# メタデータ情報共有のためのガイドライン

メタデータ情報基盤構築事業
2011 年 3 月 28 日 版

## 目 次

# 1　はじめに

## 1.1　このガイドラインの構成

本ガイドラインは、メタデータの提供者、利用者双方を対象に、メタデータの設計、作成から利用、運用管理まで、メタデータの相互運用性、長期利用可能性を高めるための指針を示します。対象とする範囲が広く、一般的な原則から具体的な記述のための技術上の注意点までを扱うため、次のように構成します。

### 1.1.1　章の構成

全般的な説明から技術的な詳細へと、段階的に章を構成します。

1. **はじめに**：メタデータとその共有についての背景説明を最初におき、ガイドラインの基本的な考え方を述べた上で、指針を一覧します。また、本ガイドラインで用いる用語を簡単に説明します。
2. **ガイドライン**：メタデータ共有のための指針を説明します。技術的な詳細には踏み込まず、メタデータ一般を扱う上での推奨事項を解説します。
3. **ガイドラインの技術詳細**：メタデータを実際に設計、作成したり利用する際に必要になる技術的細部について、推奨事項を説明します。関連するメタデータ技術を網羅するものではなく、疑問が生じやすい点、個別の記述規則などであまりカバーされていない点を中心に解説するものです。項目番号はガイドラインと対応しています。
4. **RDF について**：本ガイドラインで推奨するデータモデル RDF（Resource Description Framework）について、その概要と記述上の留意点を解説します。
5. **メタデータ語彙**：メタデータの記述によく用いられる語彙について簡単に紹介します。
6. **メタデータ・スキーマ言語**：記述規則を定義するために本ガイドラインで推奨する定義言語および簡易記述方法についての詳細を説明します。

### 1.1.2　ガイドラインの内容

指針を枠で囲んで提示し、背景や狙いを述べた上で、指針の細目（推奨するポイント）を箇条書きで示します。細目の前に、対象がメタデータの提供者（設計者）なのか利用者なのかを明示します。

> 指針の基本内容を示す文

**優先度：A〜C**
指針の背景や狙いなどを、1〜2 段落程度で説明した文章。……

メタデータ提供者｜利用者は：

1. 指針の細目

2. もうひとつの細目

3. ……

必要に応じて、サブ項目を設けてより詳しく説明します。具体的な記述例や技術解説は、同じ指針番号に対応する技術詳細において説明します。

### 1.1.3 優先順位

本ガイドラインをメタデータの設計、作成、利用に取り入れるにあたって、一度に実施することが難しい場合は、段階的に推奨事項を採用していくことも考えられます。その際の参考となるように、各指針に A、B、C の優先順位を示し、取り組みのプランを立てやすくしました。

（※優先度 C は重要度が低いということではなく、A、B の指針を実践してからのほうが効率がよい、というほどの位置付けです）

## 1.2　メタデータとは何か

出版物の電子書籍化が進み、大学・研究機関では論文等の学術系機関リポジトリが整備されつつあります。また、国立国会図書館、国立公文書館等においては明治期以降の資料についてデジタル化を進めています。

こうしてウェブ上においてもデジタルコンテンツを提供する基盤ができつつありますが、利用者が実際に求めるものを探す手段としては、全文検索を主とした検索サイトの利用が中心です。検索サイトを用いて見出されるウェブ上の情報は実に多様であり、信頼性に欠けるものも多々存在します。

より信頼度の高い情報を得るためには、個別のデジタルアーカイブが持つデータを頼ったり、各分野のサイトにおいて専用の検索フォームを利用するしかありません。分野を超えて、信頼性の高い検索を横断的に行なうためには、一定の規則に従って記述された**メタデータ**が必要となります。

### 1.2.1　メタデータとスキーマ

メタデータとは記述対象となる情報資源に関して，決められた属性についてその属性値を書き表したものです。これはペットボトルを例に考えると分かりやすいでしょう（図 1）。

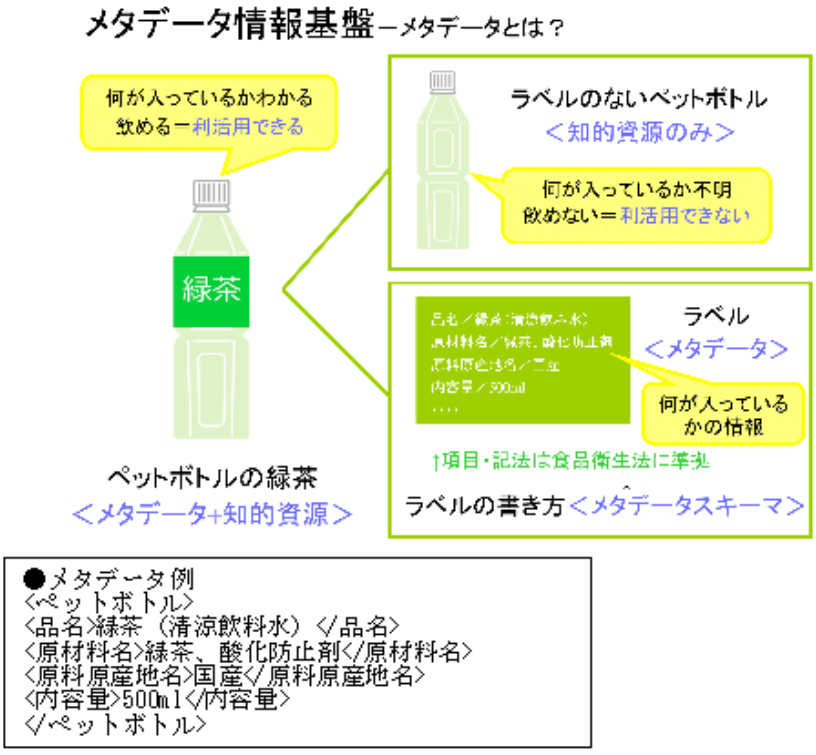


図 1: メタデータはペットボトルのラベルであり、ラベルの表記方法がメタデータスキーマ

通常、ペットボトルには商品名や説明が印字されたラベルがついており、消費者はラベルを見て中身を判断しています。そのラベルをはがしてしまうと、中に何が入っているかわからなくなり、人は飲むこと躊躇するでしょう。

液体の入ったペットボトルをデジタルコンテンツ（知的資源）になぞらえると、そのラベルはメタデータであり、ラベルの表記方法がメタデータスキーマであるといえます。

メタデータは、あらゆるところに存在します。スーパーマーケットの商品には、商品名、原産地、価格、重さなどが表示されています。おもちゃであればパッケージに製造社名、製造場所、対象年齢などが記載されているでしょう。

店の看板などもメタデータです。看板があることによって、その店で何を売っているかを推測することが可能となります。商品の情報だけでなく、利用対象者、利用環境など、細かい属性をメタデータとして設定することで、ユーザーは商品を選びやすくなります。

## 1.3 メタデータ共有の重要性

コンテンツ提供に関わる様々な組織やコミュニティは、コンテンツを利用しやすくするために、メタデータも作成・提供しています。その数は増大の一途をたどっていますが、これらのメタデータは、各機関のコンテンツの特性や要件に基づいて作成され、共有や相互利活用を前提としていない場合が少なくありません。同じ書籍のメタデータといっても、オンライン書店ごとにその内容は異なっています。

様々な目的を持った多種多様なメタデータが利用されている状況の中で、情報資源の発見や相互利用性を高めるためには、メタデータの記述方法が標準化されていなければなりません。ペットボトルの中身を消費者がきちんと理解するためには、商品ごとにラベルの読み方が違っていては具合が悪く、誰にでも分かる共通の表示方式が必要なのです。

相互に利用可能なメタデータを作成するためには、次に示す原則に従った記述の約束が求められます。

- メタデータ記述の語彙やフォーマットに、広く標準として普及しているものを用いる、もしくは新たに定義したものを標準的手法で公開する。
- 日本語の読み、日付、地理情報など、メタデータの記述項目の値（プロパティ値）は標準方法に則って記述する。
- 将来に渡って長期的にメタデータを解釈し情報資源を利用できるようにするため、バージョン管理を含むメタデータ維持管理の情報を加える。

この原則を踏まえたうえで、コンテンツ提供機関のメタデータに求められる内部的要件を損なうことなく、メタデータを相互利活用できるようにするための指針が必要です。

## 1.4　メタデータのライフサイクルとガイドライン

コンテンツが企画設計、制作、利用、保存管理という段階を持つように、メタデータにもライフサイクルがあります。メタデータは、まず目的に合わせた語彙の設計と選択、記述規則の設計が行われます。そしてコンテンツ提供者によって、その規則に従ったメタデータが作成されます。作成されたメタデータは、コンテンツ利用者によって共有・活用され、情報資源の発見、組織化、新しいサービスの創出に生かされます。メタデータおよび記述規則は、必要に応じて追加、修正、バージョン管理等を行い、運用・維持されます（図 2）。

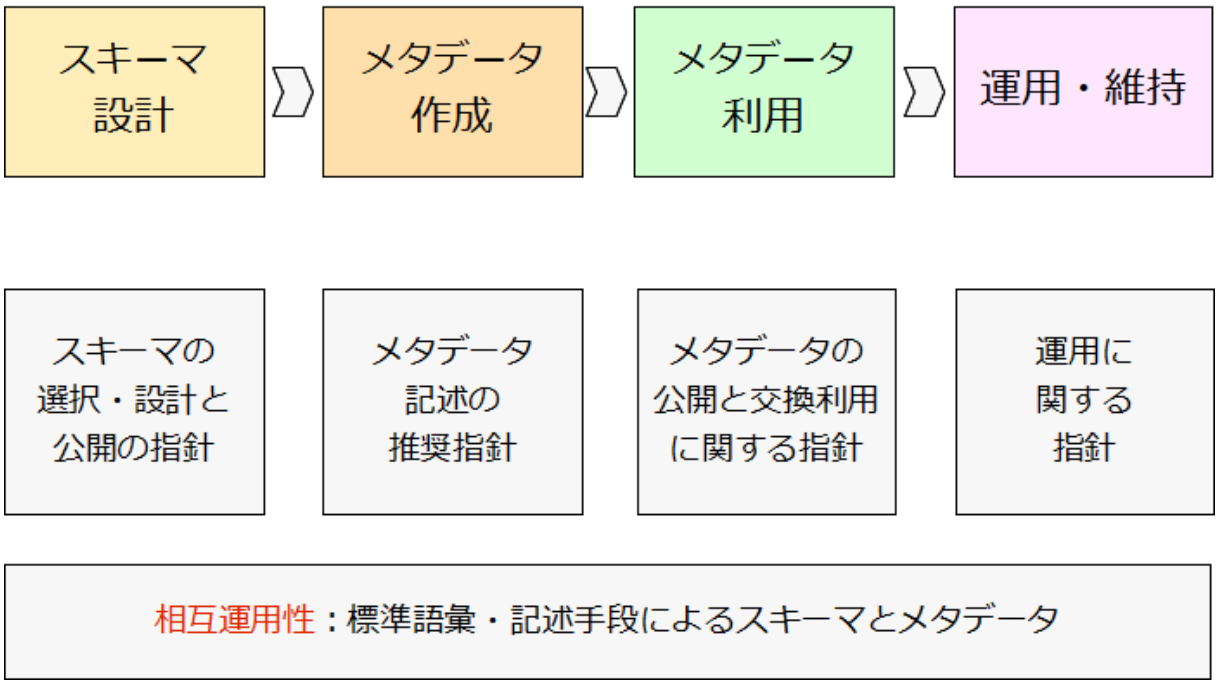


図 2: メタデータのライフサイクルとガイドライン

本ガイドラインは、このライフサイクルの各ステージごとに、推奨される指針を示します。

《設計》、《作成》の指針は主として図書館、公文書館、美術館、博物館、サービス事業者、出版社等のコンテンツ提供者、《利用》の指針は主としてコンテンツ利用者を念頭においていますが、コンテンツ提供者が指針に則ってメタデータを作成すれば、利用者は《作成》の指針に示された形でのメタデータを期待でき、より確実なメタデータ活用が可能となります。

