初級 OWL 講座

第零版 revision01

大黒学

初級 OWL 講座・第零版 revision01
著者——大黒学

2008 年　2 月 10 日（日）　第零版 revision01 発行

Copyright © 2006–2008 Daikoku Manabu
This tutorial is licensed under a Creative Commons Attribution 2.1 Japan License.

目次

第 1 章 RDF

1.1 RDF の基礎

Q 1.1.1 リソースって何ですか。

A 「リソース」(resource) というのは、ウェブ上で識別することが可能なもののことです。

「ウェブ上で識別することが可能なもの」というのは、データだけに限定されません。会社、人間、建築物のような実体を持つものや、作者、題名、価格、というような概念もまた、「ウェブ上で識別することが可能なもの」の中に含まれます。

Q 1.1.2 RDFって何ですか。

A RDF(Resource Description Framework) というのは、ウェブ上にあるリソースについての情報を表現するための言語の枠組みです。

RDF は、1999 年に W3C の勧告となって、そののち 2004 年に改訂されて現在に至っています。
RDF を土台として作られた言語としては、RSS(RDF Site Summary)[1]、FOAF(Friend Of A Friend)、OWL(Web Ontology Language) などがあります。

Q 1.1.3 RDF 文って何ですか。

A 「RDF 文」(RDF statement) というのは、何らかのリソースに関する「何々の何々は何々である」という形の言明をあらわしている記述のことです。

RDF 文は、RDF に関する文脈では普通、単に「文」(statement) と呼ばれます。
文があらわしているのは、たとえば、「このウェブページの作者は高岡菜々子である」とか、「この商品の価格は 500 円である」というような、「何々の何々は何々である」という形の言明です。
ひとつの文は、かならず三つのリソースをあらわす記述から構成されます。それらのリソースは、次のように呼ばれます。

何々の	主語 (subject)
何々は	述語 (predicate)
何々である	目的語 (object)

「何々は」と「何々である」に関しては、次のように呼ばれることもあります。

何々は	プロパティー (property)
何々である	値 (value)

Q 1.1.4 リソースって、どんなふうに記述すればいいんですか。

A リソースは、URI(Uniform Resource Identifiers) を使うことによって記述することができます。

たとえば、「このウェブページ」、「作者」、「高岡菜々子」というリソースは、URI を使って次のように記述することができます。

このウェブページ	`http://www.example.org/nanako/`
作者	`http://purl.org/dc/elements/1.1/creator`
高岡菜々子	`http://example.org/staff/nanako`

ちなみに、「このウェブページ」と「高岡奈々子」は架空のリソースですが、

```
http://purl.org/dc/elements/1.1/creator
```

は、「ダブリンコア」(Dublin Core) と呼ばれる実在するリソースの集合に含まれているリソースのひとつで、「作者」を意味しています。ダブリンコアについては、第 2.5 節で、もう少し詳しく説明したいと思います。
なお、リソースを一意的に識別するための名前として使われている URI は、それが指示している場所へアクセスしたとしても、かならずしもそこにデータが存在しているとは限りません。

[1] ただし、RSS には、RDF に基づいているバージョンと基づいていないバージョンがあります。

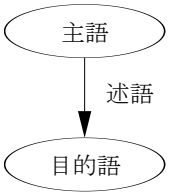
Q 1.1.5 RDF 三つ組みって何ですか。

A 「RDF 三つ組み」(RDF triple) というのは、主語、述語、目的語を、この順序で並べることによって作られた列のことです。

RDF 三つ組みは、RDF に関する文脈では普通、単に「三つ組み」(triple) と呼ばれます。
三つ組みというのは、文が記述しているそれぞれのリソースを並べることによってできる、

(主語, 述語, 目的語)

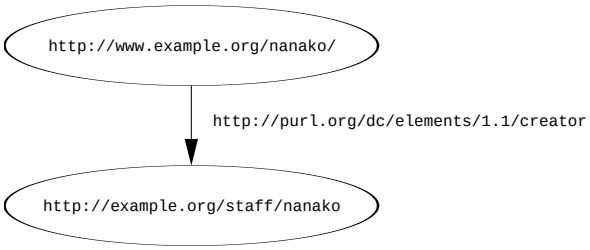
という列のことです。
三つ組みは、次のような図によって表現することができます。



このように、図では、主語と目的語のそれぞれは楕円であらわされて、述語は、主語から目的語へ向かう矢印であらわされます。たとえば、

主語 `http://www.example.org/nanako/`
述語 `http://purl.org/dc/elements/1.1/creator`
目的語 `http://example.org/staff/nanako`

という三つ組みは、



という図によって表現されます。

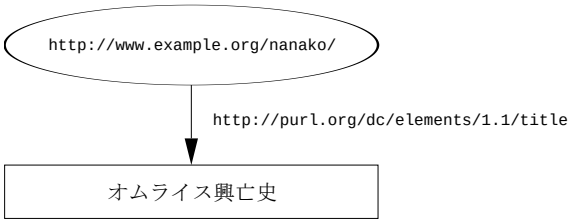
Q 1.1.6 リテラルって何ですか。

A 「リテラル」(literal) というのは、それ自体がリソースである文字列のことです。

RDF では、文の主語と述語については、かならず URI を使って記述しなければならないのですが、目的語については、リテラルを使って記述することも可能です。たとえば、ウェブページの題名や人間の氏名は、URI ではなくリテラルを使って記述することができます。
Q 1.1.5 で、三つ組みを図で表現する場合、主語と目的語は楕円であらわすと書きましたが、厳密に言えば、目的語が楕円によってあらわされるのは、それが URI で記述される場合だけです。リテラルで記述される目的語は、楕円ではなく長方形を使ってあらわします。たとえば、

このウェブページの題名は「オムライス興亡史」である。

という文によって記述される三つ組みは、



という図によって表現されます。

Q 1.1.7 RDF グラフって何ですか。

A 「RDF グラフ」(RDF graph) というのは、三つ組みの集合のことです。

RDF グラフは、RDF に関する文脈では普通、単に「グラフ」(graph) と呼ばれます。
グラフを構成しているそれぞれの主語または目的語は「ノード」(node) と呼ばれます。また、述語によって示される主語と目的語との関係は、「アーク」(arc) と呼ばれます。

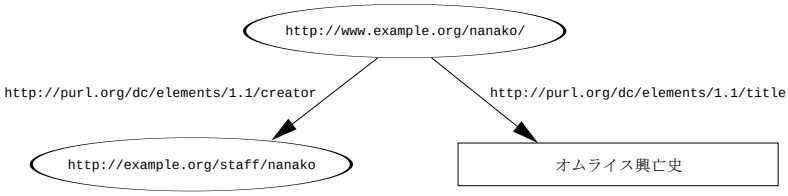
Q 1.1.8 グラフを図で表現することって、可能ですか。

A はい、可能です。

グラフは、基本的には、そのグラフを構成しているそれぞれの三つ組みを、楕円、長方形、矢印を使ってあらわすことによって、図示することができます。
ひとつのグラフの中に、同一の URI で識別されるノードを持つ複数の三つ組みが含まれている場合、それらのノードは、ひとつの楕円を共用してもかまいません。たとえば、

> このウェブページの作者は高岡菜々子である。
> このウェブページの題名は「オムライス興亡史」である。

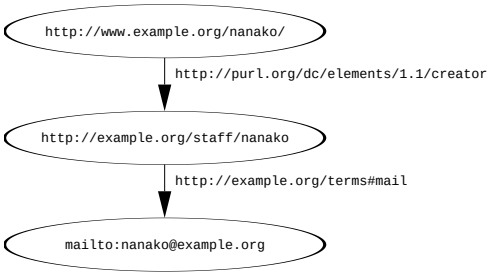
というグラフに含まれている二つの三つ組みは、主語が同一ですので、このグラフは、



という図によって表現することができます。同じように、

> このウェブページの作者は高岡菜々子である。
> 高岡菜々子のメールアドレスは `nanako@example.org` である。

というグラフに含まれている二つの三つ組みは、一方の目的語と他方の主語とが同一ですので、このグラフは、



という図によって表現することができます。

1.2 RDF/XML

Q 1.2.1 RDF/XMLって何ですか。

A RDF/XML というのは、RDF グラフを文字列で記述するための文法のひとつです。

RDF グラフを文字列で記述するための文法としては、RDF/XML だけではなくて、

- N-Triples
- N3(Notation 3)
- Turtle
- RXR(Regular XML RDF)

など、いくつかのものが提案されています。RDF/XML は、それらのうちでもっとも標準的な文法です。

RDF/XML は、その名前が示しているように、XML を使って RDF グラフを記述します。ちなみに、RXR も、XML を使って RDF グラフを記述する文法のひとつです。

Q 1.2.2 RDF/XML 文書って何ですか。

A 「RDF/XML 文書」(RDF/XML document) というのは、RDF/XML で記述された RDF グラフが、ルート要素の子供として書かれている XML 文書のことです。

Q 1.2.3 RDF/XML 文書って、MIME タイプは何ですか。

A RDF/XML 文書の MIME タイプは、`application/rdf+xml` です。

Q 1.2.4 RDF/XML 文書を保存するファイルって、どんな拡張子を付ければいいんですか。

A RDF/XML 文書を保存するファイルには、原則として `.rdf` という拡張子を付けます。

Q 1.2.5 RDF/XML で定義されてる名前って、どんな名前空間に属してるんですか。

A RDF/XML で定義されている名前が属しているのは、

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#
```

という名前空間です。

RDF/XML の名前空間接頭辞としては、通常、`rdf` が使われます。

Q 1.2.6 RDF/XML では、URI で識別されるノードは、どんなふうに記述すればいいんですか。

A RDF/XML では、URI で識別されるノードは、`rdf:Description` という要素型の要素を使って記述します。

`rdf:Description` 要素は、`rdf:about` という属性を持っています。ノードを識別する URI は、この属性に設定します。たとえば、

```
http://www.example.org/nanako/
```

という URI によって識別されるノードを RDF/XML で記述したいときは、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
</rdf:Description>
```

という要素を書きます。

URI によって識別されるノードを RDF/XML で記述している要素は、「ノード要素」(node element) と呼ばれます。

Q 1.2.7 RDF/XML では、述語はどんなふうに記述すればいいんですか。

A RDF/XML では、述語は、それを識別する URI と等価な QName を要素型名とする要素を使って記述します。

RDF/XML では、QName は、名前空間名とローカル部分とを連結してできた URI と等価であるとみなされます。たとえば、

```
xmlns:ex="http://example.org/terms#"
```

という名前空間宣言によって名前空間名と名前空間接頭辞とが結び合わされているとすると、`ex:mail` という QName は、

```
http://example.org/terms#mail
```

という URI と等価だとみなされます。

RDF/XML で述語を記述したい場合は、まず名前空間宣言で名前空間名と名前空間接頭辞とを結び合わせた上で、述語を記述する URI と等価な QName を要素型名とする要素を作ります。たとえば、

```
http://purl.org/dc/elements/1.1/creator
```

という URI で識別される述語を RDF/XML で記述したいときは、まず、

```
xmlns:dc="http://purl.org/dc/elements/1.1/"
```

という名前空間宣言によって名前空間名と名前空間接頭辞とを結び合わせた上で、

```
<dc:creator></dc:creator>
```

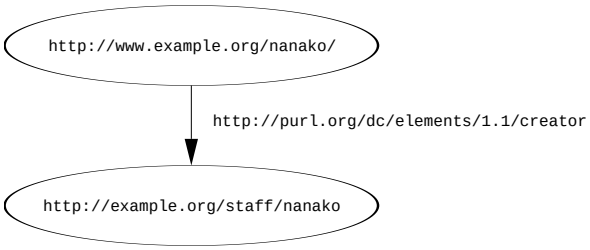
という要素を書きます。

RDF/XML で述語を記述している要素は、「プロパティー要素」(property element) と呼ばれます。

Q 1.2.8 RDF/XML では、三つ組みはどんなふうに記述すればいいんですか。

A RDF/XML で三つ組みを記述したいときは、主語をあらわすノード要素の子供としてプロパティー要素を書いて、その子供として目的語をあらわすノード要素を書きます。

ですから、RDF/XML では、ひとつの三つ組みは三重の入れ子になった要素によって記述されるということになります。たとえば、「このウェブページの作者は高岡菜々子である」という文によって記述される三つ組み、すなわち、



という図で示される三つ組みは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator>
    <rdf:Description rdf:about="http://example.org/staff/nanako">
    </rdf:Description>
  </dc:creator>
</rdf:Description>
```

という、三重の入れ子になった要素によって記述されます。

Q 1.2.9 RDF/XML 文書のルート要素って、どんな要素型を使って書けばいいんですか。

A RDF/XML 文書のルート要素は、`rdf:RDF` という名前の要素型を使って書きます。

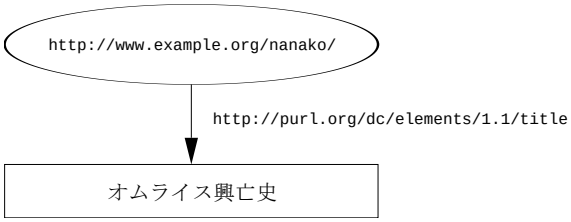
たとえば、次の XML 文書は、RDF/XML 文書の一例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.example.org/nanako/">
    <dc:creator>
      <rdf:Description rdf:about="http://example.org/staff/nanako">
      </rdf:Description>
    </dc:creator>
  </rdf:Description>
</rdf:RDF>
```

Q 1.2.10 RDF/XML では、リテラルは、どんなふうに記述すればいいんですか。

A RDF/XML では、リテラルは、そのリテラル自体を使って記述します。

たとえば、「このウェブページの題名は「オムライス興亡史」である」という文によって記述される三つ組み、すなわち、



という図で示される三つ組みは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:title>オムライス興亡史</dc:title>
</rdf:Description>
```

という要素によって記述されます。

このように、目的語がリテラルの場合は、そのリテラルをプロパティー要素の子供として書けばいいわけですが、その場合の書き方は、もうひとつあります。それは、主語をあらわすノード要素を空要素タグで書いて、その中に、述語と目的語をあらわす属性指定を書く、という書き方です。述語と目的語は、目的語がリテラルの場合、

述語 =" 目的語 "

という形の属性指定で記述することができます。ですから、上の例は、

```
<rdf:Description rdf:about="http://www.example.org/nanako/"
    dc:title="オムライス興亡史"/>
```

という要素によって記述することもできます。

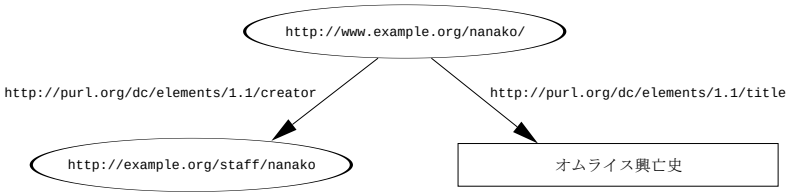
Q 1.2.11 RDF/XML では、共通の主語を持つ複数の三つ組みは、どんなふうに記述すればいいんですか。

A RDF/XML では、共通の主語を持つ複数の三つ組みは、主語をあらわすノード要素の子供として、それぞれの三つ組みの述語をあらわす複数のプロパティー要素を書くことによって記述します。

たとえば、

このウェブページの作者は高岡菜々子である。
このウェブページの題名は「オムライス興亡史」である。

という二つの文によって記述される二つの三つ組み、すなわち、



という図で示される二つの三つ組みは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator>
    <rdf:Description rdf:about="http://example.org/staff/nanako">
    </rdf:Description>
  </dc:creator>
  <dc:title>オムライス興亡史</dc:title>
</rdf:Description>
```

