI から分類記号を取り出して変数に割り当て、それを用いて順に並べるクエリです。

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?sub ?label ?ndc WHERE {
    ?top rdfs:label "図書館" ;
        skos:narrower* ?sub .
    ?sub rdfs:label ?label ;
        skos:relatedMatch ?rel.
    FILTER(regex(?rel, "^http://id.ndl.go.jp/class/ndc9/"))
    BIND (strafter(str(?rel), "ndc9/") as ?ndc)
} ORDER BY ?ndc
```

5.2.4 集計

結果の数え上げやグループ分けを行なう集計機能（SPARQL1.1 仕様書§11）は、ARC2 のエンドポイントでも部分導入されています（§2.7.4 参照）が、前述の変数割り当てと組み合わせることでより柔軟なクエリを記述できます。

次の例は、典拠を NDC の大分類（類目）ごとに数え、類目と件数を一覧するクエリです。

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?ndc (count(?id) as ?count) WHERE {
    ?id skos:relatedMatch ?rel.
    FILTER(regex(?rel, "^http://id.ndl.go.jp/class/ndc9/"))
    BIND (substr(str(?rel), 32, 1) as ?ndc)
} GROUP BY ?ndc ORDER BY ?ndc
```

5.2.5 サブクエリ

クエリ内部に別のクエリを記述して組み合わせることで、一つのクエリの結果を利用したクエリを一度に照会できます（SPARQL1.1 仕様書§12）。

次の例は、「図書館」の下位標目を調べた上で、さらにそれらがいくつの下位標目（「図書館の孫標目）を持つかを数え、ラベルとともに表示するクエリです。

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?sub ?label ?count WHERE {
    ?top rdfs:label "図書館" ;
        skos:narrower ?sub .
    ?sub rdfs:label ?label .
    {
        SELECT ?sub (count(?ssub) as ?count) WHERE{
            ?sub skos:narrower ?ssub .
        }
    }
}
```

5.3 従来の SPARQL エンドポイント（1.0）との違い

Virtuoso の制約により、読みに付与している言語タグが全て小文字になってしまっています。検索時にはシステムで違いを吸収するようにしていますが、検索結果に言語タグが含まれる場合は従来と違ったものになるので注意してください。

例えば次のクエリを実行した場合

```
PREFIX xl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT ?yomi WHERE {
    ?id xl:prefLabel [
        xl:literalForm "蔬菜" ;
        ndl:transcription ?yomi]
} LIMIT 1
```

JSON 結果フォーマットの bindings の値は従来のエンドポイントだと

```
{
  "yomi": {
    "type": "literal",
    "value": "Sosai",
    "xml:lang": "ja-Latn"
  }
}
```

であるのに対し、SPARQL 1.1 エンドポイントからの結果だと

```
{
  "yomi": {
    "type": "literal",
    "value": "Sosai",
    "xml:lang": "ja-latn"
  }
}
```

となります（xml:lang の値を比べてみてください）。

6　改訂履歴

- 2018-03-31：SPARQL 1.1 エンドポイントの試行版導入に伴い、説明のために§5 を追加。また、SPARQL 1.1 機能について従来非対応としていた説明を修正。
- 2023-03-31：テスト演算子（2.6.2）のクエリ例誤り修正。総合検索例で文字列と数値の比較に関する注記を追加（4.1.2）。