《運用・維持》の指針は、提供者が適切なバージョン管理情報を用意し、利用者がそれを参照することで、コンテンツおよびメタデータの長期にわたる利用が可能になることを目指しています。

## 1.5　指針一覧

本ガイドラインでは、メタデータ共有のために以下の指針を提案します。個々の指針については、第2章で、それぞれの技術詳細については第3章で詳述します（括弧内のA、B、Cは優先度）。

**スキーマの選択・設計と公開の指針**

1. スキーマを相互運用可能な形で選択・設計する。（A）
2. 新たなスキーマを設計する場合、使用語彙や参照記述規則の定義を尊重し、メタデータを相互運用できるよう設計する。（A）
3. 独自スキーマを設計する場合も、特定領域の知識なしに理解し、交換可能なフォーマットに変換するための情報・規則を用意する。（B）
4. スキーマ定義を、コンピュータ処理可能な標準方法でも表現し、公開する。（C）

**メタデータ記述の推奨指針**

1. リソースにグローバルな識別子（URI）を与える。（A）
2. （コンピュータ処理可能な識別子に加えて）人間に理解可能なラベルを標準的な方法で与える。（A）
3. 標準的で再利用可能な形で、コンテンツの作者を記述する。（B）
4. 曖昧さのない標準形式で日時、位置情報を付与する。（B）
5. 可能ならばキーワードを統制語彙で付与する。（B）
6. ラベルに読みを与える場合は、言語タグを用いて区別するか、ラベルを構造化して記述する。（C）
7. リテラル値のデータ型、言語タグは、目的が明確な場合に限って用い、スキーマで使用を宣言して一貫した形で与える。（C）

**メタデータの公開と交換・利用に関する指針**

1. メタデータの公開には、標準的なデータ表現方法としてRDFを用いる。（A）
2. メタデータを正しく理解・利用するためにスキーマを参照し、必要に応じてプロパティの整合調整を行なう。（B）
3. データを公開用などに変換する場合は、情報が失われないように構造と粒度を保ち、利用者がダムダウンする。主要プロパティはあらかじめ単純化値を提供する。（B）

**運用に関する指針**

1. スキーマの管理データを明示し、バージョン管理を行なう。（A）
2. メタデータには作者、作成日時、準拠スキーマなどの管理データを付与する。（A）
3. データを集約して格納する場合、由来情報とあわせて管理する。（B）

4. スキーマを公開レジストリに登録し、利用者の発見を助けるとともに、最新版、旧版を確認できるようにする。(B)
5. メタデータを作成・公開する場合、スキーマの記述規則と矛盾がないか検証する。(C)

## 1.6 本ガイドラインで用いる用語について

このガイドラインでは以下の用語を用います。

**リソース**
書籍、デジタルコンテンツ、人物など、名前をつけて記述できる対象物。

**コンテンツ**
創作活動によってつくられた作品で、印刷物、パッケージ、ネットワークなどのメディアによっ伝達されるリソース。

**メタデータ**
リソースの特徴をひとつ以上の特性（プロパティ）の組み合わせとして表現することで、リソースそのものにアクセスすることなく、その概要を把握できるようにしたデータ。

**レコード（メタデータ・レコード）**
ひとつのリソースの特徴を表現するための、特定の形式に従って表現された、メタデータ項目の集合。

**項目（メタデータ項目）**
ある特徴をメタデータとして記述するための、プロパティとその値の組み合わせ。

**プロパティ**
リソースがもつ特徴のひとつに名前をつけて表現するもの。たとえばある書籍の「タイトル」「作者」など。その名前に URI を用いたものを RDF プロパティと呼びます。

**クラス**
共通の特徴を持つリソースのグループに名前をつけて表現するもの。たとえば「書籍」「映画」など。RDF ではクラスも URI を用いて名前付けます。

**インスタンス**
あるクラスに属する実体リソース。たとえば「ISBN-4-3090-0110-6」で識別されるリソースが「Book」クラスに属するとき、これを「Book のインスタンス」と呼びます。

**ターム**
メタデータの記述に用いる個々のプロパティ名、クラス名。

**語彙**
メタデータの記述に用いるプロパティ、クラスなどの一連のタームを定義したもの。

**項目記述規則**

メタデータ項目を記述するときに、どのプロパティを用いるか、その値にはどのようなものを用いるか（値制約）、ひとつのレコードにその値は必須か、繰り返してよいか（出現回数）などの規則を定義したもの。

**レコード記述規則**

ひとつのメタデータ・レコードを記述するときに、どのような項目を持つか、レコードの対象リソースはどのクラスに属するのかなどの規則を定義したもの。

**スキーマ**

データの構造の定義。このガイドラインにおいては、メタデータの記述に用いる語彙定義と記述規則をあわせてスキーマと呼びます。

**URI（Uniform Resource Identifier）**

リソースの名前を組織・領域を超えて共有するために与えます、グローバルな識別子。`http`、`urn`などの「スキーム」に、それぞれのスキームの規約に基づく識別文字列を加えて構成します。特に `http` スキームによる URI は、WWW で広く用いられます。

**リテラル**

メタデータ項目値のうち、別の実体を参照する名前ではなく、文字列でその値を直接表現するもの。

**データ型**

リテラル値をどのようなデータとして解釈すべきかの情報。データ”1984”が、十進数の数字、西暦 1984 年、文字列（小説「1984」のタイトルなど）のどれであるかを明示するなど。

**RDF（Resource Description Framework）**

リソースについての記述を共有するために、記述対象リソース（主語）、対象の特徴（プロパティ、述語）、その特徴の値（目的語）の三つ組み（トリプル）を用いて表現する標準方法。主語、述語、目的語のそれぞれを URI で表現することで、異なる組織、領域で記述された情報を連動させることができます（4.1 参照）。

**RDF トリプル**

RDF の記述単位である主語―述語―目的語の三つ組み。また、その三つ組みで構成されるひとつのデータ。

**グラフ（RDF グラフ）**

RDF トリプルの集合。複数の情報源から集めた RDF トリプルは、いったん集約してしまうとソースの区別がつかなくなるので、情報源ごとに「グラフ」としてまとめて管理されます。RDF トリプルをデータベースに格納する際には、各トリプルがどのグラフに属するか（グラフ URI）を加えた四つ組としてしばしば扱います。

**RDF スキーマ**

RDF で用いる語彙を定義するための標準手段。プロパティ、クラスの階層関係およびプロパティとクラスの関係を示すことができ、基礎的な推論の手段を提供します。

**OWL（Web Ontology Language）**

ウェブで用いる体系的な語彙（オントロジー）の定義とバージョン管理のための標準手段。ク

ラスをプロパティの制約条件から定義したり、プロパティの型を与えるなど、高度な表現が可能で、記述論理を用いた推論の手段も提供します。RDF スキーマと組み合わせて用いることができます。

**ダムダウン（Dumb down）**

領域固有の語彙やデータ構造を、汎用標準語彙による単純構造に変換することで、データのおよその意味を理解可能とし、相互運用性を高めること。

また本ガイドラインでは、特に断らない場合は次の対応表による接頭辞を用いてプロパティなどを記述します。

| 接頭辞 | 対応 URI | 語彙名 |
|---|---|---|
| rdf: | http://www.w3.org/1999/02/22-rdf-syntax-ns# | RDF 語彙 |
| rdfs: | http://www.w3.org/2000/01/rdf-schema# | RDF スキーマ語彙 |
| owl: | http://www.w3.org/2002/07/owl# | OWL 語彙 |
| dc: | http://purl.org/dc/elements/1.1/ | シンプル DC 語彙 |
| dcterms: | http://purl.org/dc/terms/ | DC タームズ語彙 |
| skos: | http://www.w3.org/2004/02/skos/core# | SKOS 語彙 |
| skosxl: | http://www.w3.org/2008/05/skos-xl# | SKOS 拡張ラベル語彙 |
| foaf: | http://xmlns.com/foaf/0.1/ | FOAF 語彙 |
| dsp: | http://purl.org/metainfo/terms/dsp# | 記述規則定義言語語彙 |

# 2　ガイドライン

## 2.1　スキーマの選択・設計と公開の指針

交換のためのメタデータ設計について、記述語彙の選択・設計、記述規則の定義、シンプルなメタデータフォーマットへの変換に関する指針を示します。

### 2.1.1　スキーマの選択と設計

スキーマを相互運用可能な形で選択・設計する。

**優先度：A**

メタデータを設計する際に、当面扱う資料や組織内での使用のみを念頭に記述規則を決めてしまうと、外部とのメタデータの交換や共有が難しくなる恐れがあります。将来メタデータを他組織と交換したり、広く公開して流通させる可能性を考えて、できるだけ標準的な語彙を選択し、相互運用可能な形で記述規則を設計します。

メタデータ設計者は：

1. 同じ領域、もしくは近隣領域で用いられている相互運用可能な既存スキーマがあれば、できる限りそれを採用する。
2. スキーマを新たに定義する場合は、相互運用可能な形を考える。
3. 記述規則に用いる語彙は、可能な限り広く普及した標準語彙を用いる。
4. 領域固有の項目（プロパティ）を定義する場合は、共有可能な標準語彙とのマッピングを用意する。