という要素によって記述されます。

Q 1.2.12 RDF/XML では、一方の目的語が他方の主語になってる二つの三つ組みは、どんなふうに記述すればいいんですか。

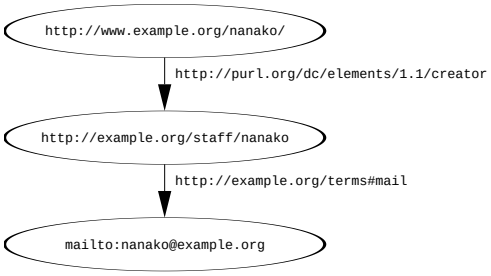
A A と B が三つ組みで、A の目的語が B の主語になっているとするとき、RDF/XML では、

それらの三つ組みは、A の目的語をあらわすノード要素の子供として B の述語をあらわすプロパティー要素を書くことによって記述します。

つまり、一方の目的語が他方の主語になっている二つの三つ組みは、外側から順番に、ノード、述語、ノード、述語、ノード、という五重の入れ子になった要素によって記述されるわけです。たとえば、

> このウェブページの作者は高岡菜々子である。
> 高岡菜々子のメールアドレスは nanako@example.org である。

という二つの文によって記述される二つの三つ組み、すなわち、



という図で示される二つの三つ組みは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator>
    <rdf:Description rdf:about="http://example.org/staff/nanako">
      <ex:mail>
        <rdf:Description rdf:about="mailto:nanako@example.org">
        </rdf:Description>
      </ex:mail>
    </rdf:Description>
  </dc:creator>
</rdf:Description>
```

という要素によって記述されます。

この例のように、RDF/XML によるグラフの記述は、ノード要素とプロパティー要素とが互い違いに組み合わさって、縞模様のように見えます。このことから、RDF/XML によるグラフの記述は、「ストライピング」(striping) と呼ばれることもあります。

目的語をあらわすノード要素は、その目的語が別の三つ組みの主語になっていないならば、空要素タグを使って書くこともできます。上の例の場合、メールアドレスの目的語は別の三つ組みの主語にはなっていませんので、

```
<rdf:Description rdf:about="mailto:nanako@example.org"/>
```

というように、空要素タグで書いてもかまいません。

また、目的語が別の三つ組みの主語になっていない場合、プロパティー要素を空要素タグで書いて、その中に目的語を属性指定の形で書く、ということも可能です。目的語をあらわす属性指定は、

```
rdf:resource="URI"
```

と書きます。この書き方を使うと、

> 高岡菜々子のメールアドレスは nanako@example.org である。

という文によって記述される三つ組みは、

```
<rdf:Description rdf:about="http://example.org/staff/nanako">
  <ex:mail rdf:resource="mailto:nanako@example.org"/>
</rdf:Description>
```

という要素によって記述することができます。

1.3 データ型

Q 1.3.1 データ型って何ですか。

A 「データ型」(datatype) というのは、データが所属している集合のことです。

別の言い方をすれば、データ型というのはデータの種類のことです。
データ型の例としては、文字列、整数、浮動小数点数、真偽値、日付などがあります。

Q 1.3.2 型付きリテラルって何ですか。

A 「型付きリテラル」(typed literal) というのは、特定の型が与えられたリテラルのことです。

Q 1.3.3 プレーンリテラルって何ですか。

A 「プレーンリテラル」(plain literal) というのは、特定の型が与えられていないリテラルのことです。

Q 1.3.4 RDF では、どのようなデータ型が定義されてるんですか。

A RDF は、「XML リテラル」(XML literal) というデータ型を定義しています。

XML リテラルというのは、XML のタグを含む文字列というデータ型のことです。たとえば、

吾輩は猿である

という文字列は、XML リテラルの一例です。

Q 1.3.5 RDF のリテラルに対して、データ型として XML リテラルを与えたいときって、どうすればいいんですか。

A プロパティー要素の `rdf:parseType` という属性に対して `Literal` という属性値を設定することによって、リテラルに対して、XML リテラルをデータ型として与えることができます。

たとえば、

```
<dc:title rdf:parseType="Literal">
  吾輩は<em>猿</em>である
</dc:title>
```

というプロパティー要素を書くことによって、その目的語となっている、

吾輩は猿である

というリテラルに対して、XML リテラルをデータ型として与えることができます。

プロパティー要素が持っている `rdf:parseType` という属性は、そのプロパティー要素の子供を解釈する方法を指示するためのもので、それに設定することのできる属性値としては、`Literal` のほかに、`Resource` と `Collection` があります。

`Resource` については Q 1.4.6 で、`Collection` については Q 1.5.6 で説明したいと思います。

Q 1.3.6 RDF のリテラルに対して与えることのできるデータ型って、XML リテラルだけなんですか。

A いいえ、RDF のリテラルに対しては、RDF の外で定義されているデータ型を与えることも可能です。

RDF のリテラルに対して与えることのできる、RDF の外で定義されているデータ型としては、XML スキーマの組み込みデータ型があります。ただし、XML スキーマの組み込みデータ型には、RDF での使用には適していないものも含まれていますので、それらのうちのすべてのデータ型を RDF で利用できるわけではありません。

XML スキーマの組み込みデータ型のうちで、RDF のリテラルに対して与えることのできるものとしては、次のようなものがあります。

`xsd:boolean`	真偽値。`true`、`false`、`1`、`0` のいずれか。
`xsd:decimal`	10 進数。
`xsd:float`	IEEE 単精度 32 ビット浮動小数点数 (IEEE 754–1985)。
`xsd:double`	IEEE 倍精度 64 ビット浮動小数点数 (IEEE 754–1985)。
`xsd:integer`	整数。

xsd:positiveInteger	プラスの整数。
xsd:negativeInteger	マイナスの整数。
xsd:nonNegativeInteger	0 以上の整数。
xsd:nonPositiveInteger	0 以下の整数。
xsd:int	−2147483648 から 2147483647 までの整数。
xsd:long	−9223372036547775808 から 9223372036547775807 までの整数。
xsd:short	−32768 から 32767 までの整数。
xsd:byte	−128 から 127 までの整数。
xsd:unsignedInt	0 から 4294967295 までの整数。
xsd:unsignedLong	0 から 18446744073709551615 までの整数。
xsd:unsignedShort	0 から 65535 までの整数。
xsd:unsignedByte	0 から 255 までの整数。
xsd:dateTime	CCYY-MM-DDThh:mm:ss という形式でグレゴリオ暦の日付と時刻を記述したもの。
xsd:date	CCYY-MM-DD という形式でグレゴリオ暦の日付を記述したもの。
xsd:time	hh:mm:ss.sss という形式で時刻を記述したもの。
xsd:gYearMonth	CCYY-MM という形式でグレゴリオ暦の年と月を記述したもの。
xsd:gYear	CCYY という形式でグレゴリオ暦の年を記述したもの。
xsd:gMonthDay	--MM-DD という形式でグレゴリオ暦の月と日を記述したもの。
xsd:gDay	--DD という形式でグレゴリオ暦の日を記述したもの。--01 から--31 まで。
xsd:gMonth	--MM--という形式でグレゴリオ暦の月を記述したもの。--01--から--12--まで。
xsd:string	文字列。
xsd:normalizedString	正規化された文字列。
xsd:token	トークン化された文字列。
xsd:language	言語識別子。
xsd:NMTOKEN	XML の NMTOKEN 型の属性値。
xsd:Name	XML において名前として使うことのできる文字列。
xsd:NCName	XML において名前として使うことのできる文字列のうちで、コロンを含まないもの。
xsd:anyURI	URI。
xsd:hexBinary	バイナリーデータを 16 進数で記述したもの。
xsd:base64Binary	バイナリーデータを base64 で符号化したもの。

Q 1.3.7 RDF のリテラルに対して、RDF の外で定義されているデータ型を与えたいときって、どうすればいいんですか。

A プロパティー要素が持っている rdf:datatype という属性に対して、データ型を識別する URI を属性値として設定すると、そのプロパティー要素の子供となっている目的語に対して、そのデータ型が与えられます。

XML スキーマの組み込みデータ型を識別する URI は、

```
http://www.w3.org/2001/XMLSchema#データ型名
```

と書きます。たとえば、date という組み込みデータ型は、

```
http://www.w3.org/2001/XMLSchema#date
```

という URI を書くことによって参照することができます。ですから、

```
<ex:creation-date
    rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
```

```
    2006-05-22
  </ex:creation-date>
```

というプロパティー要素を書くことによって、その目的語となっている、

```
2006-05-22
```

というリテラルに対して、データ型として `date` を与えることができます。

Q 1.3.8 `rdf:datatype` 属性に属性値を設定する属性指定って、名前空間接頭辞を宣言しておけば、もっと短く書けるんじゃないんですか。

A いいえ、それはできません。

残念ながら、属性値として QName を書いたとしても、その中の名前空間接頭辞は、名前空間名に対応しているとは解釈されません。

Q 1.3.9 `rdf:datatype` 属性に属性値を設定する属性指定をもっと短く書く方法って、ないんですか。

A いいえ、あります。

`rdf:datatype` 属性に属性値を設定する属性指定は、データ型を識別する URI（またはその一部分）を実体 (entity) として定義する、という方法を使うことによって、かなり短く書くことができるようになります。
たとえば、

```
rdf:datatype="http://www.w3.org/2001/XMLSchema#date"
```

という属性指定は、あらかじめ、

```
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
```

という実体宣言で `xsd` という実体を定義しておくことによって、

```
rdf:datatype="&xsd;date"
```

というように、かなり短く書くことができるようになります。

1.4 空白ノード

Q 1.4.1 空白ノードって何ですか。

A 「空白ノード」(blank node) というのは、自分を識別するための URI を持たないノードのことです。

グラフを構成するノードのうちには、それを識別する必要がないものもあります。そのようなノードは URI を持っている必要がありませんので、空白ノードにすることができます。
たとえば、日本語で書かれた次の二つの文を考えてみましょう。

> このウェブページの作者の名前は「高岡菜々子」である。
> このウェブページの作者のメールアドレスは `nanako@example.org` である。

これらの二つの日本語の文は、三つの三つ組みから構成される次のようなグラフをあらわしていると考えることができます。

> このウェブページの作者は○○○である。
> ○○○の名前は「高岡菜々子」である。
> ○○○のメールアドレスは `nanako@example.org` である。

このグラフの中で、○○○であらわされているのが空白ノードです。このグラフは、図で表現すると、図 1.1 のようになります。

Q 1.4.2 RDF/XML では、空白ノードは、どんなふうに記述すればいいんですか。

A RDF/XML では、空白ノードも、`rdf:Description` 要素を使って記述します。

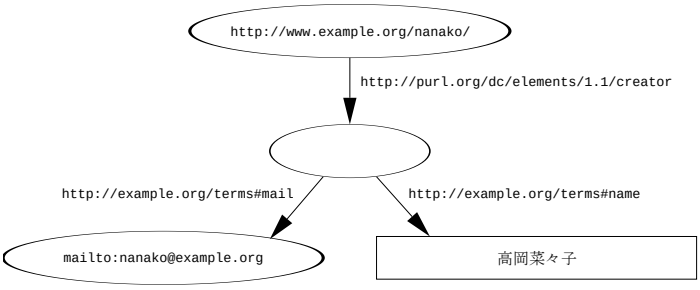


図 1.1: 空白ノードを含むグラフ

空白ノードというのは自分を識別するための URI を持たないノードのことですから、空白ノードを記述する rdf:Description 要素には、 rdf:about 属性に URI を設定する属性指定は書きません。たとえば、

> このウェブページの作者は○○○である。
> ○○○の名前は「高岡菜々子」である。
> ○○○のメールアドレスは nanako@example.org である。

という三つの文によって記述される三つの三つ組み、すなわち、図 1.1 で示される三つの三つ組みは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator>
    <rdf:Description>
      <ex:name>高岡菜々子</ex:name>
      <ex:mail rdf:resource="mailto:nanako@example.org"/>
    </rdf:Description>
  </dc:creator>
</rdf:Description>
```

という要素によって記述されます。

Q 1.4.3 空白ノード識別子って何ですか。

A 「空白ノード識別子」(blank node identifier) というのは、同じ RDF/XML 文書の中にある空白ノードを識別するための名前のことです。

空白ノードに対して空白ノード識別子を与えておくことによって、その空白ノードがあるのと同じ RDF/XML 文書の中では、その空白ノード識別子を使って、その空白ノードを識別することができるようになります。ちなみに、空白ノード識別子は、あくまで RDF/XML 文書の中だけで有効な名前ですので、外部の文書からそれを参照するということはできません。

Q 1.4.4 空白ノード識別子を定義したいときって、どうすればいいんですか。

A rdf:Description 要素が持っている rdf:nodeID という属性に対して属性値として名前を設定することによって、その要素によって記述される空白ノードに対して、空白ノード識別子としてその名前を与えることができます。

たとえば、

> ○○○の名前は「高岡菜々子」である。
> ○○○のメールアドレスは nanako@example.org である。

という二つの三つ組みは、

```
<rdf:Description rdf:nodeID="nanako">
  <ex:name>高岡菜々子</ex:name>
  <ex:mail rdf:resource="mailto:nanako@example.org"/>
</rdf:Description>
```

という要素で記述することによって、それらの主語になっている空白ノードに対して nanako という空白ノード識別子を与えることができます。

Q 1.4.5 空白ノード識別子を使って空白ノードを参照したいときって、どうすればいいんですか。

A rdf:nodeID 属性に対して、空白ノード識別子を属性値として設定することによって、それが与えられている空白ノードを参照することができます。

つまり、空白ノード識別子というのは、定義と参照とのあいだに明確な相違はありません。複数の空白ノードに対して同一の空白ノード識別子を与えれば、それらの空白ノードは同一のものとみなされる、ということなのです。

たとえば、Q 1.4.4 の例で定義した空白ノード識別子を使って空白ノードを参照することによって、

> このウェブページの作者は○○○である。

という三つ組みを記述したいときは、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator>
    <rdf:Description rdf:nodeID="nanako"/>
  </dc:creator>
</rdf:Description>
```

という要素を書きます。

この例のように、空白ノード識別子によって参照される空白ノードが目的語になっている場合、プロパティー要素で、rdf:nodeID 属性に空白ノード識別子を設定することによって、目的語のノード要素を省略することができます。ですから、上の例は、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator rdf:nodeID="nanako"/>
</rdf:Description>
```

と書いても、同じ意味になります。

Q 1.4.6 空白ノードを記述している rdf:Description 要素で、プロパティー要素とプロパティー要素の中間に位置しているものって、省略できないんですか。

A いいえ、省略できます。

空白ノードを記述している rdf:Description 要素で、プロパティー要素とプロパティー要素の中間に位置しているものは、省略することが可能です。ただし、それを省略するためには、その rdf:Description 要素の親になっているプロパティー要素の開始タグに、そのための属性指定を書く必要があります。

プロパティー要素が持っている rdf:parseType という属性に対して、Resource という属性値を設定すると、そのプロパティー要素の子供たちは、それぞれがプロパティー要素だと解釈されることになります。たとえば、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator>
    <rdf:Description>
      <ex:name>高岡菜々子</ex:name>
      <ex:mail rdf:resource="mailto:nanako@example.org"/>
    </rdf:Description>
  </dc:creator>
</rdf:Description>
```

という RDF/XML の記述は、空白ノードを記述している rdf:Description 要素を省略して、

```
<rdf:Description rdf:about="http://www.example.org/nanako/">
  <dc:creator rdf:parseType="Resource">
    <ex:name>高岡菜々子</ex:name>
    <ex:mail rdf:resource="mailto:nanako@example.org"/>
  </dc:creator>
</rdf:Description>
```

と書いても、同じ意味になります。

1.5 コンテナとコレクション

Q 1.5.1 コンテナって何ですか。

A 「コンテナ」(container) というのは、リソースのグループであるようなリソースであって、かつ、「このグループに属するリソースは、これら以外には存在しない」という限定を加えないもののことです。

コンテナに所属するそれぞれのリソースは、そのコンテナの「メンバー」(member) と呼ばれます。

Q 1.5.2 RDF/XML では、コンテナのメンバーは、どんなふうに記述すればいいんですか。

A RDF/XML では、コンテナのメンバーは、`rdf:li` という要素型の要素を使って記述します。

`rdf:li` 要素は、`rdf:resource` という属性を持っています。この属性には、コンテナのメンバーにするリソースを識別する URI を設定します。たとえば、