### 2.1.2　相互運用性を尊重した記述規則設計

新たなスキーマを設計する場合、使用語彙や参照記述規則の定義を尊重し、メタデータを相互運用できるよう設計する。

**優先度：A**

既存スキーマを利用してメタデータを設計する際に、語彙の定義や記述規則の制約が部分的にニーズに合わない場合があり得ます。このとき、記述規則で元語彙の定義と矛盾する規則を定めると、同じ語彙を用いている他のメタデータとの相互運用が困難になってしまいます。非互換を避けるため、使用語彙や参照記述規則の定義は尊重してください。

メタデータ設計者は：

1. 記述規則で既存語彙のタームを用いる場合は、タームの定義を尊重する。

2. 既存語彙の定義がニーズに合わない場合は、独自語彙を定義して、標準（既存）タームと関連付ける。

### 2.1.3 領域固有の知識なしでも理解可能なスキーマ設計

独自スキーマを設計する場合も、特定領域の知識なしに理解し、交換可能なフォーマットに変換するための情報・規則を用意する。

**優先度：B**

扱うコンテンツの特性や組織で利用するための要件によって、メタデータ記述に用いる語彙、構造はさまざまに異なります。これらの異なるフォーマットのメタデータを相互利用可能な設計が必要です。

詳細なデータを表現するためには、領域固有の専門用語による語彙が必要な場合もあります。メタデータを広く共有・交換するためには、専門外であっても最小限の共通理解を可能にするための情報も提示されていることが望まれます。

メタデータ設計者は：

1. 未知の領域の語彙を利用者（人間）が理解するための説明を提供する。
2. コンピュータ処理のために、何らかのレベルで構造を単純化した共通項を得られるようにする。

メタデータ利用者は：

1. メタデータの理解や異なるデータの組み合わせのために、スキーマで提供される情報を利用する（2.3.2 参照）

**読んで理解できる説明の付与**　未知の領域のメタデータを利用するためには、まず人間の利用者がそのメタデータの記述語彙を理解する必要があります。語彙、記述規則の定義には、機械可読情報に加えて、表示用のラベルと理解のための説明（コメント）を付与してください。

領域外からの利用も考慮し、公開スキーマには、非専門家であっても基本的な意味を理解できるような補助説明、もしくは解説への参照を加えることが望まれます。

（※この指針の狙いは、特定領域でしか扱わない特殊な専門情報を非専門家に説明するということではなく、専門的な用語ではあるものの、一般的な言葉に言い換えも可能、あるいは一般的な用語が細分化・特殊化されたものであるとき、一般用語と関連付けて説明するということです。たとえば音楽データで「フォアシュピーラー」（Vorspieler）というプロパティを用いるときに、それは「次席奏者」の意味である、あるいは `member` のサブプロパティである、などの補助情報を提供すれば、領域外でもその意味を大まかに把握し、データを利用することが可能になります。専門知識がないと利用不能な情報については、“ 領域特有プロパティ ”などのマークを与えておくと良いでしょう。）

**構造を単純化した共通項**　異なるフォーマットのメタデータをコンピュータで利用する場合、処理のために、何らかのレベルで構造を単純化した共通項が必要となります。これは、固有のプロパティをより汎用的なプロパティに置き換えたり、構造化された値を単純な文字列値に変換することを含みます（この単純化をダム・ダウンと呼びます)。

（具体的な方法については、技術詳細を参照）

### 2.1.4　標準方法によるスキーマの定義

スキーマ定義を、コンピュータ処理可能な標準方法でも表現し、公開する。

**優先度：C**

組織において一貫したメタデータを作成するためには、どのような記述項目でレコードを表現するか、それぞれの項目にはどのプロパティを用い、その値をどう記述するかを明示する記述規則が必要となります。

これらの記述規則はフォーマットがまちまちであり、規約として表現される内容も異なっているため、コンピュータによるメタデータの検証に用いたり、他組織の記述規則を再利用することが困難です。記述規則の表現方法も標準化することが、相互運用性の向上につながります。

メタデータ設計者は：

1. 人が読むための説明的記述規則だけでなく、コンピュータ処理、相互参照可能な標準形式の記述規則を用意する。
2. 標準形式記述規則では、《項目規則名》、《プロパティ名》、《最小出現回数》、《最大出現回数》、《値タイプ》、《値制約》、《コメント（説明)》を示す。
3. 標準記述規則を、コンピュータ処理しやすい形で公開する。

メタデータ利用者は：

1. 語彙定義、記述規則定義を参照し、活用する手順を組み込む。

**コンピュータ可読な規則**　記述規則をコンピュータにも利用できるようにするためには、項目ごとにセクションを分け、説明文と表を交互に提供する形よりも、すべての項目が一覧できる表の形が望まれます。たとえば、

表 1: 項目ごとに表を用いる記述規則

| 規則 | 規則内容 |
|---|---|
| 項目規則名 | タイトル |
| プロパティ名 | dc:title |
| 最小出現回数 | 1 |
| 最大出現回数 | 1 |
| 値タイプ | 文字列 |
| 値制約 | なし |
| コメント | 内容を表すタイトル |

という表を説明文とともに項目ごとに繰り返すよりも、次の一覧表のほうがコンピュータ処理には適しています。

表 2: すべての項目が一覧できる規則表

| 項目名 | プロパティ | 最小回数 | 最大回数 | 値タイプ | 値制約 | コメント |
|---|---|---|---|---|---|---|
| 書誌 ID | | 1 | 1 | ID | | 文書の ID |
| タイトル | dc:title | 1 | 1 | 文字列 | | 内容を表すタイトル |
| 著者 | dc:creator | あれば必須 | - | 文字列 | | 作者を一人ずつ記述 |
| 発行日 | dc:issued | あれば必須 | 1 | 文字列 | | コンテンツの発行日 |

メタデータ設計者は、人が参照するための記述規則に加え、別途定める「記述規則定義言語」（DSP 言語）を用いた RDF 形式、もしくは「簡易 DSP」の書式による一覧の形の規則表を、コンピュータ可読な形で用意してください。表 2 は簡易 DSP を用いた記述の一例です。

## 2.2 メタデータ記述の推奨指針

メタデータを共有・交換目的で公開する場合の推奨記述方法の指針を示します。個別の目録、データベースにおける記述は必ずしもこの指針に沿う必要はありませんが、公開に際しては指針に対応するよう変換できることが望まれます。

### 2.2.1 リソースの識別子

リソースにグローバルな識別子（URI）を与える。

**優先度：A**

組織内のデータベースにおける目録（メタデータ）であっても、一般にレコードのID（主キー）フィールドを持ちます。メタデータを公開して共有可能とするためには、このIDが他組織のIDと重複することなく、またWWWのようなグローバルな環境においてもどのレコードを示すのか確実に認識できるようにする必要があります。

メタデータ提供者は：

1. グローバルな識別子として、URI（Uniform Resource Identifier）を用いる。
2. メタデータ・レコードは、このURIを主語として記述する。

組織内データベースのIDフィールドは、必ずしもURIとする必要はなく、メタデータの名前空間URIを定め、IDと名前空間URIとの組み合わせでURIに一意に変換して公開できれば構いません。

メタデータ利用者は：

1. 複数の情報源から収集したメタデータのレコードを、URIを用いて識別、集約する。

### 2.2.2 標準ラベルの付与

（コンピュータ処理可能な識別子に加えて）人間に理解可能なラベルを標準的な方法で与える。

**優先度：A**

メタデータは、コンピュータによる処理、連動、再利用に加え、画面表示のためにも用いられます。このとき、コンテンツが何であるかをすばやく確実に把握できるような情報が必要です。

メタデータ提供者は：

1. 著作物のタイトルなど、コンテンツ作者が付与した名前（オーセンティック名）があればその名前を与える。

2. オーセンティック名がない場合でもコンテンツ提供者が付与した名前を、代表ラベルとしてメタデータ・レコードに加える。

3. 関連ラベル（サブタイトルなど）は、原則として、区切り文字で連結するか構造化して、本タイトルと一体化して扱う。代替ラベル（別称など）は、本タイトルとは別の項目として記述する。

名前は可能ならばユニークな（一組のメタデータ内で重複しません）ものを与えます。複数コンテンツのオーセンティック名が重複する場合は、補助的なプロパティとの組み合わせで識別できるようにすることが望まれます。

メタデータ利用者は：

1. 技術詳細に示す推奨記述法に準じてラベルのプロパティを処理する。

### 2.2.3 作者の標準記述

標準的で再利用可能な形で、コンテンツの作者を記述する。

**優先度：B**
作者の記述にはさまざまな方法が用いられ、それぞれの合理性がありますが、異なるメタデータ間での相互運用性を高めるために、できるだけ推奨記述方法を採用してください。

メタデータ提供者は：

1. メタデータ利用者が作者の姓名を区別できるようにする。

2. 共著・共作、連名の作者は、原則として全員の名前を、順序を保持できるように記述する。

3. 作者を名前のみで示す方法（リテラル記述）と、構造化して連絡先などの属性も含めて示す方法（実体記述）を適切に使い分ける。

（リテラルと実体記述の使い分け、および役割の記述については技術詳細を参照）

メタデータ利用者は：

1. 提供者によって作者の記述法が異なる場合があることを踏まえ、技術詳細の推奨事項に準じて作者情報を適切に処理する。

**姓名の区切り**　作者の名前は、姓名を区別できるように記述してください。
特に日本語の場合、姓名が直接つながれた名前から姓と名を分離することは難しいので、可能な限り姓と名を区切った形で名前を記述します。区切り文字には、スペース文字もしくはカンマを用います。外国名のカタカナ表記については中点「・」を用います。

- スペース文字で区切る場合、言語によって姓名の順が逆になることがあるので、言語タグを用いるなど、記述規則のコメントで解釈方法を明示する。

  - 西欧系外国名の原綴表記、日本人名のローマ字表記は、「名 姓」の順序でスペース区切りするか、「姓, 名」とカンマを用いて姓を前に出すかたちで、姓名を識別できるようにする。

- 同姓同名を区別するために所属、生没年などの補助情報を加える場合、補助情報は括弧に入れるなどして姓名と機械的に区別できるようにし、記述規則のコメントで解釈方法を明示する。

※人名の記述に関する詳細な規則提示はこのガイドラインの範囲を超えるので、国立国会図書館の「個人名・団体名標目の選択・形式基準」[1]などを参照してください。

**共著、連名の記述**　共著、連名の場合のリテラル著者ラベルは、ひとつの著者プロパティの値として、資料に記載された順序を尊重して、一貫した区切り文字で著者を区切って記します。区切り文字の約束は、記述規則で明記してください。

実体型の著者記述が必要な場合は、リテラル型著者記述と併記してください。

（詳細については技術詳細を参照）

#### 2.2.4　日時・位置情報

曖昧さのない標準形式で日時、位置情報を付与する。

**優先度：B**

メタデータの記述において、リソースを時空間に位置づける日時情報、位置情報はしばしば重要な役割を果たします。これらのプロパティおよび値が標準化されていれば、たとえば異なる情報源からのデータを組み合わせてカレンダーや地図上に表示するといったサービスで有効に利用できます。

メタデータ提供者は：

1. 日時記述には曖昧さの無い標準的なフォーマットを用いる。
2. 位置情報は、緯度経度であれば世界測地系の百分率表記を、住所の構造化には標準と互換性のある記述法を用いる。

（それぞれのプロパティについては技術詳細を参照）

メタデータ利用者は：

1. ガイドラインに示されている日時・位置情報のさまざまなフォーマットについて、対応できるよう処理手順を組み込むか、処理できない場合を想定した対応手順を組み込む。
2. プロパティ値の精度、粒度にさまざまな程度があることを踏まえてメタデータを処理する。

[1] http://www.ndl.go.jp/jp/library/data/personalname_standard.html

**日時のフォーマット**　日時プロパティの値は、原則として 2011-02-10T12:30:00 の形（XML スキーマの日時形式）もしくは 2011-02-10 の形（XML スキーマの日付形式）とし、可能ならば項目規則の値制約としてどちらかの形かを明示してください。

個別データベースにおける日時プロパティ値はこれらに限定する必要はありませんが、公開する場合は上記のいずれかに変換することを原則とします。特に、スラッシュ区切りの日付（”02/10/11”の形）は、地域によって意味が異なる[2]ので、公開メタデータの日時プロパティ値での使用は避けるようにしてください。

**位置情報の値**　位置情報を緯度経度で与える場合、世界測地系（WGS84）の百分率値で記述します。住所を構造化して記述する場合は、VCARD 語彙に対応付けできる形で記述してください。

（詳細は技術詳細を参照）

### 2.2.5　キーワード

可能ならばキーワードを統制語彙で付与する。

**優先度：B**

リソースの発見を容易にするために、メタデータにはしばしばキーワード（タグ、主題件名、分類など）が含まれますが、多くの場合これらのキーワードは任意の単語が用いられ、検索漏れやノイズを生じてしまいます。キーワードを広く共有可能な形での記述が求められます。

メタデータ提供者は：

1. 可能ならば統制語彙からキーワードを選ぶ。
2. どの語彙から選んだキーワードであるかを（できれば URI で）示す。

（具体的な値の記述は、技術詳細を参照）

メタデータ利用者は：

1. キーワードを単純文字列として処理するだけでなく、（URI による識別を含む）統制語を有効に利用するための手順を用意する。

### 2.2.6　読みについて

ラベルに読みを与える場合は、言語タグを用いて区別するか、ラベルを構造化して記述する。

[2]”02/10/11”は、日本では 2002 年 10 月 11 日、米では 2011 年 2 月 10 日、欧州では 2011 年 10 月 2 日と解釈されます。

**優先度：C**

日本語では多くの場合に読みが必須要件となりますが、欧米語圏ではラベルを単純にリテラルとしか捉えないため、読みに関する国際的な標準は確立されていません。このガイドラインでは、読みの与え方として 2 通りの方法を示します。

メタデータ提供者は：

1. 1 つのリソースについて 1 つしかない項目（たとえば本名）の場合、ラベルと読みのために同じプロパティを 2 回用い、読みを言語タグで区別して記述してよい。
2. 1 つのリソースで複数の値を持つ可能性がある項目（たとえば別名）の場合や、出現回数制約の検証を行ないたい場合は、ラベルを構造化して正規形と読みを記述する。

（具体的な記述方法は、技術詳細を参照）

メタデータ利用者は：

1. ラベルの値が構造化されて読みを持っていたり、同じプロパティが反復されて言語タグで区別される可能性があることを考慮する。
2. 読みが提供される場合、読みを利用した並べ替えや検索ができるようにする。

### 2.2.7　リテラル値のデータ型と言語タグ

リテラル値のデータ型、言語タグは、目的が明確な場合に限って用い、スキーマで使用を宣言して一貫した形で与える。

**優先度：C**

プログラムでメタデータのリテラル値を扱う際に、(1) その値がテキスト文字列なのか、数値なのか、年なのかといった「型」が分かるほうが処理がスムーズな場合があります。また (2)、音声読み上げや自動翻訳のためなど、何語（日本語、英語、仏語など）の文字列なのかが分かるほうが都合がよい場合もあります。

メタデータのリテラル値には、(1) のために「データ型」を、(2) のために「言語タグ」を持たせることができます。一方、データ型、言語タグを持つものと持たないものは、リテラル値が同じであっても、アプリケーションは異なる値として処理するので、使用には注意を要します。

メタデータ提供者は：

1. データ型は、数値、日付など、アプリケーションで単純文字列とは異なる処理を行なうための情報が必要である場合に付与する。
2. 言語タグは、複数言語によるデータが混在するケースなどの明示的区別が必要である場合に、一貫して付与する。

メタデータ利用者は：

1. リテラル値がデータ型や言語タグを持つ場合があることを理解する。
2. リテラル値の比較、特に異なる情報源からのリテラル値を比較・検索する場合は、値をプレーンテキストにキャストした上で行なう。

## 2.3 メタデータの公開と交換・利用に関する指針

作成したメタデータを公開したり、公開されているメタデータを利用したり組み合わせたりする際に、相互運用性を高めるための指針を示します。

### 2.3.1 RDF によるメタデータの公開

メタデータの公開には、標準的なデータ表現方法として RDF を用いる。

**優先度：A**

メタデータの交換、共有は、関係する組織やサービスが多くなると複雑さを増す。さまざまな組織、サービス間で柔軟にメタデータをやり取りするために、公開データの記述に RDF（4.1 参照）を用います。

メタデータ提供者は：

1. メタデータの公開、共有、交換に RDF を用いる。
2. 内部的な作成、利用、維持管理には目的に応じたシステムと表現方法を用い、公開示には RDF に変換する。

メタデータ利用者は：

1. RDF を利用して異なる情報源のメタデータを集約する。
2. RDF ではないデータを RDF に変換するゲートウェイサービスを利用する。

異なる組織・サービス間でのデータ交換のためには、それぞれのデータ項目や構造のマッピングが必要になります。この交換が複数組織・サービス間になると、必要なマッピングは N 対 N に幾何級数的に増加します。ここで、データ交換の中間項となるフォーマットがあれば、変換マッピングは各組織と中間項の間だけ（N 対 1）ですみます。

RDF は、

- 多様なデータを柔軟に表現できる
- 対象の識別、データ値に URI を用いるため、複数の情報源からのデータを集約したり関連付けられる

- プロパティ（項目名）を標準語彙に関連付けて定義することで、異なるプロパティを同一の標準語彙に集約して横断検索などを可能にする

といった特長を持ちます。これを利用し、RDF を各組織・サービスのデータ交換の中間項と位置づけることができます（4.1.5 参照）

### 2.3.2 スキーマを利用したメタデータ共有・活用

メタデータを正しく理解・利用するためにスキーマを参照し、必要に応じてプロパティの整合調整を行なう。

**優先度：B**

多様なメタデータを組み合わせて利用する場合、領域特有の関係を表すデータが含まれる可能性があります。このとき、語彙定義もしくは記述規則において定義されている上位プロパティが既知の語彙のものであれば、およその意味を把握して利用できます。

メタデータ提供者は：

1. メタデータがどのスキーマに準拠しているかの情報を加えて提供する。
2. スキーマには、プロパティ単純化（ダムダウン）のための情報を用意する（2.1.3 参照）。

メタデータ利用者は：

1. 既知でない語彙のタームが含まれる場合は、語彙定義を参照し、記述されているデータの意味を確認する。
2. データを組み合わせるためにプロパティを揃える（整合調整）必要があれば、上位プロパティ関係を利用して単純化（ダムダウン）を行なう。

具体的なダムダウンの手順については、2.3.3 を参照。

### 2.3.3 変換の際のデータ粒度とダムダウン

データを公開用などに変換する場合は、情報が失われないように構造と粒度を保ち、利用者がダムダウンする。主要プロパティはあらかじめ単純化値を提供する。

**優先度：B**

個々のデータベースで詳細に記述されているメタデータを公開用 RDF に変換する際に、汎用標準プロパティに合わせるために、複数のメタデータ項目が結合されてしまうケースがあります。いったん結合され単純化されてしまったメタデータを再度詳細化することは非常に難しいので、情報が欠落してしまい、再利用に支障をきたす場合があります。

公開のための変換時に、無理な連結や単純化をせず、利用者が変換できるようにします。また、基本的な属性については単純化した値をオリジナルと併せて提供してください。

メタデータ提供者は：

1. データを公開用に変換する際には、元データの情報が失われないように、構造と粒度を保っておく。
2. 基本的なプロパティについて、ダムダウンに相当するデータをあらかじめ加えておく。

元データの構造が冗長である場合は、情報が失われない範囲で、公開時に単純化しても構いません。
スキーマで十分な情報が提供されていれば、アプリケーションはそれを用いてダムダウンを行なうことができます。しかし、すべてのアプリケーションがスキーマを参照するとは限らず、またデータ構造によってはダムダウンにはコストがかかる場合もあります。
**タイトル**（標準ラベル）、**作者**、**日時**情報に関しては、あらかじめシンプル DC 相当（`rdfs:label` など、2.2 での推奨プロパティを含む）に変換したプロパティも併せて提供することを推奨します。

メタデータ利用者は：

1. メタデータの集約のために、まず標準プロパティによる単純値が提供されているかどうかを確認する。
2. 構造化されたデータを、情報が欠落しないように適切に保存した上で、必要な情報を単純化して利用する。

## 2.4 運用に関する指針

公開するメタデータやスキーマを長期にわたって利用可能にするための、運用に関する指針を示します。

### 2.4.1 スキーマのバージョン管理

スキーマの管理データを明示し、バージョン管理を行なう。

**優先度：A**
スキーマは一度作成したら不変とは限らず、必要に応じて語彙を追加したり、定義の細部を修正する場合があります。スキーマの更新内容によっては、古いバージョンのスキーマに準拠して書かれたメタデータが、新バージョンでは正しく解釈できなくなる可能性もあります。メタデータを長期にわたって利用可能にするため、適切な情報の提供が必要です。

メタデータ提供者は：

1. スキーマには、作成日、作成者、バージョンなどのメタデータ（管理情報）を加える。

2. 旧バージョンのスキーマも参照できる形で保存する。

メタデータ利用者は：

1. データセットを最初に利用する時に、データの準拠スキーマとそのバージョンを確認する。

### 2.4.2 メタデータ自身の管理情報の付与

メタデータには作者、作成日時、準拠スキーマなどの管理データを付与する。

**優先度：A**
メタデータを適切に利用し、長期の相互運用性を確保するためには、メタデータ自身がいつ誰によって作成されたかなどの「メタ・メタデータ」も重要になります。

メタデータ提供者は：

1. メタデータのラベル（タイトル）、作者、作成日時、準拠スキーマなどの管理データを付与する。
2. この管理データは、原則としてメタデータを提供するファイル単位で付与する。