```
http://example.org/staff/yumiko
```

という URI によって識別されるリソースをコンテナのメンバーとして記述したいという場合は、

```
<rdf:li rdf:resource="http://example.org/staff/yumiko"/>
```

という `rdf:li` 要素を書けばいいわけです。

Q 1.5.3 RDF/XML では、コンテナは、どんなふうに記述すればいいんですか。

A RDF/XML では、コンテナは、`rdf:Bag`、`rdf:Seq`、`rdf:Alt` のいずれかの要素型の要素を使って記述します。

コンテナを記述するための 3 つの要素型は、次のように使い分けます。

`rdf:Bag` メンバーの順序が重要ではない場合。

`rdf:Seq` メンバーの順序が重要である場合。

`rdf:Alt` メンバーのうちのどれかひとつを選択する必要がある場合。

コンテナを記述したいときは、上記のいずれかの要素型を使って要素を書いて、その子供としてコンテナのメンバーを記述した要素を書きます。たとえば、

> このウェブページの作者としては、嘉田由美子と倉橋美代子と中野由紀子がいる。

という文があらわしているグラフは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/planet/">
  <dc:creator>
    <rdf:Bag>
      <rdf:li rdf:resource="http://example.org/staff/yumiko"/>
      <rdf:li rdf:resource="http://example.org/staff/miyoko"/>
      <rdf:li rdf:resource="http://example.org/staff/yukiko"/>
    </rdf:Bag>
  </dc:creator>
</rdf:Description>
```

という要素によって記述されます。

Q 1.5.4 コレクションって何ですか。

A 「コレクション」(container) というのは、リソースのグループであるようなリソースであって、かつ、「このグループに属するリソースは、これら以外には存在しない」という限定を加えるもののことです。

コレクションの中に格納されているそれぞれのリソースは、コンテナの場合と同様に、そのコレクションの「メンバー」(member) と呼ばれます。

コンテナとコレクションには、共通点と相違点があります。共通点は、どちらもリソースのグループであるようなリソースだという点で、相違点は、グループのメンバーが網羅されていると

いうことを保証するかしないかという点です。コンテナは、グループのメンバーが網羅されているということを保証しないのに対して、コレクションはそれを保証します。

Q 1.5.5 RDF/XML では、コレクションのメンバーは、どんなふうに記述すればいいんですか。

A RDF/XML では、コレクションのメンバーは、`rdf:Description` 要素を使って記述します。

たとえば、

`http://example.org/staff/yumiko`

という URI によって識別されるリソースをコレクションのメンバーとして記述したいという場合は、

```
<rdf:Description rdf:about="http://example.org/staff/yumiko"/>
```

という `rdf:Description` 要素を書きます。

Q 1.5.6 RDF/XML では、コレクションは、どんなふうに記述すればいいんですか。

A RDF/XML では、コレクションは、そのメンバーを記述しているすべての要素をプロパティー要素の子供として書くことによって記述します。

ただし、コレクションを記述する場合は、プロパティー要素の開始タグの中に、

`rdf:parseType="Collection"`

という属性指定を書いておく必要があります。この属性指定は、プロパティー要素の子供たちがひとつのコレクションだということを意味しています。

たとえば、

> このウェブページの作者は嘉田由美子と倉橋美代子と中野由紀子で、ほかにはいない。

という文があらわしているグラフは、RDF/XML では、

```
<rdf:Description rdf:about="http://www.example.org/planet/">
  <dc:creator rdf:parseType="Collection">
    <rdf:Description rdf:about="http://example.org/staff/yumiko"/>
    <rdf:Description rdf:about="http://example.org/staff/miyoko"/>
    <rdf:Description rdf:about="http://example.org/staff/yukiko"/>
  </dc:creator>
</rdf:Description>
```

という要素によって記述されます。

第 2 章　RDF 語彙

2.1　RDF 語彙の基礎

Q 2.1.1 個体って何ですか。

A 「個体」(individual) というのは、具体的に存在するリソースのことです。

たとえば、第 1 章で例として使ったリソースのうち、「このウェブページ」や「高岡菜々子」などは、具体的に存在するものですので、個体だと考えることができます。

Q 2.1.2 クラスって何ですか。

A 「クラス」(class) というのは、個体の種類であるようなリソースのことです。

たとえば、「ウェブページ」や「人間」というリソースは、個体の種類であるようなリソースですので、クラスだと考えることができます。

Q 2.1.3 インスタンスって何ですか。

A クラスに所属している個体は、そのクラスの「インスタンス」(instance) と呼ばれます。

たとえば、「高岡奈々子」が「人間」というクラスに所属しているとするならば、「高岡奈々子は人間のインスタンスである」と言うことができます。

Q 2.1.4　プロパティーって何ですか。

A　「プロパティー」(property) というのは、リソースとリソースとのあいだの関係であるようなリソースのことです。

たとえば、第 1 章で例として使ったリソースのうち、「作者」や「題名」などは、リソースとリソースとのあいだの関係ですので、プロパティーだと考えることができます。
Q 1.1.3 で説明したように、「何々の何々は何々である」という RDF 文が言及している三つのリソースのうちで、「何々は」に相当するもののことを「述語」または「プロパティー」と呼びます。つまり、「述語」と「プロパティー」は、ほとんど同じ意味の言葉だと考えていい、ということです。

Q 2.1.5　RDF 語彙って何ですか。

A　「RDF 語彙」(RDF vocabulary) というのは、RDF/XML でリソースを記述するために使われるクラスまたはプロパティーのことです。

RDF 語彙は、RDF に関する文脈では普通、単に「語彙」(vocabulary) と呼ばれます。

Q 2.1.6　RDF 語彙記述言語って何ですか。

A　「RDF 語彙記述言語」(RDF vocabulary description language) というのは、RDF 語彙についての記述を書くための言語のことです。

Q 2.1.7　RDF スキーマって何ですか。

A　「RDF スキーマ」(RDF Schema) というのは、RDF 語彙記述言語のひとつです。

RDF スキーマは、W3C の勧告によって規定されている言語です。
RDF スキーマは、語彙について記述するために必要となる名前を RDF/XML に追加したものです。語彙について記述するために使う文法は、RDF/XML とまったく同じです。

Q 2.1.8　RDF スキーマで定義されてる名前って、どんな名前空間に属してるんですか。

A　RDF スキーマで定義されている名前が属しているのは、

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

という名前空間です。

RDF スキーマの名前空間接頭辞としては、通常、`rdfs` が使われます。

Q 2.1.9　RDF スキーマ文書って何ですか。

A　「RDF スキーマ文書」(RDF Schema document) というのは、RDF スキーマを使って書かれた文書のことです。

RDF スキーマ文書は、RDF スキーマで定義されている名前を使って書かれている RDF 文書です。ルート要素の要素型としては、`rdf:RDF` を使います。

Q 2.1.10　RDF/XML では、「何々は何々のインスタンスである」っていう言明は、どんなふうに記述すればいいんですか。

A　RDF/XML では、`rdf:type` というプロパティーを使って、「何々は何々のインスタンスである」という言明を記述します。

`rdf:type` というのは、「主語は目的語のインスタンスである」ということをあらわすプロパティーです。ですから、「主語は目的語のインスタンスである」という言明は、

主語の `rdf:type` は目的語である。

という RDF 文で記述することができます。

それでは、例として、

高岡菜々子は人間のインスタンスである。

という言明を RDF/XML で記述してみましょう。「高岡菜々子」と「人間」というリソースのそれぞれが、

高岡菜々子　`http://example.org/staff/nanako`
人間　`http://example.org/terms/Person`

という URI で識別されるとすると、その言明は、

```
<rdf:Description rdf:about="http://example.org/staff/nanako">
  <rdf:type rdf:resource="http://example.org/terms/Person"/>
</rdf:Description>
```

という要素によって記述することができます。

Q 2.1.11　RDF/XML には、「何々は何々のインスタンスである」っていう言明を、もっと簡潔に書く方法って、ないんですか。

A　いいえ、あります。

RDF/XML では、「主語は目的語のインスタンスである」という言明、つまり、

主語の `rdf:type` は目的語である。

という RDF 文であらわされる言明は、

<目的語 `rdf:about="` 主語 `"/>`

という略記法で記述することができます。

RDF/XML は、「「主語は目的語のインスタンスである」ということを略記法で記述した要素」という概念を意味する言葉を定義していません。しかし、その概念に言及することはしばしば必要になりますので、何か言葉を定義しておくと便利です。[神崎,2005] では、その概念は「型付ノード要素」と呼ばれていますので、このチュートリアルでも、この言葉を使うことにします。

Q 2.1.10 で例として使った、

高岡菜々子は人間のインスタンスである。

という言明は、

```
xmlns:ex="http://example.org/terms/"
```

という名前空間宣言によって名前空間名と名前空間接頭辞とが結び合わされているとすると、

```
<ex:Person rdf:about="http://example.org/staff/nanako"/>
```

という型付ノード要素によって記述することができます。

型付ノード要素は、`rdf:about` に設定されたリソースを記述したノード要素だとみなすことができます。ですから、RDF/XML で RDF 文を記述するときに、主語または目的語として型付ノード要素を書くことも可能です。

たとえば、

高岡菜々子は人間のインスタンスである。
高岡菜々子の氏名は「高岡菜々子」である。

という二つの言明は、`ex:name` という QName が「氏名」というプロパティーをあらわしているとすれば、

```
<ex:Person rdf:about="http://example.org/staff/nanako">
  <ex:name>高岡菜々子</ex:name>
</ex:Person>
```

という要素によって記述することができます。

2.2 断片識別子

Q 2.2.1　断片識別子って何ですか。

A 「断片識別子」(fragment identifier) というのは、RDF/XML 文書の中でリソースを参照するために使うことのできる名前のことです。

RDF/XML 文書は、「断片識別子」と呼ばれる名前をリソースに割り当てておいて、その名前を使ってリソースを参照するということができます。

Q 2.2.2 RDF/XML で、リソースに断片識別子を割り当てたいときって、どうすればいいんですか。

A RDF/XML では、`rdf:ID` という属性を使うことによって、リソースに断片識別子を割り当てることができます。

`rdf:Description` 要素は、`rdf:ID` という属性を持っています。この属性に対して名前を設定すると、その名前は、その要素によって記述されたリソースに対して、断片識別子として割り当てられます。たとえば、

```
<rdf:Description rdf:ID="no00582">
  <ex:title>野菜学概論</ex:title>
</rdf:Description>
```

という要素は、この要素によって記述されたリソースに対して `no00582` という名前を断片識別子として割り当てます。

Q 2.2.3 RDF/XML では、断片識別子が割り当てられたリソースを参照したいときって、どうすればいいんですか。

A リソースに断片識別子を割り当てた RDF/XML 文書の中では、その断片識別子から作られた相対 URI を使って、そのリソースを参照することができます。

断片識別子の相対 URI は、その断片識別子の左側に `#` を書くことによって作られます。
たとえば、Q 2.2.2 で例として書いた要素を含んでいる RDF/XML 文書の中では、その要素が記述したリソースを、`#no00582` という相対 URI で参照することができます。ですから、

```
<rdf:Description rdf:about="#no00582">
  <ex:publishedYear>1978</ex:publishedYear>
</rdf:Description>
```

というような要素を書くことによって、`no00582` という断片識別子が割り当てられたリソースについて記述することができます。

Q 2.2.4 RDF/XML では、断片識別子が割り当てられたリソースを、それを割り当てた文書とは別の文書の中から参照したいときって、どうすればいいんですか。

A リソースに断片識別子を割り当てた RDF/XML 文書とは別の RDF/XML 文書の中から、そのリソースを参照したいときは、その断片識別子から作られた絶対 URI を使います。

断片識別子の絶対 URI は、その断片識別子の相対 URI の左側に、その断片識別子をリソースに割り当てた RDF/XML 文書の URI、つまり基底 URI を連結することによって作られます。
たとえば、Q 2.2.2 で例として書いた要素を含んでいる RDF/XML 文書が、

```
http://example.org/library/catalog.rdf
```

という URI で識別されるとすると、それとは別の RDF/XML 文書の中では、

```
http://example.org/library/catalog.rdf#no00582
```

という絶対 URI によって、`no00582` という断片識別子が割り当てられたリソースを参照することができます。

Q 2.2.5 RDF/XML 文書の基底 URI として、実際の URI とは異なるものを与えたいときって、どうすればいいんですか。

A RDF/XML 文書のルート要素で、`xml:base` という属性に URI を設定することによって、その URI をその RDF/XML 文書の基底 URI にすることができます。

たとえば、

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="http://example.org/terms/"
    xml:base="http://example.org/book">
  <rdf:Description rdf:ID="no00582">
    <ex:title>野菜学概論</ex:title>
  </rdf:Description>
</rdf:RDF>
```

というように、RDF/XML 文書のルート要素で、xml:base 属性に URI を設定しておけば、その RDF/XML 文書の実際の URI とは無関係に、

```
http://example.org/book#no00582
```

という絶対 URI によって、その RDF/XML 文書の中で断片識別子が割り当てられているリソースを参照することができるようになります。

Q 2.2.6 リソースに断片識別子を割り当てる要素を、型付ノード要素で書くことは可能ですか。

A はい、可能です。

rdf:about の属性指定の代わりに rdf:ID の属性指定を持つ型付ノード要素を書くと、それによって記述されたリソースに対して断片識別子が割り当てられることになります。たとえば、ex:Book という QName が「本」というクラスをあらわしているとすると、

```
<ex:Book rdf:ID="no01706"/>
```

という型付ノード要素を書くことによって、この要素で記述された本に対して no01706 という断片識別子を割り当てることができます。

2.3 クラス

Q 2.3.1 RDF/XML では、「何々はクラスである」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、rdfs:Class というクラスを使って、「何々はクラスである」という言明を記述します。

rdfs:Class というのは、「クラス」をあらわすクラスです。
「何々はクラスである」という言明は、言い換えると、「何々はクラスというクラスのインスタンスである」ということですから、

何々の rdf:type は rdfs:Class である。

という RDF 文で記述することができます。そして、この形の RDF 文は、

```
<rdfs:Class rdf:about="何々"/>
```

または、

```
<rdfs:Class rdf:ID="何々"/>
```

という型付ノード要素で記述することができます。
それでは、例として、

人間はクラスである。

という言明を RDF/XML で記述してみましょう。「人間」というリソースに対して Person という断片識別子を割り当てるとすると、その言明は、

```
<rdfs:Class rdf:ID="Person"/>
```

という要素によって記述することができます。

Q 2.3.2 サブクラスって何ですか。

A A と B がクラスで、A を特殊化したものが B だとするとき、B は A の「サブクラス」(subclass) である、と言います。

クラスとクラスとのあいだには、一方を特殊化したものが他方であるという関係が成り立っている場合があります。その場合、特殊なほうのクラスは、一般的なほうのクラスの「サブクラス」であると言われます。

たとえば、「人間」というクラスと「職員」というクラスとのあいだには、前者を特殊化したものが後者であるという関係が成り立っています。ですから、「職員は人間のサブクラスである」と言うことができます。

RDF や RDF スキーマで定義されているクラスの大多数も、何らかのクラスのサブクラスです。たとえば、`rdfs:Class` は、`rdfs:Resource` というクラスのサブクラスです。

`rdfs:Resource` というのは、「リソース」をあらわすクラスです。これは、RDF や RDF スキーマで定義されているクラスの中でもっとも一般的なクラスですので、このクラスをサブクラスとするクラスというのは存在しません。

Q 2.3.3 RDF/XML では、「何々は何々のサブクラスである」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、`rdfs:subClassOf` というプロパティーを使って、「何々は何々のサブクラスである」という言明を記述します。

`rdfs:subClassOf` というのは、「主語は目的語のサブクラスである」ということをあらわすプロパティーです。

それでは、

> 職員は人間のサブクラスである。

という言明を RDF/XML で記述してみましょう。「職員」に対して `Staff` という断片識別子、「人間」に対して `Person` という断片識別子が割り当てられているとすると、その言明は、