メタデータ利用者は：

1. メタデータ取得時に管理情報でデータの鮮度、信頼度などを確認する。
2. さらに、次項の由来情報のために、メタ・メタデータも保存しておく。

### 2.4.3 由来情報の保持

データを集約して格納する場合、由来情報とあわせて管理する。

**優先度：B**
複数の情報源のデータは、いったん集約してしまうと情報源を区別できなくなってしまいます。また同じ情報源であっても、収集のタイミングによってデータが異なる可能性があり、どの時点での情報であるかが重要になることがあります。

メタデータ利用者は：

1. データを集約、保管する場合は、情報源ごとにデータセットを管理できるようにする。
2. データセッごとに、取得元、収集日時、2.4.2 のメタ・メタデータなどの情報を記録する。

### 2.4.4 レジストリへの登録

スキーマを公開レジストリに登録し、利用者の発見を助けるとともに、最新版、旧版を確認できるようにする。

**優先度：B**

メタデータを設計・作成するために、またメタデータを利用する際にその規則を確認するために、スキーマを検索したい場合があります。またあるスキーマについて、過去にさかのぼってその定義を確認したい場合もあります。こうした検索・発見を容易にするために、公開レジストリを活用できます。

メタデータ提供者は：

1. 公開するスキーマは、可能な限り公開レジストリに登録する。
2. レジストリには、最新版に加え、旧版のスキーマも確認できるように登録する。

メタデータを新たに設計・作成する場合に、既存スキーマを参考にできれば開発期間が短縮でき相互運用性も高まりますが、適当なスキーマを見つけるのは簡単ではありません。またメタデータを利用する際に、準拠スキーマが明示されていなくても、ジャンルや使用プロパティからスキーマを検索できれば、規則を確認できる可能性があります。

メタデータ利用者は：

1. スキーマ情報の確認にレジストリを利用する。

### 2.4.5 メタデータの検証

メタデータを作成・公開する場合、スキーマの記述規則と矛盾がないか検証する。

**優先度：C**

メタデータを利用する場合、そのデータがスキーマに正しく準拠していないと、アプリケーションでの処理がうまくいかなくなってしまいます。

メタデータ提供者は：

1. メタデータを公開する際は、準拠スキーマを明示するとともに、そのスキーマに対して妥当であるかの検証を事前に行なっておく。

スキーマの明示については、2.3.2 を参照。

# 3　ガイドラインの技術詳細

第 2 章で示した各指針を実現するための、技術的な詳細を解説します。各指針の節・項番号は、第 2 章と対応しています。

## 3.1　スキーマの選択・設計と公開の指針の技術詳細

### 3.1.1　スキーマの選択と設計

スキーマを相互運用可能な形で選択・設計する。

（この指針に対応する技術詳細はありません）

### 3.1.2　相互運用性を尊重した記述規則設計

新たなスキーマを設計する場合、使用語彙や参照記述規則の定義を尊重し、メタデータを相互運用できるよう設計する。

メタデータ設計者 が、スキーマ設計に際して既存語彙を利用するとき、定義の一部がニーズに合わないことがあります。たとえば《著者やタイトルを構造化するメタデータの設計に DC タームズ（dcterms:）を用いたいが、title プロパティの値域がリテラルと定義されており、読みを付与するための構造化ができない》といったケースが挙げられます。

このとき、記述規則で「dcterms:title の目的語に構造化タイトルを用いる」と制約を定義すると、元語彙定義との間で矛盾が生じ、同じ dcterms:title を用いている他のメタデータとの相互運用が困難になってしまいます。このケースでは、「構造化には dc:title を用いる」という本来の約束を尊重し、記述規則でも dc:title による構造化の制約を定義します[3]。

- 記述規則で既存語彙のタームを用いる場合は、タームの定義を尊重する。
- 既存語彙の定義がニーズに合わない場合は、独自語彙で必要なタームを定義して、標準（既存）タームと関連付ける。

**複数語彙の混在と独自語彙定義**　メタデータ設計者 が、既存の語彙を再利用して記述規則を定義しようとすると、複数語彙の組み合わせが必要となる場合が少なくありません。前の例で言えば、基本的な記述には dcterms:を用いつつ、title のみ dc:語彙を用いることになります。また、著者などの人物記述には FOAF 語彙を用いたり、ラベルの構造化のために SKOS 拡張ラベル語彙を用いるなど、それぞれの目的に応じた語彙を組み合わせるため、参照する名前空間（語彙定義）が増えてしまうことになります。

語彙の組み合わせに関しては、2 つの観点があります。

[3]なお、ここでの本来の定義は「dc:title は構造化記述専用」という意味ではないことに注意してください。dc:title は値域が定義されていないので、リテラル値として用いても構造化値として用いても矛盾は生じません。利用アプリケーションは、どちらの値でも処理できるように対応する必要があります。

1. **語彙は混在して構わない**：基本的には、RDF においてはそれぞれのターム（プロパティ、クラス）は URI（名前空間 URI とローカル名を連結した URI）として扱われるので、「名前空間」は大きな意味を持たず、多数の名前空間が混在してもアプリケーションによるデータ処理上の違いはない。

2. **できれば１つ（少数）の語彙で規則を定義したい**：未知のタームがあるときは、意味を確認するために URI をたどってスキーマ定義を参照する。ひとつのスキーマ（名前空間）に必要なタームがまとめて定義されていれば便利ではある。語彙の組合せが複雑になると、メタデータ記述者が混乱する可能性もあるかもしれない。

後者の観点から、（既存語彙でカバーできるタームも含めて）必要なタームをすべて独自語彙として定義しても構いません。定義した各タームは、標準汎用語彙のタームのサブプロパティとして関連付けておいてください[4]。

いずれにしても、既存語彙の定義と矛盾しないよう、相互運用性に配慮することが重要です。

### 3.1.3 領域固有の知識なしでも理解可能なスキーマ設計

> 独自スキーマを設計する場合も、特定領域の知識なしに理解し、交換可能なフォーマットに変換するための情報・規則を用意する。

異なるフォーマットのメタデータをコンピュータで利用する場合、処理のために、何らかのレベルで構造を単純化した共通項が必要となります。メタデータ設計者 は、シンプル DC による非構造化メタデータをその共通項とし、個別メタデータからこのシンプル DC による共通項に単純化（ダムダウン）するための対応規則を、機械可読な記述規則に用意することを推奨します。

データ利用の際のダムダウン解釈については 2.3.3、ダムダウンのためのマッピング方法については 4.2 も参照。

**プロパティの単純化のための情報**　最も汎用的な標準語彙であるシンプル DC との関連を示すことによって、未知の語彙をコンピュータ処理する際に、最小限の相互運用性を確保できるようにします。

- 記述に用いるプロパティは、直接的もしくは間接的にシンプル DC との関連を示しておく。具体的には、
- 語彙定義において `rdfs:subPropertyOf` 関係を用いる、もしくは
- OWL-DSP の記述規則（3.1.4 参照）において、`dsp:propertyMapping` を用いてシンプル DC のプロパティに関連付ける。

ダブリンコアは、シンプル DC と DC タームズで同じ名前のプロパティ（たとえば `title`）を提供し、後者を前者のサブプロパティと定義しています。したがって、後者とのサブプロパティ関係を定義すれば、シンプル DC とは間接的に関連付けることができます。

[4] 上位プロパティの定義と矛盾しないように関連付ける必要があります。たとえば構造化タイトルを定義した場合、dcterms:title のサブプロパティでは矛盾が生じるので、dc:title のサブプロパティとします。

**DC との直接関連付けと間接関連付け**　ダブリンコアに関連付けるとき、直接、間接のどちらとするかは、実際のプロパティの定義において DC タームズの定義域、値域を継承したいかどうかによります。

たとえば ex:subtitle というプロパティを、dcterms:title のサブプロパティとすれば、値域 rdfs:Literal が継承されます。ex:subtitle の値を構造化して読みを与えたければ、dc:title のサブプロパティとして定義しなければなりません。

逆に、DC タームズの方に意味の近いプロパティがあれば、そちらのサブプロパティとして定義します。たとえば ex:composed は、dc:date でなく dcterms:created のサブプロパティとします。利用アプリケーションは、

```
<#aSymphony> ex:composed "1801" .
```

から、サブプロパティ関係を利用して

```
<#aSymphony> dcterms:created "1801" .
<#aSymphony> dc:date "1801" .
```

の両方を推論できます。アプリケーションが DC タームズを理解するなら、より適切な意味でこの値を扱えることになります。

（※ rdfs:subPropertyOf はこうした汎用的な推論ができるので、dsp:propertyMapping よりも応用範囲が広くなります）

**構造化値を単純化するための情報**　構造化記述の場合、内容のテキスト要素だけを連結しても意味があるようにするか、テキストとして取り出すべき内容を次の方法で提示します：

- 構造化記述の要素のうち、どれかひとつ代表値がある場合は、その値を rdf:value とする。
- ひとつの要素が代表とならない場合は、構造化値の情報として必要十分なテキストを連結し、rdfs:label として提供する。

```
#代表値がある場合
<#aProduct>
  ex:price [
    rdf:value "2000" ;
    ex:unit "JPY"
  ] ;
  ...

#ひとつの要素が代表値とならない場合
<#aPublisher>
  ex:address [
    rdfs:label "100-0000 東京都千代田区八ツ橋 1-1-1" ;
    ex:postal-code "100-0000" ;
```

```
    ex:region "東京都" ;
    ex:locality "千代田区八ツ橋" ;
    ex:street-address "1-1-1"
  ] ;
```

**全体・部分関係の記述**　全体・部分関係は、両者を実体として表現して、部分―dcterms:isPartOf →全体、あるいは全体―dcterms:hasPart →部分で関連付け、部分、全体リソースのメタデータは、それぞれのレコードで記述します。

```
<#bibl123>
  dcterms:title "道草" ;
  dcterms:isPartOf <#volume012> ;
  ...

<#volume012>
  dcterms:title "日本文学全集第 12 巻" ;
  ...
```

その上で、全体、部分の主要プロパティを直接表現するためのショートカット用プロパティを用意しても構いません。

```
<#bibl123>
  dcterms:title "道草" ;
  dcterms:isPartOf <#volume012> ;
  dcndl:seriesTitle "日本文学全集第 12 巻" ;
  ...
```

また、基本レコードと上位（下位）構造のメタデータを連結した項目（「全集タイトル - 作品タイトル」など）を用意し、シンプル DC にマッピングしても構いません（3.2.2「標準ラベルの付与」を参照）。

データベースにおいて全体、部分が独立した実体として扱われていない場合、移行措置として dcterms:isPartOf（dcterms:hasPart）の値を構造化しても構いません[5]。

```
<#bibl123>
  dcterms:title "道草" ;
  dcterms:isPartOf [
    dcterms:title "日本文学全集第 12 巻" ;
    dc:creator "近代文学研究会 編" ;
    ...
  ] ;
```

[5] この場合、上位リソース（全集）は空白ノードとなり、再利用性が低くなるので、暫定的な記述方法と考えるべきです。

...