```
<rdf:Description rdf:about="#Staff">
  <rdfs:subClassOf rdf:resource="#Person"/>
</rdf:Description>
```

という要素によって記述することができます。

Q 2.3.4 何かがクラスであるという記述と、それが何のサブクラスなのかという記述とを、ひとつにまとめて書くことって、可能ですか。

A はい、可能です。

Q 2.1.11 で説明したように、型付ノード要素はノード要素だとみなすことができます。ですから、それを応用すれば、

> ○○○はクラスである。
> ○○○は●●●のサブクラスである。

という二つの言明をひとつにまとめて記述することも可能です。たとえば、

> 職員はクラスである。
> 職員は人間のサブクラスである。

という二つの言明は、

```
<rdfs:Class rdf:ID="Staff">
  <rdfs:subClassOf rdf:resource="#Person"/>
</rdfs:Class>
```

という要素によって記述することができます。

Q 2.3.5 既定義 RDF クラスって何ですか。

A 「既定義 RDF クラス」(predefined RDF class) というのは、RDF または RDF スキーマで定義されているクラスのことです。

RDF で定義されている既定義 RDF クラスには、次のようなものがあります。

rdf:Property	プロパティーのクラス。rdfs:Resource のサブクラス。Q 2.4.1 参照。
rdf:XMLLiteral	XML リテラル（Q 1.3.4 参照）のクラス。rdfs:Literal（Q 2.4.9 参照）のサブクラス。

また、RDF スキーマで定義されている既定義 RDF クラスには、次のようなものがあります。

rdfs:Resource	リソースのクラス。Q 2.3.2 参照。
rdfs:Literal	リテラルのクラス。rdfs:Resource のサブクラス。Q 2.4.9 参照。
rdfs:Class	クラスのクラス。rdfs:Resource のサブクラス。Q 2.3.1 参照。
rdfs:Datatype	データ型のクラス。rdfs:Class のサブクラス。Q 2.4.11 参照。

2.4 プロパティー

Q 2.4.1 RDF/XML では、「何々はプロパティーである」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、rdf:Property というクラスを使って、「何々はプロパティーである」という言明を記述します。

rdf:Property というのは、「プロパティー」をあらわすクラスで、rdfs:Resource のサブクラスです。

「何々はプロパティーである」という言明は、言い換えると、「何々はプロパティーというクラスのインスタンスである」ということですから、

何々の rdf:type は rdf:Property である。

という RDF 文で記述することができます。そして、この形の RDF 文は、

```
<rdf:Property rdf:about="何々"/>
```

または、

```
<rdf:Property rdf:ID="何々"/>
```

という型付ノード要素で記述することができます。

それでは、例として、

「好きな本」はプロパティーである。

という言明を RDF/XML で記述してみましょう。favoriteBook という断片識別子を「好きな本」というリソースに対して割り当てるとすると、その言明は、

```
<rdf:Property rdf:ID="favoriteBook"/>
```

という要素によって記述することができます。

Q 2.4.2 定義域って何ですか。

A 何らかのプロパティーについて、その主語が所属しているクラスを、そのプロパティーの「定義域」(domain) と呼びます。

たとえば、「好きな本」というプロパティーの主語は「人間」というクラスのインスタンスですので、「「好きな本」の定義域は人間である」ということになります。

Q 2.4.3 値域って何ですか。

A 何らかのプロパティーについて、その目的語が所属しているクラスを、そのプロパティーの「値域」(range) と呼びます。

たとえば、「好きな本」というプロパティーの目的語は「本」というクラスのインスタンスですので、「「好きな本」の値域は本である」ということになります。

Q 2.4.4 RDF/XML では、「何々の定義域は何々である」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、rdfs:domain というプロパティーを使って、「何々の定義域は何々である」という言明を記述します。

rdfs:domain というのは、「主語の定義域は目的語である」ということをあらわすプロパティーで、その定義域は rdf:Property で、その値域は rdfs:Class です。
それでは、例として、

「好きな本」の定義域は人間である。

という言明を RDF/XML で記述してみましょう。「好きな本」に対して favoriteBook、「人間」に対して Person という断片識別子が割り当てられているとすると、その言明は、

```
<rdf:Description rdf:about="#favoriteBook">
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Description>
```

という要素によって記述することができます。

Q 2.4.5 RDF/XML では、「何々の値域は何々である」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、rdfs:range というプロパティーを使って、「何々の値域は何々である」という言明を記述します。

rdfs:range というのは、「主語の値域は目的語である」ということをあらわすプロパティーで、その定義域は rdf:Property で、その値域は rdfs:Class です。
それでは、例として、

「好きな本」の値域は本である。

という言明を RDF/XML で記述してみましょう。「好きな本」に対して favoriteBook、「本」に対して Book という断片識別子が割り当てられているとすると、その言明は、

```
<rdf:Description rdf:about="#favoriteBook">
  <rdfs:range rdf:resource="#Book"/>
</rdf:Description>
```

という要素によって記述することができます。

Q 2.4.6 何かがプロパティーだという記述と、そのプロパティーの定義域と値域の記述をひとつにまとめて書くことって、可能ですか。

A はい、可能です。

Q 2.1.11 で説明したように、型付ノード要素はノード要素だとみなすことができます。ですから、それを応用すれば、

○○○はプロパティーである。
○○○の定義域は●●●である。
○○○の値域は◎◎◎である。

という三つの言明をひとつにまとめて記述することも可能です。たとえば、

「好きな本」はプロパティーである。
「好きな本」の定義域は人間である。
「好きな本」の値域は本である。

という三つの言明は、

```
<rdf:Property rdf:ID="favoriteBook">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Book"/>
</rdf:Property>
```

という要素によって記述することができます。

Q 2.4.7 サブプロパティーって何ですか。

A A と B がプロパティーで、A を特殊化したものが B だとするとき、B は A の「サブプロパティー」(subproperty) である、と言います。

プロパティーとプロパティーとのあいだには、一方を特殊化したものが他方であるという関係が成り立っている場合があります。その場合、特殊なほうのプロパティーは、一般的なほうのプロパティーの「サブプロパティー」であると言われます。
たとえば、「好きな本」というプロパティーと「一番好きな本」というプロパティーとのあいだには、前者を特殊化したものが後者であるという関係が成り立っています。ですから、「「一番好きな本」は「好きな本」のサブプロパティーである」と言うことができます。

Q 2.4.8 RDF/XML では、「何々は何々のサブプロパティーである」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、`rdfs:subPropertyOf` というプロパティーを使って、「何々は何々のサブプロパティーである」という言明を記述します。

`rdfs:subPropertyOf` というのは、「主語は目的語のサブプロパティーである」ということをあらわすプロパティーで、その定義域と値域は、ともに `rdf:Property` です。
それでは、

「一番好きな本」は「好きな本」のサブプロパティーである。

という言明を RDF/XML で記述してみましょう。「一番好きな本」に対して `mostFavoriteBook`、「好きな本」に対して `favoriteBook` という断片識別子が割り当てられているとすると、その言明は、

```
<rdf:Description rdf:about="#mostFavoriteBook">
  <rdfs:subPropertyOf rdf:resource="#favoriteBook"/>
</rdf:Description>
```

という要素によって記述することができます。

Q 2.4.9 RDF/XML では、「何々の値域はリテラルである」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、`rdfs:Literal` というクラスを使って、「何々の値域はリテラルである」という言明を記述します。

`rdfs:Literal` というのは、「リテラル」をあらわすクラスで、`rdfs:Resource` のサブクラスです。
それでは、例として、

「題名」の値域はリテラルである。

という言明を RDF/XML で記述してみましょう。あらかじめ、

```
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
```

という実体宣言で `rdfs` という実体が定義されていて、「題名」に対して `title` という断片識別子が割り当てられているとすると、その言明は、

```
<rdf:Description rdf:about="#title">
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Description>
```

という要素によって記述することができます。

Q 2.4.10 XML スキーマで定義されているデータ型を値域として使うことは可能ですか。

A はい、可能です。

それでは、例として、

「ページ数」の値域は `xsd:nonNegativeInteger` である。

という言明を RDF/XML で記述してみましょう。あらかじめ、

```
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
```

という実体宣言で xsd という実体が定義されていて、pages という断片識別子が「ページ数」というリソースに割り当てられているとすると、その言明は、

```
<rdf:Description rdf:about="#pages">
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Description>
```

という要素によって記述することができます。

プロパティーの値域として何らかのデータ型を記述した場合、そのプロパティーを使って RDF/XML の要素を書くときには、そのプロパティーの値に対してそのデータ型を与える必要があります。

たとえば、上の例のように、

「ページ数」の値域は xsd:nonNegativeInteger である。

という記述を書いた場合は、

no01704 のページ数は 328 である。

という言明を記述する場合、

```
<rdf:Description rdf:about="#no01704">
  <ex:pages rdf:datatype="&xsd;nonNegativeInteger">
    328
  </ex:pages>
</rdf:Description>
```

というように、rdf:datatype 属性を使って、xsd:nonNegativeInteger というデータ型をプロパティーの値に与える必要があります。

Q 2.4.11 RDF/XML では、「何々はデータ型である」っていう言明は、どんなふうに記述すればいいんですか。

A RDF/XML では、rdfs:Datatype というクラスを使って、「何々はデータ型である」という言明を記述します。

rdfs:Datatype というのは、「データ型」をあらわすクラスで、rdfs:Class のサブクラスです。それでは、例として、

xsd:nonNegativeInteger はデータ型である。

という言明を RDF/XML で記述してみましょう。あらかじめ、

```
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
```

という実体宣言で xsd という実体が定義されているとすると、その言明は、

```
<rdfs:Datatype rdf:about="&xsd;nonNegativeInteger"/>
```

という要素によって記述することができます。

2.5 ダブリンコア

Q 2.5.1 メタデータって何ですか。

A 「メタデータ」(metadata) というのは、データについてのデータのことです。

たとえば、何らかのデータについて、「このデータの作者は誰々である」とか、「このデータの題名はこれこれである」というような言明を記述したデータは、そのデータのメタデータであると言われます。

Q 2.5.2 ダブリンコアって何ですか。

A 「ダブリンコア」(Dublin Core) というのは、メタデータを記述するために使われる RDF 語彙の集合です。

ダブリンコアの開発は、1995 年 3 月にダブリン[1]で開催された、メタデータに関するワークショ

[1] アイルランドのダブリンではなくて、合衆国オハイオ州のダブリンです。

ップで開始されました。現在は、DCMI(Dublin Core Metadata Initiative)[2]という組織によって開発が進められています。

Q 2.5.3 DCMESって何ですか。

A DCMES(Dublin Core Metadata Element Set) というのは、ダブリンコアが定義している語彙のうちで、もっとも基本的なプロパティーから構成されている集合のことです。

DCMES で定義されている名前は、

```
http://purl.org/dc/elements/1.1/
```

という名前空間に属していて、名前空間接頭辞としては、通常、dc が使われます。
DCMES が定義しているのは、次の 15 個のプロパティーです。

dc:title	リソースの題名。
dc:creator	リソースを作成した人間や組織。
dc:subject	リソースの主題。
dc:description	リソースの要約や概要など。
dc:publisher	リソースを公開している人間や組織。
dc:contributor	リソースの作成に貢献した人間や組織。
dc:date	リソースを公開した日付。W3CDTF 形式 (YYYY-MM-DDThh:mm:ss+TZ)。
dc:type	リソースの内容の性質。 Collection Dataset Event Image Interactive resource Service Software Sound Text
dc:format	リソースの MIME タイプ。
dc:identifier	リソースへの曖昧ではない参照。
dc:source	リソースを派生させる源泉となったリソースへの参照。
dc:language	リソースが使用している言語。jp、en、de、fr などの ISO639 の言語コード。
dc:relation	リソースに関連するリソースへの参照。
dc:coverage	リソースの内容が適用される範囲（地域や期間など）。
dc:rights	リソースに関連する権利。

次の例は、DCMES を使ってウェブサイトについて記述した RDF/XML 文書です。

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://aquinas.jp/">
    <dc:title>トマス・アクィナス・リンク集</dc:title>
    <dc:subject>トマス・アクィナス</dc:subject>
    <dc:format>text/html</dc:format>
    <dc:language>jp</dc:language>
  </rdf:Description>
</rdf:RDF>
```

第 3 章　OWL の基礎

3.1　オントロジー

Q 3.1.1 概念化って何ですか。

A 「概念化」(conceptualization) というのは、世界を認識するための概念の体系を構築することです。

ここで「世界」と呼んでいるものは、文字通りの全世界であってもかまいませんし、特定の知識分野であってもかまいません。

[2]DCMI のウェブサイトは、http://dublincore.org/ です。

人間が世界を認識するためには、その世界に適用することのできる概念を作って、それらの概念のあいだの関係について考察すること、つまり概念の体系を構築することが必要です。そのような作業のことを「概念化」と呼びます。

Q 3.1.2　オントロジーって何ですか。

A　「オントロジー」(ontology) というのは、概念の体系について形式的に記述したもののことです。

オントロジーは、形式的な記述である必要があります。ここで使われている「形式的な」という言葉は、人間だけではなくてコンピュータもそれを処理することができるような形の、という意味です。

オントロジーは、コンピュータを使って何らかの世界についての知識を処理したい、というときに必要になります。つまり、知識というものをコンピュータに処理させるためには、対象となっている世界を概念化することによって得られた概念の体系について記述したもの、つまりオントロジーをコンピュータに与える必要があるということです。

「オントロジー」という日本語の単語に対応する ontology という英単語には、「存在とは何かということについて探究する哲学の分野」という意味もあります。この意味での ontology は、日本語では「存在論」と呼ばれます。

Q 3.1.3　意味論って何ですか。

A　「意味論」(semantics) というのは、記号の列が持っている意味についての記述のことです。

たとえば、辞書というのは、言葉が持っている意味について記述しているわけですから、そこに書かれている内容は意味論だと考えることができます。

何らかの世界で使われている言葉の意味は、その世界を認識するための概念の体系を構成します。ですから、それらの言葉の意味論を形式的に記述すると、それは、その世界のオントロジーになります。

Q 3.1.4　セマンティックウェブって何ですか。

A　「セマンティックウェブ」(semantic web) というのは、オントロジーを持つウェブのことです。

「セマンティック」(semantic) というのは「意味論的な」という意味です。したがって、「セマンティックウェブ」というのは直訳すれば「意味論的ウェブ」ということになります。しかし、意味論を持っているウェブを作ったとしても、その意味論が、人間にしか処理できない形で書かれていたならば、そのウェブを「セマンティックウェブ」と呼ぶことはできません。セマンティックウェブは、コンピュータに処理できる形で意味論を記述したもの、つまりオントロジーを持っている必要があります。

Q 3.1.5　ウェブがオントロジーを持つことによって、どんな利点があるんですか。

A　ウェブがオントロジーを持つことによる利点のうちでもっとも顕著なのは、ウェブに対する質問や問い合わせが可能になるということです。

人間がウェブを使って何らかの疑問を解決しようと思った場合、そのウェブがオントロジーを持っていないとすれば、そのためのもっとも有効な手段は、ウェブの中で文字列の検索をすることです。しかし、文字列の検索は、疑問を解決するための手段としては、かなり歯がゆいものだと言っていいでしょう。

疑問を解決するための手段として、ウェブに対して質問や問い合わせをすることができれば、そのほうが文字列の検索よりもストレートです。オントロジーを持つウェブ、すなわちセマンティックウェブでは、コンピュータが、オントロジーを処理することによって人間による質問や問い合わせに答えることができるのです。

3.2　OWL の基礎の基礎

Q 3.2.1　オントロジー記述言語って何ですか。

A 「オントロジー記述言語」(ontology description language) というのは、オントロジーを記述するための言語のことです。

オントロジー記述言語としては、Ontolingua、DAML-ONT(DARPA Agent Markup Language Ontology)、OIL(Ontology Inference Layer)、OWL(Web Ontology Language) などがあります。

Q 3.2.2 OWLって何ですか。

A OWL(Web Ontology Language) というのは、オントロジー記述言語のひとつで、ウェブのオントロジーの記述を目的とするものです。

OWL は、オントロジーを記述するために必要となるさまざまな語彙を RDF スキーマに追加した言語で、RDF や RDF スキーマと同様に、W3C の勧告によって規定されています。
ちなみに、OWL は、「アウル」と発音します。

Q 3.2.3 OWL が Web Ontology Language の略称だとすれば、OWL じゃなくて WOL が正しいんじゃないんですか。

A Web Ontology Language の略称が WOL ではなくて OWL になったのは、OWL が作られるよりも以前から存在していた WOL というデータベース操作言語との混同を避けるためという理由と、OWL にすることによって智恵の象徴であるフクロウ (owl) と同じ綴りになるからという理由によるものです。

Q 3.2.4 OWL オントロジーって何ですか。

A OWL では、RDF グラフのことを「OWL オントロジー」(OWL ontology) と呼びます。

つまり、「OWL オントロジー」は「RDF グラフ」の別名です。

Q 3.2.5 OWL 文書って何ですか。

A 「OWL 文書」(OWL document) というのは、OWL オントロジーを記述した文書のことです。

OWL オントロジーは、RDF グラフと同じように、さまざまな文法を使って記述することができます。そして、もっとも標準的な文法が RDF/XML だという点も、RDF グラフと同様です。
なお、このチュートリアルでは、RDF/XML を使って書かれている OWL 文書に言及する場合、記述に使用する文法については明記しないで、それを単に「OWL 文書」と書くことにします。

Q 3.2.6 OWL 文書って、MIME タイプは何ですか。

A OWL 文書の MIME タイプは、`application/rdf+xml` です。

つまり、RDF/XML を使って記述されていれば、RDF/XML 文書も OWL 文書も MIME タイプは同じだということです。

Q 3.2.7 OWL 文書を保存するファイルって、どんな拡張子を付ければいいんですか。

A OWL 文書を保存するファイルには、`.rdf` または `.owl` という拡張子を付けることが推奨されています。

Q 3.2.8 OWL 組み込み語彙って何ですか。

A 「OWL 組み込み語彙」(OWL built-in vocabulary) というのは、OWL 自身によって定義されている語彙のことです。

Q 3.2.9 OWL 組み込み語彙って、どんな名前空間に属してるんですか。

A OWL 組み込み語彙が属しているのは、

```
http://www.w3.org/2002/07/owl#
```

という名前空間です。

OWL 組み込み語彙の名前空間接頭辞としては、通常、`owl` が使われます。

Q 3.2.10　OWL 文書のルート要素って、どんな要素型を使って書けばいいんですか。

A　OWL 文書のルート要素は、RDF/XML 文書と同じように、`rdf:RDF` という名前の要素型を使って書きます。

Q 3.2.11　OWL 文書って、どのような部分から構成されるんですか。

A　OWL 文書は、0 個または 1 個の「オントロジーヘッダー」(ontology header) と、そして任意の個数の「クラス公理」(class axiom)、「プロパティー公理」(property axiom)、「個体公理」(individual axiom) から構成されます。

OWL 文書の構成要素は、どのような順序で書いてもかまいません。また、それらの順序は意味を持ちません。

オントロジーヘッダーについては第 3.3 節、クラス公理については第 4 章、プロパティー公理については第 5 章、個体公理については第 6 章で説明したいと思います。

Q 3.2.12　OWL の下位言語って何ですか。

A　OWL の「下位言語」(sublanguage) というのは、OWL の部分集合である三つの言語のそれぞれのことです。

OWL には三つの下位言語があって、それぞれ、OWL Lite、OWL DL、OWL Full と呼ばれます。これらの下位言語は、それぞれが OWL の部分集合になっているわけですが、

$$\text{OWL Lite} \subset \text{OWL DL} \subset \text{OWL Full} = \text{OWL}$$

というように、それぞれの下位言語のあいだには包含関係があって、OWL Full と OWL とは同じものです。

任意の RDF グラフは、それを OWL オントロジーとみなすことが可能ですが、かならずしも OWL Lite オントロジーまたは OWL DL オントロジーとみなすことができるとは限りません。

Q 3.2.13　OWL Liteって、どういう下位言語なんですか。

A　OWL Lite は、アプリケーションを簡単に実装できるようにすることを目的とする OWL の下位言語です。

OWL Lite は、三つの下位言語のうちで、もっとも記述力が制限されている言語です。したがって、比較的単純なオントロジーしか記述することができません。しかし、そのオントロジーを処理するアプリケーションの実装は、他の下位言語の場合に比べて容易です。

Q 3.2.14　OWL DLって、どういう下位言語なんですか。

A　OWL は、完全な推論を実現するための制約が加えられた OWL の下位言語です。

DL という名前は、「記述論理」(description logic) と呼ばれる論理体系に由来するものです。

OWL Full で記述されたオントロジーでは、記述論理のもとで完全な推論を実行することができません。それを完全なものにするためには、「型分離」(type separation) と呼ばれる制約を OWL Full に加えた言語でオントロジーを記述する必要があります。

型分離というのは、「クラスは個体にもプロパティーにもなることができず、プロパティーは個体にもクラスにもなることができない」という制約のことです。OWL DL は、OWL Full に対して型分離などの制約が加えられたものです。

Q 3.2.15　開世界仮説って何ですか。

A　「開世界仮説」(open world assumption) というのは、「記述されていない命題の真偽は不明である」という仮説のことです。

OWL は開世界仮説を採用しています。このことは、ひとつのリソースについての記述は、ひとつの OWL オントロジーの中だけに書かれているとは限らない、ということを意味しています。ですから、すでに存在している OWL オントロジーの中で記述されているリソースについて、そ

れとは別の OWL オントロジーを書くことによって、記述を拡張することができる、ということになります。

ただし、記述の拡張は、「単調性」(monotonicity) を持っています。単調性というのは、古い記述とは矛盾する新しい記述を書いたとしても、それによって古い記述が否定されるわけではない、という性質のことです。

ちなみに、開世界仮説とは対照的な、「記述されていない命題は偽である」という仮説は、「閉世界仮説」(closed world assumption) と呼ばれます。

Q 3.2.16 一意名仮説って何ですか。

A 「一意名仮説」(unique name assumption) というのは、「ひとつの個体の名前はひとつだけしかない」という仮説のことです。

言い換えれば、一意名仮説というのは、「名前が異なっているならば、それらの名前を持つそれぞれのものは異なる個体である」という仮説のことです。

OWL は一意名仮説を採用していません。ですから、OWL では、異なる複数の名前がひとつの個体に与えられている場合もある、ということになります。

OWL は、ウェブ上に分散しているいくつかのオントロジーによってひとつの主題に関するオントロジーを構成することができるようにする、という方針のもとに設計されています。OWL が開世界仮説を採用しているのも、一意名仮説を採用していないのも、そのような設計方針によるものです。

3.3 オントロジーヘッダー

Q 3.3.1 オントロジーヘッダーって何ですか。

A 「オントロジーヘッダー」(ontology header) というのは、OWL 文書の中に書かれる、その OWL 文書自身についての言明を記述した要素のことです。

オントロジーヘッダーは、書いてもかまいませんし、書かなくてもかまいません。ただし、ひとつの OWL 文書の中に 2 個以上のオントロジーヘッダーを書くことはできません。

OWL 文書の中にオントロジーヘッダーを書く位置については、特に規定はありませんが、ルート要素の最初の子供として書くのが普通です。

Q 3.3.2 オントロジーヘッダーって、どんな要素型の要素なんですか。

A オントロジーヘッダーは、`owl:Ontology` という要素型の要素です。

`owl:Ontology` というのは、OWL 文書のクラスです。そして、もっとも短いオントロジーヘッダーは、

```
<owl:Ontology rdf:about=""/>
```

と書かれることになります。

このことからわかるとおり、実は、オントロジーヘッダーというのは、

> この文書は OWL 文書である。

という言明を記述した型付ノード要素のことなのです。

オントロジーヘッダーの `rdf:about` 属性には、通常、

```
rdf:about=""
```

というように空文字列を設定します。この場合の空文字列は、OWL 文書自身の基底 URI を意味しています。

Q 3.3.3 OWL 文書自身についての言明って、どんなふうに書けばいいんですか。

A OWL 文書自身についての言明は、オントロジーヘッダーの中にプロパティー要素として書きます。

Q 3.3.2 で説明したように、オントロジーヘッダーというのは、

> この文書は OWL 文書である。

という言明を記述した型付ノード要素のことです。ということは、オントロジーヘッダーの子供としてプロパティー要素を書くことによって、

　この文書の何々は何々である。

という言明を記述できるということになります。

Q 3.3.4　注記プロパティーって何ですか。

A　「注記プロパティー」(annotation property) というのは、OWL オントロジーの中で注記として使われるプロパティーのことです。

　注記プロパティーは、基本的にはオントロジーヘッダーの中で使うものですが、OWL オントロジーのそれ以外の場所でも使うことができます。
　OWL は、次の五つのプロパティーを注記プロパティーとして定義しています。

- `owl:versionInfo`
- `rdfs:label`
- `rdfs:comment`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`

　次の例は、注記を記述したオントロジーヘッダーです。

```
<owl:Ontology rdf:about="">
  <rdfs:comment>野菜オントロジー</rdfs:comment>
</owl:Ontology>
```

Q 3.3.5　注記プロパティーとして OWL が定義している五つ以外のプロパティーを注記プロパティーとして使うことは可能ですか。

A　はい、可能です。

　注記プロパティーとして使うことのできるプロパティーは、OWL が注記プロパティーとして定義している五つだけではありません。たとえば、`dc:creator` を注記プロパティーとして使うことも可能です。
　ただし、注記プロパティーとして OWL が定義していないプロパティーを注記プロパティーとして使うためには、それが注記プロパティーだということを記述しておく必要があります。
　注記プロパティーというのは、`owl:AnnotationProperty` というクラスのインスタンスです。ですから、

　何々は注記プロパティーである。

という言明は、

```
<owl:AnnotationProperty rdf:about="何々"/>
```

という型付ノード要素によって記述することができます。
　ですから、

　`dc:creator` は注記プロパティーである。

という言明を記述したもの、すなわち、

```
<owl:AnnotationProperty
    rdf:about="http://purl.org/dc/elements/1.1/creator"/>
```

という型付ノード要素を書くことによって、

```
<owl:Ontology rdf:about="">
  <dc:creator rdf:resource="http://example.org/staff/nanako"/>
</owl:Ontology>
```

というような、`dc:creator` を注記プロパティーとして使ったオントロジーヘッダーを書くことが可能になります。

Q 3.3.6　OWL 文書を取り込むって、どういうことですか。

A　OWL 文書を「取り込む」(import) というのは、OWL 文書が、指定された OWL 文書を参照して、それが記述しているオントロジーを、自分が記述しているオントロジーの一部分にする、ということです。

Q 3.2.16 で説明したように、OWL は、ひとつの主題に関するオントロジーをウェブ上で分散させることができるようにするという方針で設計されています。それを実現するために、OWL は、ウェブ上に分散して存在している OWL 文書を取り込むことができるという機能を持っています。

Q 3.3.7　OWL 文書を取り込みたいときって、どうすればいいんですか。

A　`owl:imports` というプロパティーを使うことによって、OWL 文書を取り込むことができます。

OWL 文書を取り込みたいときは、`owl:imports` を述語、取り込みたい OWL 文書の URI を目的語とするプロパティー要素をオントロジーヘッダーの子供として書きます。たとえば、

```
http://www.example.org/owl/common.owl
```

という OWL 文書を取り込みたいときは、

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://www.example.org/owl/common.owl"/>
</owl:Ontology>
```

というオントロジーヘッダーを書けばいいわけです。

`owl:imports` の機能は推移的です。ですから、A、B、C が OWL 文書だとするとき、A が B を取り込んでいて、B が C を取り込んでいるとするならば、A は、B だけではなくて C も取り込むことになります。

Q 3.3.8　バージョン情報って何ですか。

A　「バージョン情報」(version information) というのは、バージョンについての情報のことです。

オントロジーヘッダーの子供として書くプロパティー要素としては、注記と取り込みを記述するもののほかに、バージョン情報を記述するものを書くこともできます。OWL は、バージョン情報を記述するためのプロパティーとして、次のような語彙を定義しています。

`owl:versionInfo`	バージョンをあらわす文字列を記述する。注記プロパティーである。オントロジーヘッダー以外の場所で使ってもよい。
`owl:priorVersion`	以前のバージョンの `OWL` 文書を参照する `URI` を記述する。
`owl:backwardCompatibleWith`	以前のバージョンの `OWL` 文書を参照する `URI` を記述して、それに対して下位互換性があることを示す。
`owl:incompatibleWith`	以前のバージョンの `OWL` 文書を参照する `URI` を記述して、それに対して下位互換性がないことを示す。

また、オントロジーヘッダーの中で使うものではありませんが、OWL は、バージョン情報を記述するために次のようなクラスを定義しています。

`owl:DeprecatedClass`	非推奨のクラス。
`owl:DeprecatedProperty`	非推奨のプロパティー。

これらのクラスは、将来的には廃止する予定の語彙を、下位互換性を維持するために残しておく場合に使われるものです。

第 4 章　クラス

4.1　クラスの基礎

Q 4.1.1　クラスの外延って何のことですか。

A　クラスのインスタンスの集合を、そのクラスの「外延」(extension) と呼びます。

Q 4.1.2　クラスの内包って何のことですか。

A　クラスが意味している概念を、そのクラスの「内包」(intension) と呼びます。

内包が同一である二つのクラスは同じものだと考えることができるのに対して、外延が同一である二つのクラスは、かならずしも同じものだとは言えません。外延が同一であっても、内包が異なっているならば、それらの二つのクラスは異なる別々のものです。

Q 4.1.3　OWL では、「クラス」という概念はどのようなクラスによってあらわされるんですか。

A　OWL では、`owl:Class` というクラスによって「クラス」という概念をあらわします。

`owl:Class` は、`rdfs:Class` のサブクラスです。

OWL において、`rdfs:Class` とは別に `owl:Class` というクラスが定義されている理由は、OWL Lite と OWL DL には型分離という制限が加えられているためです。すなわち、OWL Lite と OWL DL においては、`owl:Class` の外延は、クラスであると同時にプロパティーであるようなリソースや、クラスであると同時に個体であるようなリソースを含んでいないのです。

`rdfs:Class` というクラスには型分離という制限が加えられていませんので、その外延は、OWL Lite と OWL DL においては認められていないクラスをも含んでいます。

OWL Full においては、`owl:Class` と `rdfs:Class` とは等価なクラスだとみなすことができます。

Q 4.1.4　`owl:Thing` って何ですか。

A　`owl:Thing` は、「個体」をあらわすクラスです。

`owl:Thing` の外延は、すべての個体の集合です。また、任意の OWL のクラスは、`owl:Thing` のサブクラスです。

Q 4.1.5　`owl:Nothing` って何ですか。

A　`owl:Nothing` は、「存在しないもの」をあらわすクラスです。

`owl:Nothing` の外延は、空集合です。また、`owl:Nothing` は、任意の OWL のクラスのサブクラスです。

Q 4.1.6　クラス公理って何ですか。

A　「クラス公理」(class axiom) というのは、クラスを定義する記述のことです。

クラス公理については、第 4.4 節で、もう少し詳しく説明したいと思います。

Q 4.1.7　クラス記述って何ですか。

A　「クラス記述」(class description) というのは、クラス公理を構成する基本的な記述の単位のことです。

クラス記述には、次の 6 個のタイプがあります。

(1) クラス名の記述。
(2) クラスのすべてのインスタンスの列挙。
(3) プロパティー制約。
(4) ふたつ以上のクラス記述の積集合。
(5) ふたつ以上のクラス記述の和集合。
(6) クラス記述の補集合。

クラス名の記述については Q 4.1.8 で、インスタンスの列挙については Q 4.1.9 で、プロパティー制約については第 4.2 節で、積集合、和集合、補集合については第 4.3 節で説明したいと思います。

Q 4.1.8　クラス名の記述っていうタイプのクラス記述って、どんなものなんですか。

A クラス名の記述というタイプのクラス記述というのは、断片識別子によって識別されるリソースがクラスであると主張する記述のことです。

クラスに割り当てられた断片識別子は、「クラス名」(class identifier) と呼ばれます。
たとえば、 Movie というクラス名の記述は、