### 3.1.4 標準方法によるスキーマの定義

スキーマ定義を、コンピュータ処理可能な標準方法でも表現し、公開する。

記述規則を相互に理解・利用可能とし、機械的な検証にも活用するため、メタデータ設計者は（内部的に利用するための表形式記述規則に加えて）「記述規則定義言語」（OWL-DSP）を用いて記述規則を定義してください。OWL-DSP による記述が難しい場合は、表形式記述規則の表現方法を標準化し、OWL-DSP への変換手順を定めた「簡易 DSP」を用いた記述規則定義を用意してください。

**記述規則定義言語（OWL-DSP）**　コンピュータ処理可能な記述規則定義のために**記述規則定義言語**（OWL-DSP）を定めます。

OWL-DSP は、DCMI のシンガポール・フレームワーク[6]で取り入れられた Description Set Profile（記述セットプロファイル[7]）の定義言語[8]をベースに、OWL[9]による定義ができるように拡張・改定した記述言語で、次の要素で構成されます。

- ひとつのメタデータレコードを記述する規則の総体を**レコード記述規則**（DescriptionTemplate）として定義し、個々の項目の記述規則を**項目記述規則**（StatementTemplate）として定義する。
- レコード記述規則は、レコードを構成する項目記述規則と、レコード全体としての規則から成る。
- 項目記述規則は、
  - 項目の記述に用いるプロパティと、その出現回数制約（入力レベル）、および値の制約を表現できる。
  - 項目の値として構造化した記述（たとえば「作者」項目の値が「氏名」「連絡先」などの入れ子項目を持つ）を表現できる。

OWL-DSP のスキーマ（オントロジー記述）を 6.1 に示します。

**簡易 DSP**　簡易 DSP は、表 3 の要素を用いて各項目の記述規則を定義します。

この要素を用いた項目記述規則「行」の集まりとしての表を、レコード記述規則「表」とします。値を構造化する場合は、入れ子となる項目の集合をあらたなレコード記述規則として定義し、値制約から参照する形をとります。複数のレコード記述規則「表」は、「ブロック ID」をつけて識別します。

値制約を検証可能にするためには、データ型、語彙は（名前空間）URI で記述します。これらを修飾名で短縮記述するために、名前空間 URI を接頭辞にマッピングする特別なブロックを先頭に置くことができます（簡易 DSP の詳細は、6.2 を参照）。

簡易 DSP で記述した定義表は、OWL-DSP に変換できます。

[6] http://dublincore.org/documents/singapore-framework/

[7] DCMI のシンガポール・フレームワークにおいて、記述セットプロファイルは「アプリケーション・プロファイルによって定義されるシステムやサービスにおいて、メタデータの実体（レコード）の構造制約を定義し、妥当性検証を可能にする要素」とされています。

[8] http://dublincore.org/architecturewiki/DescriptionSetProfile

[9] OWL は、詳細なクラス定義とプロパティの論理的性質の定義を行ない、論理的な推論を可能にするための、ウェブオントロジー言語。http://www.w3.org/2004/OWL/ を参照。

表 3: 簡易 DSP の要素

| 要素 | 役割 |
|---|---|
| 項目規則名 | 「タイトル」などの項目名 |
| プロパティ | `dc:title` など、項目に対応する RDF プロパティ |
| 最小出現回数 | 任意であれば`0`、必須であれば`1` などの整数（ただし「推奨」「あれば必須」などのテキストでも可） |
| 最大出現回数 | 繰り返し不可であれば`1` などの整数、もしくは制約無しならば`-` |
| 値タイプ | 文字列、構造型（入れ子に相当）、参照値（URI）など |
| 値制約 | 文字列のデータ型（`xsd:date` など）や値の語彙（NDLSH など） |
| コメント | 記述に関する補足説明 |

**RDF スキーマ、XML スキーマと DSP**　メタデータの記述に用いる RDF 語彙の定義は、RDF スキーマ（4.1.3 参照）や OWL を用いて行ないます。OWL-DSP（記述規則定義言語）は、これらで定義した語彙をどのように組み合わせ、記述にどのような制約を与えるかを定義します（図 3）。

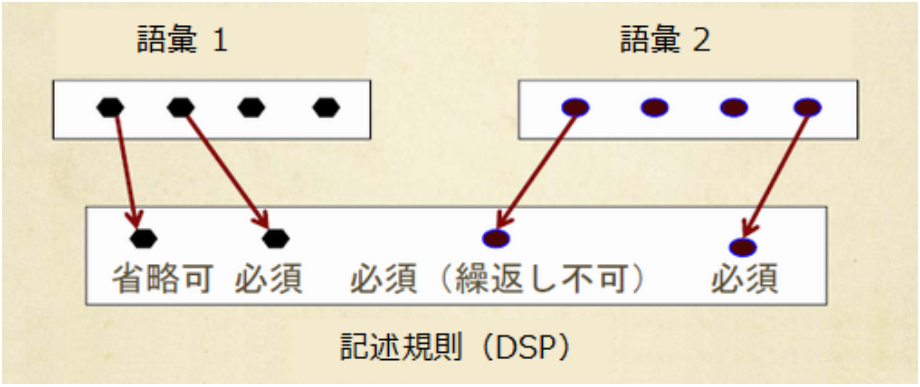


図 3: DSP では、複数の語彙から選んだプロパティを組み合わせ、制約条件を加える

XML スキーマは、（RDF の記述ではないという点を除けば）語彙定義と記述規則の両方を表現できます。XML スキーマも外部語彙をインポートすることは可能ですが、一般には独自の語彙とその記述制約を一括して定義するために用いられることが多いでしょう。

メタデータ作成においては、語彙はできるだけ既存の標準的なものを再利用し、そこに必要な制約条件を加えてスキーマを定義することで、相互運用性を高めることができます。そのため DSP は、語彙定義機能を分離し、組み合わせと制約条件（記述規則）を定義するものとなっています。

## 3.2　メタデータ記述の推奨指針の技術詳細

### 3.2.1　リソースの識別子

リソースにグローバルな識別子（URI）を与える。

メタデータ提供者 は、URI について以下の点に留意してください。

- URI は、urn:などの一般にネットワークアクセスできないスキームより、http:スキームを用いるほうが望ましい。
- いったん公開した URI は、原則として変更しない。そのためには、時間の経過によって変化する可能性のある名前、たとえば組織名（組織変更や統合で名前が変わるかもしれない）や使用技術名（データベースやプログラミング言語を変更するかもしれない）を、URI のパス部分に含めないほうがよい。
- URI 自身はレコードの意味を表す（URI の一部にレコードのタイトルを組み込むなど）必要はないが、必要以上に複雑なものよりは、人間がメタデータを確認する際にも把握可能なものが望ましい。

（いずれも必須の要件ではありません）

**情報リソースと非情報リソース**　ウェブ上のデジタルコンテンツのように、ネットワーク経由でその本質的内容を取得できるリソースを**情報リソース**と呼びます。これに対し、印刷物や人間、会社、抽象概念など、ネットワークで直接アクセスできないリソースを**非情報リソース**と呼びます[10]。これら URI の与え方については、以下の基準を推奨します[11]。

- ネットワーク上のデジタルコンテンツ（情報リソース）のメタデータの場合、そのコンテンツを取得できる（参照解決可能な）URI をメタデータの主語とする。
- 印刷物や所蔵品などの非情報リソースのメタデータの場合、メタデータ自身を公開する URI と、コンテンツを識別する URI を区別する。具体的には、次の方法がある[12]：
  - メタデータファイルの URI に#でコンテンツ識別子を加えた URI（ハッシュURI）をコンテンツ URI とする。もしくは
  - コンテンツ URI が参照されたときに、HTTP 応答コード 303 を用いてメタデータファイルに転送する。
- 主題件名、分類コード、典拠データなど、なんらかの概念や実体を体系化して表現するデータの場合、情報リソースと位置づけて参照解決可能な URI を与えてよい。ただし表現する実体のメタデータ（たとえば著者名典拠における著者の生没年）を記述する場合は、その実体を独立したリソースとして扱って区別し、foaf:primaryTopic、foaf:focus などのプロパティで関連付ける。次の例を参照。

```
#典拠
auth:p0123
    skos:prefLabel "夏目，漱石，1867-1916" ;
    skos:altLabel "夏目，金之助" ;
```

[10] Architecture of the World Wide Web, Volume One http://www.w3.org/TR/webarch/ を参照
[11] 情報リソースと非情報リソースでの URI 使い分けには、W3C において長い議論の末に出された WWW の基本理念の一つですが、理解しにくかったり、手法としても複雑で対応が難しい場合もあります。本ガイドラインでは、URI が何を指すのかについて利用者の混乱が生じない限り、この区分は必須とせず推奨事項にとどめます。
[12] Cool URIs for the Semantic Web http://www.w3.org/TR/cooluris/ を参照。

```
    dcterms:modified "2010-08-03T10:44:00" ;
    dcterms:created "1979-04-01" ;
    foaf:primaryTopic entity:e0123 .

#人物
entity:e0123
    a foaf:Person ;
    foaf:name "夏目漱石" ;
    ex:dateOfBirth "1867" ;
    ex:dateOfDeath "1916" .
```

### 3.2.2 標準ラベルの付与

（コンピュータ処理可能な識別子に加えて）人間に理解可能なラベルを標準的な方法で与える。

メタデータ提供者 は、以下の点に留意してラベルを付与します。メタデータ利用者 は、ラベルの処理に際して以下の指針を考慮してください。

**標準ラベルのプロパティ**　ラベルを示すプロパティは、次の方針で選択します。

- コンテンツにオーセンティック名（作者が付与した名前）がある場合は、ダブリンコアの `title` プロパティを用いる。
  - オーセンティック名でなくコンテンツ提供者が付与した名前でも、コンテンツに由来する名称と考えられる場合は、ダブリンコアの `title` プロパティを用いる。
- 件名標目、シソーラスなどの場合は、`skos:prefLabel` を用いる。
- タイトルには相当しない一般的な名前を与える場合は、`rdfs:label` を用いる。
- 収集品の断片などの場合のように、個別の名前を持たない場合でも、メタデータの記述対象とする場合は、それが何であるかを判別できる呼称を与え、`rdfs:label` としてメタデータに記述する。
- 独自のプロパティで名前を付与する場合は、`dc:title` にダムダウンできるよう定義する。
- ラベルを構造化する（読みを与えるなど）場合は、シンプル DC の `dc:title`（件名標目の場合は `skosxl:prefLabel`）を用いるか、独自のラベル用プロパティを定義して用いる（読みについては 2.2.6 を参照）。
- 人物、団体など、エージェント（`foaf:Agent` あるいは `dcterms:Agent`）に相当するリソースの場合は、基本的に名前（`foaf:name`[13]）をラベルとして考えてよい（3.2.3 も参照）。