```
<owl:Class rdf:ID="Movie"/>
```

と書くことができます。
なお、クラスであると主張されたリソースは、自動的に owl:Thing のサブクラスになります。

Q 4.1.9 インスタンスの列挙っていうタイプのクラス記述って、どんなものなんですか。

A インスタンスの列挙というタイプのクラス記述というのは、クラスのすべてのインスタンスを列挙する記述のことです。

インスタンスの列挙が使えるのは、OWL Full と OWL DL だけです。OWL Lite では使えません。
インスタンスの列挙には、 owl:oneOf というプロパティーを使います。このプロパティーは、定義域が owl:Class で、値域は、コレクションをあらわす rdf:List というクラスです。
インスタンスの列挙は、その全体がひとつの owl:Class 要素です。そして、その子供として owl:oneOf 要素を書いて、さらにその子供として、それぞれのインスタンスの記述を書きます。
owl:oneOf 要素は、すべてのインスタンスを網羅しているということを示すために、コレクションとして記述する必要があります。つまり、 owl:oneOf 要素は、

```
<owl:oneOf rdf:parseType="Collection">
</owl:oneOf>
```

と書かないといけない、ということです。
それぞれのインスタンスは、

何々は個体である。

という主張をあらわす型付ノード要素、という形で記述します。OWL では、「個体」という概念は owl:Thing というクラスであらわされますので、インスタンスの記述は、

```
<owl:Thing rdf:about="何々"/>
```

と書けばいい、ということになります。
それでは、例として、「地球型惑星」というクラスのクラス記述を、インスタンスの列挙で書いてみましょう。水星、金星、地球、火星のそれぞれを、 Mercury 、 Venus 、 Earth 、 Mars という断片識別子であらわすとすると、地球型惑星のクラス記述は、

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Mercury"/>
    <owl:Thing rdf:about="#Venus"/>
    <owl:Thing rdf:about="#Earth"/>
    <owl:Thing rdf:about="#Mars"/>
  </owl:oneOf>
</owl:Class>
```

と書くことができます。

4.2 プロパティー制約

Q 4.2.1 プロパティー制約っていうタイプのクラス記述って、どんなものなんですか。

A 「プロパティー制約」(property restriction) というタイプのクラス記述というのは、プロパティーに対する何らかの制約の記述のことです。

プロパティー制約は、 owl:Restriction という要素型の要素として書きます。
owl:Restriction は、OWL で定義されている、「制約」(restriction) という概念をあらわすクラスで、 owl:Class のサブクラスです。

制約の対象となるプロパティーは、owl:onProperty というプロパティーを使って記述します。このプロパティーは、定義域が owl:Restriction で、値域が rdf:Property です。

プロパティー制約は、それらのクラスやプロパティーを使って、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="URI"/>
  制約の記述
</owl:Restriction>
```

という形で記述します。この中の「URI」というところには、制約の対象となるプロパティーを識別する URI を書きます。そして、「制約の記述」というところには、どのような制約なのかということをあらわすプロパティー要素を 1 個だけ書きます。

プロパティーに対する制約は、「値制約」(value constraint) と「個数制約」(cardinality constraint) の 2 種類に分類することができます。

Q 4.2.2 値制約って、どういう制約なんですか。

A 値制約というのは、プロパティーの値が何らかの条件を満足する必要があるという制約のことです。

値制約は、次の 3 個のプロパティーのいずれかを使って記述します。

プロパティー名	定義域	値域
owl:allValuesFrom	owl:Restriction	rdfs:Class
owl:someValuesFrom	owl:Restriction	rdfs:Class
owl:hasValue	owl:Restriction	指定なし

Q 4.2.3 owl:allValuesFrom って、どういう制約なんですか。

A owl:allValuesFrom は、指定されたプロパティーの値が、すべて、目的語として指定されたクラスのインスタンスであるか、または目的語として指定されたデータ型のデータ値でなければならない、という制約です。

この制約を使うことによって、

何々がすべて何々である個体のクラス

という匿名のクラスを記述することができます。たとえば、hasChild が「誰々を子供として持つ」というプロパティー、Female が「女性」というクラスをあらわしているとするとき、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:allValuesFrom rdf:resource="#Female"/>
</owl:Restriction>
```

という owl:Restriction 要素を書くことによって、

子供がすべて女性である個体のクラス

という匿名のクラスを記述することができます。

owl:allValuesFrom の機能は、述語論理で使われる全称限量子 (universal quantifier) に類似するものだと考えることができます。

OWL Lite では、owl:allValuesFrom の目的語として許されるのは、クラス名によるクラスの記述だけです。

Q 4.2.4 owl:someValuesFrom って、どういう制約なんですか。

A owl:someValuesFrom は、指定されたプロパティーの値のうちの少なくともひとつが、目的語として指定されたクラスのインスタンスであるか、または目的語として指定されたデータ型のデータ値でなければならない、という制約です。

この制約を使うことによって、

何々のうちの少なくともひとつが何々である個体のクラス

という匿名のクラスを記述することができます。たとえば、hasChildが「誰々を子供として持つ」というプロパティー、Femaleが「女性」というクラスをあらわしているとするとき、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:someValuesFrom rdf:resource="#Female"/>
</owl:Restriction>
```

というowl:Restriction要素を書くことによって、

子供のうちの少なくとも一人が女性である個体のクラス

という匿名のクラスを記述することができます。

owl:someValuesFromの機能は、述語論理で使われる存在限量子 (existential quantifier) に類似するものだと考えることができます。

OWL Liteでは、owl:someValuesFromの目的語として許されるのは、クラス名によるクラスの記述だけです。

Q 4.2.5 owl:hasValueって、どういう制約なんですか。

A owl:hasValueは、指定されたプロパティーの値のひとつが、目的語として指定された特定の個体またはデータ値でなければならない、という制約です。

この制約を使うことによって、

何々のひとつが特定の何々である個体のクラス

という匿名のクラスを記述することができます。たとえば、hasChildが「誰々を子供として持つ」というプロパティー、Naokoが「奈緒子」という個体をあらわしているとするとき、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:hasValue rdf:resource="#Naoko"/>
</owl:Restriction>
```

というowl:Restriction要素を書くことによって、

奈緒子を子供として持つ個体のクラス

という匿名のクラスを記述することができます。

owl:hasValueは、OWL Liteには含まれていません。

Q 4.2.6 個数制約って、どういう制約なんですか。

A 個数制約というのは、プロパティーの値の個数に関する制約のことです。

値の個数というのは、ひとつの個体とひとつのプロパティーについて、その個体を主語として、そのプロパティーを述語とする三つ組みのうちで、目的語が異なっているものの個数のことです。たとえば、「昌代」という個体と、「誰々を子供として持つ」というプロパティーについて、その個体を主語として、そのプロパティーを述語とする三つ組みが、

昌代は智子を子供として持つ。
昌代は道雄を子供として持つ。
昌代は静子を子供として持つ。

という3個だけ存在する場合、値の個数は3ということになります。

個数制約は、次の3個のプロパティーのいずれかを使って記述します。

プロパティー名	定義域	値域
owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger
owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger
owl:cardinality	owl:Restriction	xsd:nonNegativeInteger

Q 4.2.7 owl:maxCardinality って、どういう制約なんですか。

A owl:maxCardinality は、プロパティーの値の個数が、目的語として指定されたマイナスではない整数を超えないという制約です。

この制約を使うことによって、

何々が何個以下である個体のクラス

という匿名のクラスを記述することができます。たとえば、hasChild が「誰々を子供として持つ」というプロパティーをあらわしているとするとき、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
    3
  </owl:maxCardinality>
</owl:Restriction>
```

という owl:Restriction 要素を書くことによって、

子供が 3 人以下である個体のクラス

という匿名のクラスを記述することができます。

owl:maxCardinality が 1 であるという制約が与えられたプロパティーは、「関数型プロパティー」(functional property) と呼ばれます。

OWL Lite では、owl:maxCardinality の目的語として許されるのは、0 または 1 のいずれかだけです。

Q 4.2.8 owl:minCardinality って、どういう制約なんですか。

A owl:minCardinality は、プロパティーの値の個数が、目的語として指定されたマイナスではない整数を下回らないという制約です。

この制約を使うことによって、

何々が何個以上である個体のクラス

という匿名のクラスを記述することができます。たとえば、hasChild が「誰々を子供として持つ」というプロパティーをあらわしているとするとき、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
    3
  </owl:minCardinality>
</owl:Restriction>
```

という owl:Restriction 要素を書くことによって、

子供が 3 人以上である個体のクラス

という匿名のクラスを記述することができます。

OWL Lite では、owl:minCardinality の目的語として許されるのは、0 または 1 のいずれかだけです。

Q 4.2.9 owl:cardinality って、どういう制約なんですか。

A owl:cardinality は、プロパティーの値の個数が、目的語として指定されたマイナスではない整数と等しいという制約です。

この制約を使うことによって、

何々がちょうど何個である個体のクラス

という匿名のクラスを記述することができます。たとえば、hasChild が「誰々を子供として持つ」というプロパティーをあらわしているとするとき、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
    3
  </owl:cardinality>
</owl:Restriction>
```

という owl:Restriction 要素を書くことによって、

子供がちょうど 3 人である個体のクラス

という匿名のクラスを記述することができます。
OWL Lite では、owl:cardinality の目的語として許されるのは、0 または 1 のいずれかだけです。

4.3 積集合と和集合と補集合

Q 4.3.1 積集合っていうタイプのクラス記述って、どんなものなんですか。

A 「積集合」(intersection) というタイプのクラス記述というのは、ふたつ以上のクラスを列挙することによって、それらのクラスの外延に共通して含まれる個体から構成される集合を外延とするクラスをあらわす記述のことです。

積集合は、owl:intersectionOf というプロパティーを使って記述します。このプロパティーは、定義域が owl:Class で、値域が rdf:List です。
積集合のクラス記述は、owl:Class 要素の子供として owl:intersectionOf 要素を書いたものです。そして、その子供として、クラス記述、またはクラスを参照する owl:Class 要素を、コレクションとして列挙します。そうすると、そのコレクションで列挙されたクラスの外延に共通して含まれる個体から構成される集合を外延とするクラスを記述したことになります。
積集合を使うことによって、

何々かつ何々かつ……かつ何々であるクラス

という匿名のクラスを記述することができます。たとえば、Student が「学生」というクラス、Worker が「労働者」というクラスをあらわしているとするとき、

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Student"/>
    <owl:Class rdf:about="#Worker"/>
  </owl:intersectionOf>
</owl:Class>
```

という owl:Class 要素を書くことによって、

学生でかつ労働者であるクラス

という匿名のクラスを記述することができます。

Q 4.3.2 和集合っていうタイプのクラス記述って、どんなものなんですか。

A 「和集合」(union) というタイプのクラス記述というのは、ふたつ以上のクラスを列挙することによって、それらのクラスの外延のうちのいずれかに含まれる個体から構成される集合を外延とするクラスをあらわす記述のことです。

和集合は、owl:unionOf というプロパティーを使って記述します。このプロパティーは、定義域が owl:Class で、値域が rdf:List です。
和集合のクラス記述は、owl:Class 要素の子供として owl:unionOf 要素を書いたものです。そして、その子供として、クラス記述、またはクラスを参照する owl:Class 要素を、コレクションとして列挙します。そうすると、そのコレクションで列挙されたクラスの外延のうちのいずれかに含まれる個体から構成される集合を外延とするクラスを記述したことになります。
和集合を使うことによって、

何々または何々または……または何々であるクラス

という匿名のクラスを記述することができます。たとえば、Student が「学生」というクラス、Worker が「労働者」というクラスをあらわしているとするとき、

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Student"/>
    <owl:Class rdf:about="#Worker"/>
  </owl:unionOf>
</owl:Class>
```

という owl:Class 要素を書くことによって、

学生かまたは労働者であるクラス

という匿名のクラスを記述することができます。

owl:unionOf は、OWL Lite には含まれていません。

Q 4.3.3 補集合っていうタイプのクラス記述って、どんなものなんですか。

A 「補集合」(complement) というタイプのクラス記述というのは、ひとつのクラスの外延に含まれない個体から構成される集合を外延とするクラスをあらわす記述のことです。

補集合は、owl:complementOf というプロパティーを使って記述します。このプロパティーは、定義域も値域も owl:Class です。

補集合のクラス記述は、owl:Class 要素の子供として owl:complementOf 要素を書いたものです。そして、その子供として、クラス記述、またはクラスを参照する owl:Class 要素を 1 個だけ書きます。そうすると、そのクラスの外延に含まれない個体から構成される集合を外延とするクラスを記述したことになります。

補集合を使うことによって、

何々ではないクラス

という匿名のクラスを記述することができます。たとえば、Student が「学生」というクラスをあらわしているとするとき、

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Student"/>
  </owl:complementOf>
</owl:Class>
```

という owl:Class 要素を書くことによって、

学生ではないクラス

という匿名のクラスを記述することができます。

ちなみに、「学生ではないクラス」の外延は、人間だけではなく、人間以外の動物や、無生物や、抽象概念なども含んでいます。もしも、

学生ではない人間のクラス

を記述したいとするならば、補集合の記述と積集合の記述とを組み合わせる必要があります。つまり、Student が「学生」というクラス、Human が「人間」というクラスをあらわしているとすれば、「学生ではない人間のクラス」は、

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:complementOf>
        <owl:Class rdf:about="#Student"/>
      </owl:complementOf>
    </owl:Class>
    <owl:Class rdf:about="#Human"/>
  </owl:intersectionOf>
</owl:Class>
```

と記述すればいい、ということです。

owl:complementOf は、OWL Lite には含まれていません。

4.4 クラス公理

Q 4.4.1 クラス公理って、どういう記述なんですか。

A クラス公理は、rdf:ID 属性に名前が設定されるか、または rdf:about 属性に URI が設定された owl:Class 要素です。

したがって、もっとも単純なクラス公理は、

```
<owl:Class rdf:ID="Movie"/>
```

のような、クラス名の記述というタイプのクラス記述です。

インスタンスの列挙、積集合、和集合、補集合も、owl:Class 要素の rdf:ID 属性に名前を設定することによって、クラス公理になります。たとえば、

```
<owl:Class rdf:ID="EarthTypePlanet">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Mercury"/>
    <owl:Thing rdf:about="#Venus"/>
    <owl:Thing rdf:about="#Earth"/>
    <owl:Thing rdf:about="#Mars"/>
  </owl:oneOf>
</owl:Class>
```

というクラス記述は、owl:Class 要素の rdf:ID 属性に名前が設定されていますので、クラス公理になります。

以上のようなクラス公理のほかに、rdf:ID 属性に名前が設定された owl:Class 要素の子供として、次のプロパティーを使ったプロパティー要素を書いたものも、クラス公理になります。

プロパティー名	定義域	値域
rdfs:subClassOf	rdfs:Class	rdfs:Class
owl:equivalentClass	owl:Class	owl:Class
owl:disjointWith	owl:Class	owl:Class

Q 4.4.2 サブクラス公理って何ですか。

A 「サブクラス公理」(subclass axioms) というのは、rdfs:subClassOf というプロパティーを使ったプロパティー要素を子供として持つクラス公理のことです。

rdfs:subClassOf は、Q 2.3.3 で説明したように、「主語は目的語のサブクラスである」ということを意味するプロパティーです。

たとえば、「男声合唱団 (MaleChoir) は合唱団 (Choir) のサブクラスである」という主張を記述した、

```
<owl:Class rdf:ID="MaleChoir">
  <rdfs:subClassOf rdf:resource="#Choir"/>
</owl:Class>
```

というクラス公理は、サブクラス公理です。

ひとつのクラスについて、2 個以上のサブクラス公理が存在していてもかまいません。ですから、MaleChoir クラスについてのサブクラス公理として、上に書いたサブクラス公理のほかに、「男声合唱団 (MaleChoir) は、メンバー (hasMember) がすべて男性 (Male) である個体のクラスのサブクラスである」という主張を記述した、

```
<owl:Class rdf:about="#MaleChoir">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
      <owl:allValuesFrom rdf:resource="#Male"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

というサブクラス公理が存在していてもかまいません。

サブクラス公理は、個体がクラスの外延に含まれるための必要条件を示してはいますが、それは十分条件ではありません。ですから、サブクラス公理は、クラスの定義としては不完全です。

OWL Lite で rdfs:subClassOf を使う場合、その主語はクラス名でないといけません。そして、その目的語は、クラス名またはプロパティー制約のどちらかでないといけません。

Q 4.4.3 等価クラス公理って何ですか。

A 「等価クラス公理」(equivalent class axiom) というのは、owl:equivalentClass というプロパティーを使ったプロパティー要素を子供として持つクラス公理のことです。

owl:equivalentClass は、「主語は目的語と同じ外延を持つクラスである」ということを意味するプロパティーです。

たとえば、Hoge と Munya がクラスだとするとき、

```
<owl:Class rdf:about="#Hoge">
  <owl:equivalentClass rdf:resource="#Munya"/>
</owl:Class>
```

という等価クラス公理を書くことによって、Hoge の外延と Munya の外延とが同一の集合だということを記述することができます。

owl:equivalentClass は、クラスの外延が同一だということをあらわしているだけで、クラスの内包が同一だということまであらわしているわけではありません。つまり、等価クラス公理は、二つのクラスが同一であるということを意味しているわけではないのです。ですから、上の例は、Hoge と Munya とが同一のクラスであるという意味ではありません。

OWL Full では、owl:sameAs というプロパティーを使うことによって、二つのクラスが同一であるということを記述することができます。OWL DL と OWL Lite では、それを記述することはできません。

OWL DL と OWL Lite で、二つのクラスが同一であるということを記述することができない理由は、owl:sameAs が、「主語と述語とは同一の個体である」ということをあらわすプロパティーだからです。つまり、このプロパティーを使って二つのクラスが同一であるということを記述するためには、クラスを個体として扱うことができないといけないのです。

OWL Lite で owl:equivalentClass を使う場合、その主語はクラス名でないといけません。そして、その目的語は、クラス名またはプロパティー制約のどちらかでないといけません。

Q 4.4.4 owl:Restriction 要素の rdf:ID 属性に名前を設定したものって、クラス公理になるんですか。

A いいえ、owl:Restriction 要素自体をクラス公理にする、ということはできません。

しかし、owl:equivalentClass を使うことによって、プロパティー制約をクラス公理にすることも可能です。たとえば、

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:someValuesFrom rdf:resource="#Female"/>
</owl:Restriction>
```

というプロパティー制約は、

```
<owl:Class rdf:ID="ParentOfDaughter">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:someValuesFrom rdf:resource="#Female"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

というように、owl:equivalentClass を使うことによって、クラス公理にすることができます。

Q 4.4.5 排他クラス公理って何ですか。

A 「排他クラス公理」(disjoint class axiom) というのは、owl:disjointWith というプロパティーを使ったプロパティー要素を子供として持つクラス公理のことです。

owl:disjointWith は、「主語の外延と目的語の外延とは互いに素である」ということを意味するプロパティーです。「外延が互いに素である」というのは、両方の外延に共通する個体は存在しない、ということです。

たとえば、Angel が「天使」というクラス、Devil が「悪魔」というクラスをあらわしているとするとき、

```
<owl:Class rdf:about="#Angel">
  <owl:disjointWith rdf:resource="#Devel"/>
</owl:Class>
```

という排他クラス公理を書くことによって、「天使であってかつ悪魔である個体は存在しない」ということを記述することができます。

サブクラス公理と同じように、排他クラス公理も、記述しているのはクラスの必要条件であって、十分条件ではありません。ですから、排他クラス公理も、クラスの定義としては不完全です。

owl:disjointWith は、OWL Lite には含まれていません。

第 5 章　プロパティー

5.1　プロパティーの基礎

Q 5.1.1　プロパティーのインスタンスって何のことですか。

A　プロパティーのインスタンスというのは、そのプロパティーを使って主張される文の主語と目的語から構成されるペアのことです。

P がプロパティーだとするとき、

x の P は y である。

という文が主張されているとするならば、

(x, y)

というペアは、P のインスタンスです。

たとえば、「誰々を子供として持つ」というプロパティーがあって、それを使って、

昌代は智子を子供として持つ。

という文が主張されているとするならば、

(昌代, 智子)

というペアは、そのプロパティーのインスタンスになります。

Q 5.1.2　プロパティーの外延って何のことですか。

A　プロパティーの外延というのは、そのプロパティーのインスタンスの集合のことです。

Q 5.1.3　プロパティー公理って何ですか。

A　「プロパティー公理」(property axiom) というのは、プロパティーを定義する記述のことです。

ひとつのプロパティー公理は、プロパティーが持っている性質のひとつについて記述したものです。プロパティーの性質としては、次のようなものがあります。

(1) 定義域。
(2) 値域。
(3) ほかのプロパティーとの関係。
(4) 大域的個数制約。
(5) 論理的な性質。

定義域については Q 5.1.5 で、値域については Q 5.1.6 で、ほかのプロパティーとの関係については第 5.2 節で、大域的個数制約については第 5.3 節で、論理的な性質については第 5.4 節で説明したいと思います。

Q 5.1.4 プロパティー公理は、rdf:Propertyというクラスを使って書けばいいんですか。

A いいえ、プロパティー公理は、rdf:Propertyのサブクラスを使って書きます。

OWLでは、プロパティーは、オブジェクトプロパティーとデータ型プロパティーの2種類に分類されます。

「オブジェクトプロパティー」(object property)というのは、個体を個体に結び付けるプロパティー、つまり、URIで識別されるノードまたは空白ノードを目的語とするプロパティーのことです。

「データ型プロパティー」(datatype property)というのは、個体をデータ値に結び付けるプロパティー、つまり、リテラルを目的語とするプロパティーのことです。

OWLでは、プロパティーは、それがオブジェクトプロパティーなのかデータ型プロパティーなのかという種類に応じて、rdf:Propertyのサブクラスとして定義されている次の二つのクラスのどちらかのインスタンスとして定義されます。

owl:ObjectProperty	オブジェクトプロパティー。
owl:DatatypeProperty	データ型プロパティー。

プロパティー公理というのは、これらの二つのクラス、あるいはこれらのクラスのサブクラスの名前を要素型とする型付ノード要素のことです。ですから、

```
<owl:ObjectProperty rdf:ID="hasMember"/>
```

という型付ノード要素は、プロパティー公理の一例です。このプロパティー公理は、hasMember(誰々をメンバーとして持つ)というのがオブジェクトプロパティーだということを主張しています。

Q 5.1.5 定義域について記述するプロパティー公理って、どんなふうに書けばいいんですか。

A 定義域について記述するプロパティー公理は、rdfs:domainというプロパティーを使って書きます。

rdfs:domainというのは、Q 2.4.4で説明したように、「主語の定義域は目的語である」ということをあらわすプロパティーで、その定義域はrdf:Propertyで、その値域はrdfs:Classです。

たとえば、Groupが「グループ」というクラスをあらわしているとするとき、

```
<owl:ObjectProperty rdf:ID="hasMember">
  <rdfs:domain rdf:resource="#Group"/>
</owl:ObjectProperty>
```

というプロパティー公理を書くことによって、hasMember(誰々をメンバーとして持つ)というのはオブジェクトプロパティーで、その定義域はグループだということを記述することができます。

OWL Liteでは、rdfs:domainの値はクラス名でないといけません。それに対して、OWL DLとOWL Fullでは、rdfs:domainの値をクラス記述で書くこともできます。ですから、owl:unionOfを使うことによって、複数の集合の和集合の形で定義域を記述することも可能です。

たとえば、Movieが「映画」というクラスをあらわしていて、Dramaが「ドラマ」というクラスをあらわしているとするとき、

```
<owl:ObjectProperty rdf:ID="original">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Movie"/>
        <owl:Class rdf:about="#Drama"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

というプロパティー公理を書くことによって、original(原作)というのはオブジェクトプロパティーで、その定義域は映画とドラマの和集合だということを記述することができます。

Q 5.1.6 値域について記述するプロパティー公理って、どんなふうに書けばいいんですか。

A 値域について記述するプロパティー公理は、rdfs:rangeというプロパティーを使って書きます。

rdfs:rangeというのは、Q 2.4.5で説明したように、「主語の値域は目的語である」ということをあらわすプロパティーで、その定義域はrdf:Propertyで、その値域はrdfs:Classです。
たとえば、Humanが「人間」というクラスをあらわしているとするとき、

```
<owl:ObjectProperty rdf:ID="hasMember">
  <rdfs:range rdf:resource="#Human"/>
</owl:ObjectProperty>
```

というプロパティー公理を書くことによって、hasMember（誰々をメンバーとして持つ）というのはオブジェクトプロパティーで、その値域は人間だということを記述することができます。
同じように、

```
<owl:DatatypeProperty rdf:about="#dateOfBirth">
  <rdf:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>
```

というプロパティー公理を書くことによって、dateOfBirth（誕生日の日付）というのはデータ型プロパティーで、その値域は日付だということを記述することができます。
OWL Liteでは、オブジェクトプロパティーのrdfs:rangeの値は、クラス名でないといけません。それに対して、OWL DLとOWL Fullでは、オブジェクトプロパティーのrdfs:rangeの値をクラス記述で書くこともできます。ですから、rdfs:domainの場合と同じように、owl:unionOfを使うことによって、複数の集合の和集合の形で値域を記述することも可能です。
また、OWL Liteでは、データ型プロパティーのrdfs:rangeの値として使うことができるのは、データ型またはrdfs:Literalだけです。それに対して、OWL DLとOWL Fullでは、データ型プロパティーのrdfs:rangeの値として列挙データ型[1]を使うこともできます。

5.2 ほかのプロパティーとの関係

Q 5.2.1 プロパティー公理で記述される、ほかのプロパティーとの関係って、どんな関係のことなんですか。

A プロパティー公理によって記述される、ほかのプロパティーとの関係には、サブプロパティー関係、等価関係、逆関係という3種類のものがあります。

Q 5.2.2 サブプロパティー関係って、どういう関係なんですか。

A 「サブプロパティー関係」(subproperty relation)というのは、プロパティーがほかのプロパティーのサブプロパティーになっているという関係のことです。

サブプロパティーとは何かということについては、Q 2.4.7を参照してください。

Q 5.2.3 サブプロパティー関係について記述するプロパティー公理って、どんなふうに書けばいいんですか。

A サブプロパティー関係について記述するプロパティー公理は、rdfs:subPropertyOfというプロパティーを使って書きます。

rdfs:subPropertyOfというのは、Q 2.4.8で説明したように、「主語は目的語のサブプロパティーである」ということをあらわすプロパティーで、その定義域と値域は、ともにrdf:Propertyです。
たとえば、

```
<owl:ObjectProperty rdf:ID="hasMaleChild">
  <rdfs:subPropertyOf rdf:resource="#hasChild"/>
</owl:ObjectProperty>
```

というプロパティー公理を書くことによって、hasMaleChild（誰々を男の子供として持つ）というのはオブジェクトプロパティーで、それはhasChild（誰々を子供として持つ）のサブプロパティーだということを記述することができます。

[1]列挙データ型については、[OWLReference,2004], 6.2を参照してください。

OWL Lite と OWL DL では、rdfs:subPropertyOf を使う場合、主語がオブジェクトプロパティーならば述語もオブジェクトプロパティー、主語がデータ型プロパティーならば述語もデータ型プロパティーでないといけません。

Q 5.2.4 等価関係って、どういう関係なんですか。

A 「等価関係」(equivalence relation) というのは、プロパティーとプロパティーとが同一の外延を持っているという関係のことです。

二つのプロパティーが等価関係を持っているというのは、単に、それらの外延が同一だというだけのことであって、それらが同一のプロパティーだということを意味しているわけではありません。二つのプロパティーが同一だということを記述したいときは、owl:sameAs というプロパティーを使う必要があります。

ただし、Q 4.4.3 で説明したように、owl:sameAs は、個体の同一性をあらわすプロパティーです。ですから、このプロパティーを使ってプロパティーの同一性を記述することができるのは、プロパティーを個体として扱うことができる OWL Full に限定されます。

Q 5.2.5 等価関係について記述するプロパティー公理って、どんなふうに書けばいいんですか。

A 等価関係について記述するプロパティー公理は、owl:equivalentProperty というプロパティーを使って書きます。

owl:equivalentProperty というのは、「主語は目的語と等価関係にある」ということをあらわすプロパティーで、その定義域と値域は、ともに rdf:Property です。

たとえば、

```
<owl:ObjectProperty rdf:ID="hoge">
  <owl:equivalentProperty rdf:resource="#munya"/>
</owl:ObjectProperty>
```

というプロパティー公理を書くことによって、hoge というのはオブジェクトプロパティーで、それは munya と等価関係にあるプロパティーだということを記述することができます。

Q 5.2.6 逆関係って、どういう関係なんですか。

A 「逆関係」(inverse relation) というのは、プロパティーがほかのプロパティーの逆になっているという関係のことです。

「逆」(inverse) というのは、インスタンスの主語と目的語とを入れ替えることによってできる集合を外延とするプロパティーのことです。たとえば、「誰々を子供として持つ」というプロパティーは、「誰々を親として持つ」というプロパティーの逆になります。

Q 5.2.7 逆関係について記述するプロパティー公理って、どんなふうに書けばいいんですか。

A 逆関係について記述するプロパティー公理は、owl:inverseOf というプロパティーを使って書きます。

owl:inverseOf というのは、「主語は目的語の逆である」ということをあらわすプロパティーで、その定義域と値域は、ともに owl:ObjectProperty です。

たとえば、

```
<owl:ObjectProperty rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

というプロパティー公理を書くことによって、hasChild（誰々を子供として持つ）というのはオブジェクトプロパティーで、それは hasParent（誰々を親として持つ）の逆だということを記述することができます。

5.3 大域的個数制約

Q 5.3.1 大域的個数制約って、プロパティーのどういう性質のことなんですか。

A 「大域的個数制約」(global cardinality constraint) というのは、主語または目的語の個数に関する大域的な制約のことです。

大域的個数制約が「大域的」(global) である理由は、その制約が与えられたプロパティーは、いかなる場合でもその制約が有効だからです。

それに対して、`owl:maxCardinality` などによって記述される制約は、プロパティーが特定のクラスに対して適用された場合にだけ有効ですので、「大域的」ではありません。

大域的個数制約には、関数的と逆関数的という二つの制約があります。関数的という制約については Q 5.3.2 で、逆関数的という制約については Q 5.3.4 で説明したいと思います。

Q 5.3.2 関数的って、どういう制約なんですか。

A 「関数的」(functional) というのは、特定の主語に対応する目的語は一意に定まる、という制約のことです。

関数的という制約は、オブジェクトプロパティーとデータ型プロパティーのどちらにも与えることができます。

関数的という制約を与えられたプロパティーは、「関数的プロパティー」(functional property) と呼ばれます。

P が関数的プロパティーだとするとき、(x, y_1) というペアが P のインスタンスとして存在するならば、y_1 とは異なる y_2 という個体またはデータ値を目的語とする (x, y_2) というペアが P のインスタンスとして存在するということはあり得ません。

Q 5.3.3 プロパティーが関数的だということを記述するプロパティー公理って、どんなふうに書けばいいんですか。

A `owl:FunctionalProperty` というクラスを使うことによって、プロパティーが関数的だということを記述するプロパティー公理を書くことができます。