[13] foaf:name は rdfs:label のサブプロパティと定義されています。

```
#オーセンティック名がある作品
<#bibl9876>
  dc:title "吾輩は猫である" .

#楽譜が発見された場所による曲名
<#K45a>
  dc:title "旧ランバッハ交響曲" .

#シソーラス
<#concept123>
  skos:prefLabel "交響曲" .

#構造化ラベル
<http://id.ndl.go.jp/auth/ndlsh/00566494>
  skosxl:prefLabel [
    skosxl:literalForm "交響曲" ;
    dcndl:transcription "コウキョウキョク"
  ] .

#断片
<#frag0456>
  rdfs:label "彫像・頭部" ;
  dcterms:isPartOf ....
```

**関連ラベルと代替ラベル**　サブタイトルなどの関連ラベルは、原則として本ラベルと一体として扱います。

- 区切り文字を用いて本タイトルと連結する。区切り文字は、ISBD の" : "（スペース＋コロン＋スペース）のような、標準的に用いられるものが望ましい。
- 構造化ラベルとして扱う。記述方法は標準化されていないが、`skosxl:Label`、MODS の RDF 記述法[14]などを用いることができる。3.1.3 で示したラベル値を単純化する方法にも注意する。

```
<#BN04205645>
  dc:title "誰のためのデザイン? : 認知科学者のデザイン原論" ;
  ...

<#BN04205645>
  mods:title [
```

[14]MIT の SIMILE プロジェクトによる実験版があります。MODS とはタームの使い方も異なりますが、構造は分かりやすくなっています。MODS からの変換 XSLT も用意されています。http://simile.mit.edu/2006/01/ontologies/mods3#

```
    mods:value "誰のためのデザイン? " ;
    mods:sub "認知科学者のデザイン原論"
  ] ;
  ...
```

代替ラベルは、本ラベル（優先ラベル）とは別の項目として扱います。skos:altLabel を用いて記述できます。

```
<#sh95000541>
  skos:prefLabel "World Wide Web" ;
  skos:altLabel "WWW";
  ...
```

別言語のラベルは、本ラベルのプロパティを繰り返して言語タグで区別します。ただし翻訳書の原題のように、本ラベルと同列に扱えない別言語ラベルは、dcterms:alternative で別扱いしても構いません。

**全体・部分関係とラベル**　全集やシリーズなどの上位構造を持つ場合、

- リソース自身の標準ラベルには奥付などの情報源に従った名前を与える。
- 上位構造を別のリソースとして記述し、全集、シリーズなどの名前をそのリソースのラベルとして与える。
- リソース自身と上位リソースを、dcterms:isPartOf で関連付ける。
- 上位構造の名前を含めたラベルの方がリソース自身のラベルよりも代表的と考えられる場合は、リソースの代表ラベルを「上位タイトル - リソースタイトル」の形で記述し、リソース固有のラベルは異なるプロパティで表現してもよい。

```
<#bibl123>
  dcterms:title "道草" ;
  dcterms:isPartOf <#volume012> ;
  ...

<#bibl124>
  dcterms:title "明暗" ;
  dcterms:isPartOf <#volume012> ;
  ...

<#volume012>
  dcterms:title "日本文学全集第 12 巻" ;
  ex:label "第 12 巻　夏目漱石" ;
  dcterms:isPartOf <#set001> ;
```

```
  ...

<#set001>
  dcterms:title "日本文学全集" ;
  ...
```

出版物などの具現化形態（体現形）をコンテンツの基本単位とする場合は、逆に dctems:hasPart で結びつけます。

```
<#volume012>
  dcterms:title "日本文学全集第 12 巻　夏目漱石" ;
  dcterms:hasPart <#bibl123>, <#bibl124>, ... ;
  dcterms:isPartOf <#set001> ;
  ...
```

### 3.2.3　作者の標準記述

標準的で再利用可能な形で、コンテンツの作者を記述する。

作者の記述にはさまざまな方法が用いられ、それぞれの合理性がありますが、異なるメタデータ間での相互運用性を高めるためには、一定の指針も必要です。メタデータ提供者 は、できるだけ以下の推奨記述方法を採用するとともに、シンプル DC へのダムダウン手段を提供してください。
メタデータ利用者 は、作者プロパティの処理に際して以下の指針を考慮してください。

**リテラルと実体記述の使い分け**　作者を名前のみ（リテラル）で記述する場合と実体として記述する場合でプロパティを区別し、可能ならば両方を記述します。

メタデータ・レコードにおける作者の表現方法には、名前のみをリテラル値として記述する方法と、著者を独立した実体として扱い、著者実体への参照として記述します（あるいは入れ子構造で記述します）方法の両方が用いられます。メタデータ利用の際にこれらを区別して扱うことができるよう、プロパティを使い分けます。

- リテラルで記述する場合は dc:creator、もしくはそのサブプロパティ（ただし値域に実体が定義されていないもの[15]）を用いる。
- 実体として記述する場合は dcterms:creator、もしくはそのサブプロパティを用いる。
- 実体レコードのメタデータとして記述する名前は、foaf:name もしくはそのサブプロパティを用いる。
- 名前の読みを加える場合は、実体レコードに言語タグ付き foaf:name で示す。

[15]たとえば dcterms:creator は dc:creator のサブプロパティですが、値域が dcterms:Agent とされているので、値をリテラルにすると矛盾が生じます

```
<#bibl9876>
  dc:title "吾輩は猫である" ;
  dc:creator "夏目漱石 著" ;
  dcterms:creator entity:e0123 ;
  ...

entity:e0123
    a foaf:Person ;
    foaf:name "夏目漱石" ;
    ...
```

一般に実体として記述するほうが情報量が多く、同一著者による複数コンテンツを集約する場合にも有利ですが、後述の連名の場合の順序のように、リテラル値でないと表現が難しい情報もあるので、可能であれば両方を記述します。

実体による記述を含める場合、可能であれば空白ノードによる構造化記述ではなく、国立国会図書館名称典拠 URI[16]などの典拠レコードの URI（もしくは典拠 ID）を用い、メタデータを統合する場合に同一人物を識別、集約できるようにします。

（※ dcterms:creator の値域は dcterms:Agent なので、典拠レコードで標目の URI と実体の URI を使い分けている場合は、実体の URI を目的語とすします（3.2.1 参照）。使い分けがない場合は、典拠の URI を dcterms:Agent とみなしても構いません。）

**共著、連名の記述**　メタデータを RDF として記述する場合、各プロパティ（トリプル）の順序が保存されないので、実体による著者記述だけでは、共著、連名の著者順を示すことが困難です。一方で、特に学術論文の場合、連名著者の場合の順序が重要とされます。

この情報を利用しやすい形で保持するために、リテラルによる複数著者記述には：

- プロパティを繰り返すのではなく、ひとつのプロパティ値に著者を列挙し、資料に記載された順序をそのまま反映する。
- 著者を区切るためには、カンマ、セミコロンなどの区切り文字を用いる。この区切り文字は、姓名の区切りとは異なる文字とし、その用法を記述規則で明示する。
- 複数著者の場合、原則として著者の省略表記（他、et al. などの表記）を用いず、全著者を列挙する。ただし、大半のコンテンツの著者を省略なしに記述できる範囲で、記述規則において上限を定めてもよい。

```
<#ISBN-4-3090-0110-6>
  dc:title "小林秀雄をこえて" ;
  dc:creator "柄谷行人, 中上健次" ;
  dcterms:creator
```

[16]2011 年 3 月現在、開発中

```
    <http://id.ndl.go.jp/auth/entity/00026849> ,
    <http://id.ndl.go.jp/auth/entity/00050918> ;
  ...
```

実体記述の RDF で著者順序を示すためには、RDF リストを用いることもできます（4.3.1 を参照）。

**役割の記述**　主著者以外の編者、監修者などは、dc:contributor として記述できます。役割を詳細化して明示したい場合は、MARC Code List for Relators 語彙[17]のプロパティなどが利用できます。リテラル値記述と実体値記述を併用する場合、リテラル値に役割を示す文字を含めても構いません。

- MARC Code List for Relators 語彙の名前空間は http://id.loc.gov/vocabulary/relators/ で、各タームは dc:contributor のサブプロパティとして定義されているので、ダムダウンが可能。
- リテラル値に役割を示す文字を含める場合、姓名と機械的に区別できるように区切りを設け、記述規則に明示する。

```
<R100000001-I001496096-00>
  dc:title "ロミオとジュリエット" ;
  dc:creator "ウィリアム・シェークスピア 著, 小田島　雄志 訳" ;
  dcterms:creator [foaf:name "ウィリアム・シェークスピア"] ;
  relator:trl [foaf:name "小田島　雄志"] ;
  ...
```

### 3.2.4　日時・位置情報

曖昧さのない標準形式で日時、位置情報を付与する。

メタデータ提供者が日時、位置情報を記述するにあたっては、以下の形を推奨します。メタデータ利用者は、日時・位置情報の処理に際して以下の指針を考慮してください。

**日時のプロパティ**　次の観点でプロパティを選択してください。

- レコードの記述に単一の日時しか必要がない場合は、dc:date を用いる。
- コンテンツの作成、公開、更新などのライフサイクルがある場合は、dcterms:created、dcterms:issued、dcterms:modified などを用いる。

[17] http://id.loc.gov/vocabulary/relators.html を参照

**期間の表現**　期間を表すためには、開始、終了それぞれのプロパティを直接記述する方法と、日時のプロパティ値を構造化して記述する方法があります。この場合、

- イベント情報のような、カレンダーアプリケーションでの利用が想定されるメタデータの場合は、RDF カレンダー語彙[18]など、iCalendar 形式の DTSTART、DTEND に対応付けたプロパティを用いる。
  - iCalendar での日付値はその日の 00:00 と同等に扱われるので、DTEND に日時ではなく日付値を用いる場合、「終了日の翌日」とする必要があることに注意する。
- 作品の作成期間、改訂期間など、期間をひとつの属性として扱うことが望ましい場合は、構造化記述を用いる。
- ひとつの記述規則において、同じプロパティを単一日時と期間の両方の記述には用いない。
- 構造化記述のプロパティとして dc:date を用いてもよいが、その場合は記述規則の値制約で期間であることを明示する。誤解が生じる恐れがあるときは、ミュージアム資料情報構造化モデル[19]の mo:dateRange のように、独自のプロパティを定義して用いる。