`owl:FunctionalProperty` というのは、「関数的プロパティー」をあらわすクラスです。このクラスは、`rdf:Property` のサブクラスです。

それでは、例として、「`hasFirstborn`（誰々を最初の子供として持つ）というプロパティーは関数的である」ということを記述するプロパティー公理を書いてみましょう。

その場合に注意しないといけないことは、ただ単に、

```
<owl:FunctionalProperty rdf:ID="hasFirstborn">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:FunctionalProperty>
```

と書いただけでは、`hasFirstborn` がオブジェクトプロパティーなのかデータ型プロパティーなのかということが記述されていないということです。ですから、

```
<owl:ObjectProperty rdf:ID="hasFirstborn">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:ObjectProperty>
```

というプロパティー公理を書いた上で、

```
<owl:FunctionalProperty rdf:about="#hasFirstborn"/>
```

というプロパティー公理を書くか、あるいは、

```
<!ENTITY owl "http://www.w3.org/2002/07/owl#">
```

という実体宣言で `owl` という実体を定義しておいて、

```
<owl:ObjectProperty rdf:ID="hasFirstborn">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:ObjectProperty>
```

というプロパティー公理を書くことが必要になります。

Q 5.3.4　逆関数的って、どういう制約なんですか。

A　「逆関数的」(inverse-functional) というのは、特定の目的語に対応する主語は一意に定まる、という制約のことです。

逆関数的という制約を与えられたプロパティーは、「逆関数的プロパティー」(inverse-functional property) または「IFP」と呼ばれます。

P が逆関数的プロパティーだとするとき、(x_1, y) というペアが P のインスタンスとして存在するならば、x_1 とは異なる x_2 という個体を主語とする (x_2, y) というペアが P のインスタンスとして存在するということはあり得ません。

Q 5.3.5　プロパティーが逆関数的だということを記述するプロパティー公理って、どんなふうに書けばいいんですか。

A　`owl:InverseFunctionalProperty` というクラスを使うことによって、プロパティーが逆関数的だということを記述するプロパティー公理を書くことができます。

`owl:InverseFunctionalProperty` というのは、「逆関数的プロパティー」をあらわすクラスです。このクラスは、`owl:ObjectProperty` のサブクラスです。

たとえば、

```
<owl:InverseFunctionalProperty rdf:ID="firstbornOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:InverseFunctionalProperty>
```

というプロパティー公理を書くことによって、「`firstbornOf`（誰々は誰々の最初の子供である）というプロパティーは逆関数的である」ということを記述することができます。

`owl:InverseFunctionalProperty` は、`owl:ObjectProperty` のサブクラスです。ですから、逆関数的プロパティーは、自動的にオブジェクトプロパティーになります。しかし、このことは、データ型プロパティーを逆関数的プロパティーにすることが絶対に不可能だということを意味しているわけではありません。

OWL Full では、データ値を個体として扱うことができます。このことは、OWL Full では事実上、データ型プロパティーはオブジェクトプロパティーのサブクラスになる、ということを意味しています。ですから、OWL Full では、データ型プロパティーを逆関数的プロパティーにすることも可能です。

それに対して、OWL Lite と OWL DL では、データ型プロパティーはオブジェクトプロパティーのサブクラスではありませんので、データ型プロパティーを逆関数的プロパティーにすることはできません。

5.4　論理的な性質

Q 5.4.1　プロパティー公理で記述される、プロパティーの論理的な性質って、どんな性質のことなんですか。

A　プロパティー公理によって記述される、プロパティーの論理的な性質には、推移的と対称的という 2 種類のものがあります。

推移的という性質については Q 5.4.2 で、対称的という性質については Q 5.4.4 で説明したいと思います。

Q 5.4.2　推移的って、どういう性質なんですか。

A　プロパティー P が「推移的」(transitive) であるというのは、(x, y) が P のインスタンスで、かつ (y, z) も P のインスタンスならば、かならず (x, z) もまた P のインスタンスである、ということです。

推移的という性質を持つプロパティーは、「推移的プロパティー」(transitive property) と呼ばれます。

Q 5.4.3　プロパティーが推移的だということを記述するプロパティー公理って、どんなふうに書けばいいんですか。

A　owl:TransitiveProperty というクラスを使うことによって、プロパティーが推移的だということを記述するプロパティー公理を書くことができます。

owl:TransitiveProperty というのは、「推移的プロパティー」をあらわすクラスです。このクラスは、owl:ObjectProperty のサブクラスです。
たとえば、

```
<owl:TransitiveProperty rdf:ID="ancestorOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:TransitiveProperty>
```

というプロパティー公理を書くことによって、「ancestorOf（誰々は誰々の先祖である）というプロパティーは推移的である」ということを記述することができます。
OWL Lite と OWL DL においては、推移的プロパティーに対して、局所的個数制約または大域的個数制約を与えることはできません。さらに、推移的プロパティーをサブプロパティーとするプロパティー、推移的プロパティーと逆関係にあるプロパティー、推移的プロパティーと逆関係にあるプロパティーをサブプロパティーとするプロパティーに対しても、同様に、局所的個数制約または大域的個数制約を与えることはできません。

Q 5.4.4　対称的って、どういう性質なんですか。

A　プロパティー P が「対称的」(symmetric) であるというのは、(x, y) が P のインスタンスならば、かならず (y, x) もまた P のインスタンスである、ということです。

対称的という性質を持つプロパティーは、「対称的プロパティー」(symmetric property) と呼ばれます。

Q 5.4.5　プロパティーが対称的だということを記述するプロパティー公理って、どんなふうに書けばいいんですか。

A　owl:SymmetricProperty というクラスを使うことによって、プロパティーが対称的だということを記述するプロパティー公理を書くことができます。

owl:SymmetricProperty というのは、「対称的プロパティー」をあらわすクラスです。このクラスは、owl:ObjectProperty のサブクラスです。
たとえば、

```
<owl:SymmetricProperty rdf:ID="spouseOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:SymmetricProperty>
```

というプロパティー公理を書くことによって、「spouseOf（誰々は誰々の配偶者である）というプロパティーは対称的である」ということを記述することができます。

第 6 章　個体

6.1　個体の基礎

Q 6.1.1　個体公理って何ですか。

A　「個体公理」(individual axiom) というのは、個体を定義する記述のことです。

個体公理は、「事実」(fact) と呼ばれることもあります。
個体公理は、次の二つの種類に分類することができます。

(1) クラスへの個体の所属および個体が持つプロパティーの値についての記述。
(2) 個体の同一性についての記述。

クラスへの所属とプロパティーの値について記述する個体公理については 6.2 で、同一性について記述する個体公理については 6.3 で説明したいと思います。

6.2　クラスへの所属とプロパティーの値

Q 6.2.1　クラスへの個体の所属とか、個体が持つプロパティーの値とかについて記述する個体公理って、どんなふうに書けばいいんですか。

A　クラスへの個体の所属および個体が持つプロパティーの値について記述する個体公理というのは、それについて主張する型付ノード要素のことです。

たとえば、

- `Hiroko`（寛子）は `Human`（人間）というクラスに所属している。
- `dateOfBirth`（誕生日の日付）というプロパティーの値は `1977-08-22` である。
- `hasChild`（誰々を子供として持つ）というプロパティーの値は `Yasuko`（靖子）と `Mineo`（峰雄）である。

ということを記述する個体公理は、

```
<Human rdf:ID="Hiroko">
  <dateOfBirth rdf:datatype="&xsd;date">
    1977-08-22
  </dateOfBirth>
  <hasChild rdf:resource="#Yasuko"/>
  <hasChild rdf:resource="#Mineo"/>
</Human>
```

と書くことができます。

Q 6.2.2　名前のない個体について記述する個体公理を書くことって、可能ですか。

A　はい、可能です。

たとえば、

- 名前のない個体が `Human`（人間）というクラスに所属している。
- `spouseOf`（誰々は誰々の配偶者である）というプロパティーの値は `Miyoko` である。

ということを、

```
<Human>
  <spouseOf rdf:resource="#Miyoko"/>
</Human>
```

という個体公理を書くことによって記述することができます。

6.3　同一性

Q 6.3.1　個体の同一性について記述するって、どういうことですか。

A　個体の「同一性」(identity) について記述するというのは、異なる名前 (URI) によって参照されるそれぞれの個体が、実は同一のものである、ということを記述することです。

Q 3.2.16 で説明したように、OWL では、一意名仮説が採用されていませんので、ひとつの個体を参照する名前はかならずしもひとつだけとは限りません。ですから、異なる名前によって参照されるそれぞれの個体というのは、それらの同一性を肯定または否定する明示的な記述がない限り、同一であるか異なるかということについては、両方の可能性があるということになります。

Q 6.3.2　異なる名前 (URI) によって参照されるそれぞれの個体が同一のものだということを記述する個体公理って、どんなふうに書けばいいんですか。

A　`owl:sameAs` というプロパティーを使うことによって、異なる名前 (URI) によって参照されるそれぞれの個体が同一のものだということを記述する個体公理を書くことができます。

owl:sameAs というのは、「何々は何々と同一の個体である」ということをあらわすプロパティーで、その定義域と値域は、ともに owl:Thing です。
たとえば、

```
<rdf:Description rdf:about="#Kouboudaishi">
  <owl:sameAs rdf:resource="#Kuukai"/>
</rdf:Description>
```

という個体公理を書くことによって、「Kouboudaishi（弘法大師）は Kuukai（空海）と同一の個体である」ということを記述することができます。
OWL Full では、クラスやプロパティーを個体として扱うことができますので、クラスの同一性やプロパティーの同一性について記述する個体公理も、owl:sameAs を使って書くことが可能です。

Q 6.3.3 異なる名前 (URI) によって参照されるそれぞれの個体が異なるものだということを記述する個体公理って、どんなふうに書けばいいんですか。

A owl:differentFrom というプロパティーを使うことによって、異なる名前 (URI) によって参照されるそれぞれの個体が異なるものだということを記述する個体公理を書くことができます。

owl:differentFrom というのは、「何々は何々とは異なる個体である」ということをあらわすプロパティーで、その定義域と値域は、ともに owl:Thing です。
たとえば、

```
<rdf:Description rdf:about="#Satomi">
  <owl:differentFrom rdf:resource="#Hiroko"/>
</rdf:Description>
```

という個体公理を書くことによって、「Satomi は Hiroko とは異なる個体である」ということを記述することができます。

owl:differentFrom を使って、3 個以上の名前によって参照されるそれぞれのものが異なる個体だということを記述する場合は、それらの名前のすべての組み合わせについて、それらによって参照されるものは異なる個体だという記述を書く必要があります。たとえば、

```
<rdf:Description rdf:about="#Yukiko">
  <owl:differentFrom rdf:resource="#Wakana"/>
  <owl:differentFrom rdf:resource="#Masayo"/>
</rdf:Description>
```

という個体公理は、Yukiko と Wakana とが異なる個体だということと、Yukiko と Masayo とが異なる個体だということを記述していますが、Wakana と Masayo とが異なる個体だということまでは記述していません。ですから、Yukiko と Wakana と Masayo という 3 個の名前によって参照されるそれぞれのものが異なる個体だということを記述するためには、

```
<rdf:Description rdf:about="#Yukiko">
  <owl:differentFrom rdf:resource="#Wakana"/>
  <owl:differentFrom rdf:resource="#Masayo"/>
</rdf:Description>

<rdf:Description rdf:about="#Wakana">
  <owl:differentFrom rdf:resource="#Masayo"/>
</rdf:Description>
```

というように、二つの個体公理を書くことが必要になります。

Q 6.3.4 3 個以上の名前によって参照されるそれぞれのものが異なる個体だという記述をひとつの個体公理で書くことって、できないんですか。

A いいえ、できます。

owl:AllDifferent というクラスと owl:distinctMembers というプロパティーを使うことによって、3 個以上の名前によって参照されるそれぞれのものが異なる個体だという記述をひとつの個体公理で書くことができます。たとえば、

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Opera rdf:about="#Yukiko"/>
    <Opera rdf:about="#Wakana"/>
    <Opera rdf:about="#Masayo"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

という個体公理を書くことによって、「Yukiko と Wakana と Masayo はそれぞれ異なる個体である」ということを記述することができます。

参考文献

[DCMES,2004] “Dublin Core Metadata Element Set, Version 1.1: Reference Description”, Dublin Core Metadata Initiative, 2004.

[Lacy,2005] Lee W. Lacy, *OWL: Representing Information Using the Web Ontology Language*, Trafford Publishing, 2005, ISBN 1-4120-3448-5.

[OWLGuide,2004] Michael K. Smith, Chris Welty and Deborah L. McGuinness eds., “OWL Web Ontology Language Guide”, World Wide Web Consortium, 2004.

[OWLOverview,2004] Deborah L. McGuinness and Frank van Harmelen eds., “OWL Web Ontology Language Overview”, World Wide Web Consortium, 2004.

[OWLReference,2004] Mike Dean and Guus Schreiber eds., Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider and Lynn Andrea Stein, “OWL Web Ontology Language Reference”, World Wide Web Consortium, 2004.

[OWLSemantics,2004] Peter F. Patel-Schneider, Patrick Hayes and Ian Horrocks eds., “OWL Web Ontology Language Semantics and Abstract Syntax”, World Wide Web Consortium, 2004.

[OWLTestCases,2004] Jeremy J. Carroll and Jos De Roo eds., “OWL Web Ontology Language Test Cases”, World Wide Web Consortium, 2004.

[OWLUseCases,2004] Jeff Heflin eds., “OWL Web Ontology Language Use Cases and Requirements”, World Wide Web Consortium, 2004.

[RDFConcepts,2004] Graham Klyne and Jeremy J. Carroll eds., “Resource Description Framework (RDF): Concepts and Abstract Syntax”, World Wide Web Consortium, 2004.

[RDFPrimer,2004] Frank Manola and Eric Miller eds., “RDF Primer”, World Wide Web Consortium, 2004.

[RDFSchema,2004] Dan Brickley and R.V. Guha eds., “RDF Vocabulary Description Language 1.0: RDF Schema”, World Wide Web Consortium, 2004.

[RDFSemantics,2004] Patrick Hayes eds., “RDF Semantics”, World Wide Web Consortium, 2004.

[RDFSyntax,2004] Dave Beckett eds., “RDF/XML Syntax Specification (Revised)”, World Wide Web Consortium, 2004.

[RDFTestCases,2004] Jan Grant and Dave Beckett eds., “RDF Test Cases”, World Wide Web Consortium, 2004.

[神崎,2005] 神崎正英、『セマンティック・ウェブのための RDF/OWL 入門』、森北出版、2005、ISBN 4-627-82931-0。

[小町,2005] 小町祐史、大野邦夫、須栗裕樹、山田篤、『オントロジ技術入門——ウェブオントロジと OWL——』、東京電機大学出版局、2005、ISBN 4-501-54010-9。

[津田,2005] 津田宏、「RDF と RDF スキーマ」、『コンピュータソフトウェア』、第 22 巻第 4 号、pp. 6–11、2005。

[中山,2001] 中山幹敏、奥井康弘、吉田稔、村上泰介、『改訂版標準 XML 完全解説』、上、技術評論社、2001、ISBN 4-7741-1186-4。

[萩野,2005] 萩野達也、「セマンティック Web の期待と技術階層」、『コンピュータソフトウェア』、第 22 巻第 4 号、pp. 2–5、2005。

[溝口,2005] 溝口理一郎、『オントロジー工学』、「知の科学」、オーム社、2005、ISBN 4-274-20017-5。

[溝口,2006] 溝口理一郎、古崎晃司、來村徳信、笹島宗彦、『オントロジー構築入門』、オーム社、2006、ISBN 4-274-20292-5。

[屋内,2003] 屋内恭輔、安陪隆明、『XML スキーマ書法』、毎日コミュニケーションズ、2003、ISBN 4-8399-1102-9。

[山口,2005] 山口高平、福田直樹、小出誠二、「Web オントロジー記述言語 OWL とその記述能力」、『コンピュータソフトウェア』、第 22 巻第 4 号、pp. 12–18、2005。

索引