開始、終了日時を記述する場合、開始日時を dc:date にダムダウン可能にします。

**日時プロパティの値**　日時プロパティの値は、原則として XML スキーマの日時形式（xsd:dateTime）もしくは日付形式（xsd:date）[20]とし、可能ならば項目規則の値制約としてどちらかのデータ型を明示します。日時の精度を特定できない（年、年月、日付、日時などが混在し得る）場合は、値制約を dcterms:W3CDTF[21]とします。

**確定できない日時の記述**　古典作品の作成日の「1850 年頃」「17 世紀」などように、特定の日時として記述できない値をとりうる項目の場合は、dc:date などではなく、目的に応じたプロパティ（たとえば ex:created）を定義して、規則のコメントに値に関する注記を加えます。

作成日に「1810〜1812 年」と幅がある場合、「1810〜1812 年」の 3 年間をかけて制作した（期間）のか、「1810〜1812 年」頃の作成と考えられる（推定範囲）のかを区別します。前者は dateRange などの期間を表すプロパティで記述できますが、後者は上記の ex:created のように曖昧さを織り込んだプロパティで記述します。

**位置情報のプロパティと値**　位置情報を緯度経度で与える場合、W3C の基本 Geo 語彙[22]を用い、世界測地系（WGS84）の百分率値で記述してください。

```
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
[]
  rdfs:label "東京駅" ;
```

[18] RDF calendar http://www.w3.org/TR/rdfcal/ を参照。なおこの W3C Note では、語彙名前空間が複数記載されているが、通常は http://www.w3.org/2002/12/cal/icaltzd# を用いる。
[19] http://webarchives.tnm.jp/docs/informatics/smmoi-rdf/
[20] 2011-02-14 の形式。http://www.w3.org/TR/xmlschema-2/#dateTime を参照。
[21] http://www.w3.org/TR/NOTE-datetime を参照。
[22] http://www.w3.org/2003/01/geo/ 参照。

```
  geo:lat "35.68113" ;
  geo:long "139.76706" .
```

住所を構造化して記述する場合は、RDF VCard[23]などの vCard 語彙に対応付けできるプロパティを用います。

```
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
[]
  vcard:postal-code "100-0000" ;
  vcard:region "東京都" ;
  vcard:locality "千代田区八ツ橋" ;
  vcard:street-address "1-1-1" .
```

**レコードの主語と位置情報の関係**　位置情報を記述する場合、レコードを記述しているリソースがその位置にあるのか、リソースがその位置となんらかの関係を持つのかを区別してください。

例えば博物館の収蔵品の場合、発掘地はその収蔵品自身の現在位置ではないので、発掘地を表す「場所リソース」と関連付けて示す必要があります。リソースと場所を関連付ける汎用プロパティは標準化されていないので、必要に応じてプロパティを選択もしくは定義してください。

### 3.2.5　キーワード

可能ならばキーワードを統制語彙で付与する。

統制語彙によるキーワードの与え方は、

1. キーワードを URI として表現する
2. キーワードが含まれる統制語彙の名前空間 URI をデータ型として用い、型付リテラルとして表現する

の 2 つの方法があります。メタデータ提供者 がキーワード記述に用いるプロパティとして、1. の方法の場合は `dcterms:subject` を、2. の場合は `dc:subject` を、このガイドラインでは推奨します。「リンクするデータ」の原則[24]を生かすためには、1. の URI 型キーワードがより望ましいといえます。

**URI 型キーワード**　代表的なものとして、国立国会図書館件名標目（NDLSH）、米議会図書館件名標目（LCSH）、DBpedia（Wikipedia の RDF 版）を挙げます。

```
<#ISBN9784839931957>
  dcterms:subject <http://id.ndl.go.jp/auth/ndlsh/ハイパーテキスト> .
```

[23] http://www.w3.org/2006/vcard/ns-2006.html 参照。
[24] http://www.w3.org/DesignIssues/LinkedData.html

| 統制語彙 | 名前空間 URI |
|---|---|
| NDLSH | http://id.ndl.go.jp/auth/ndlsh/ |
| LCSH | http://id.loc.gov/authorities/lcsh/ |
| DBpedia | http://dbpedia.org/resource/ |

**データ型付キーワード**　キーワードが URI として定義されていない場合、たとえば DCMI の符号化スキームの URI を名前空間（データ型）として用い、型付リテラルとしてのキーワードを表現できます[25]。

DCMI 符号化スキームの代表例

| スキーム名 | 統制語彙 |
|---|---|
| dcterms:DDC | デューイ十進分類法 |
| dcterms:LCC | 米国議会図書館分類法 |
| dcterms:UDC | 国際十進分類法 |
| dcterms:TGN | ゲッティ地名シソーラス |

```
<#JPNO67012880>
  dc:title "東京物語" ;
  dc:subject "1001032"^^dcterms:TGN ;
  ...
```

### 3.2.6　読みについて

ラベルに読みを与える場合は、言語タグを用いて区別するか、ラベルを構造化して記述する。

メタデータ提供者 がラベルの読みを記述する場合は、以下の方法を推奨します。メタデータ利用者 は、ラベルの処理に際して以下の形で読み情報が加えられる場合があることを考慮してください。

**言語タグによる読み**　たとえば「タイトルは 1 つのリソースについて 1 つ」と限定できる場合、そのタイトルの読みを、同じプロパティを反復して記述し、言語タグで区別して構いません。

```
<#JPNO67012880>
  dc:title "東京物語"@ja-Kanji ;
  dc:title "とうきょうものがたり"@ja-Kana ;
  ...
```

[25] DCMI 抽象モデルにおける RDF コード化ガイドラインは、dcam:memberOf と rdf:value を組み合わせた構造化記述を示していますが、不必要な構造化は処理を複雑にするので、ここではデータ型付きリテラルを推奨します

この方法は、メタデータの構造を単純なものとできる利点があります。一方で、記述規則の制約を「タイトルは 1 回のみ」として検証を行なうことができません。また、そもそもタイトルが複数ある場合は、読みがどのタイトルに対応するかを判別できません。

- 言語タグが異なれば、プロパティを複数繰り返しても（たとえばカナの読みとローマ字による読みを加えても）よいが、プロパティ値の「意味」は同じものでなければならない。
- この方法でプロパティを繰り返す場合は、対象プロパティのすべての値を言語タグ付きリテラルとする。
- この方法を用いる場合は、項目記述規則の出現回数制約において「1 言語タグあたり最大 1 回」であることを明記する。OWL-DSP では `dsp:cardinalityNote` で示す[26]。

**構造化ラベルによる読み**　単純な言語タグでの読みでは不十分な場合は、`skosxl:Label` 型のリソースとしてラベルを構造化し、その内容にラベル値と読みを記述します。読みのプロパティとしては `dcndl:transcription` を用いることができます。

```
<#JPNO67012880>
  dc:title [
    a skosxl:Label ;
    skosxl:literalForm "東京物語" ;
    dcndl:transcription "とうきょうものがたり"
  ] ;
  ...
```

- 読みをひとつだけ与える場合は、言語タグは用いなくてよい。カナとローマ字など、複数の読みを与える場合は、すべての読みに言語タグを与える。
- `dcndl:transcription` は 1 言語タグあたり 1 回のみ用いるものとし、代替読みを与える場合は、そのためのプロパティを別途用意して記述する。

なお 2010 年 10 月の DC 語彙改定で、構造化ラベルには `dcterms:title` ではなく `dc:title` を用いるか、独自のプロパティを定義することとされています[27]。

```
auth:p0883
  skosxl:prefLabel [
    skosxl:literalForm "藤原，定家，1162-1241" ;
    dcndl:transcription "フジワラ，サダイエ"@ja-Kana ;
    dcndl:transcription "Fujiwara, Sadaie"@ja-Latn ;
    # 別の読みがあるとき、@ja-Kana の dcndl:transcription を繰り返さず、
    # 異なるプロパティで表現。
    ex:altTranscription "フジワラ，テイカ"@ja-Kana
  ] ;
```

[26] dsp:perLangMaxCardinality プロパティの導入を検討中
[27] http://dublincore.org/usage/decisions/2010/dcterms-changes/#sect-2 参照

```
...
# prefLabel の別読みとせず、代替ラベル（altLabel）として記述することもできる
```

### 3.2.7 リテラル値のデータ型と言語タグ

> リテラル値のデータ型、言語タグは、目的が明確な場合に限って用い、スキーマで使用を宣言して一貫した形で与える。

メタデータ提供者 は、データ型、言語タグを用いる場合は、次の点に留意してください。

- データ型を用いる場合は、記述規則もしくはプロパティの値域でデータ型付リテラルであることを明示する。
- 言語タグを与える場合は、記述規則で言語タグが必須であることを明示した上で、常に用いる[28]。

**リテラル値の比較とデータ型・言語タグ**　メタデータ提供者 は、データ型、言語タグを持つものと持たないものは、リテラル値が同じであっても、アプリケーションは異なる値として処理することに留意してください。

メタデータ利用者 は、同じプロパティでも、メタデータによってデータ型、言語タグの与え方にはばらつきがあることに留意します。異なる情報源のメタデータを処理するアプリケーションは、リテラル値を比較するときに、言語タグやデータ型を取り除いた単純リテラルにいったん変換するほうが安全である場合が多いといえます。

**XML における言語タグの注意**　XML で `xml:lang` を用いて言語タグを指定した場合、この属性は子孫要素に継承されます。そのため、不用意にルート要素で言語タグを指定すると、本来言語タグを持つはずのない日付データが言語タグを持ったり、英語標記のラベルが日本語とされてしまったりする場合があります。

```
<rdf:RDF
   xmlns:ex="http://example.org/terms#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xml:lang="ja"
>
<rdf:Description rdf:about="#sample">
  <ex:date>2011-02-14</ex:date>
  <ex:label>English label</ex:label>
</rdf:Description>
</rdf:RDF>
```

この RDF/XML から得られるグラフを Turtle で記述すると、次のようになります。

[28] dsp:langTagOccurrence の導入を検討中

```
@prefix ex: <http://example.org/terms#> .
<#sample>
 ex:date "2011-02-14"@ja ;
 ex:label "English label"@ja .
```

RDF/XML 構文では、言語タグはルート要素におかず、必要なプロパティ要素にのみ記述します（RDFa にも同様の問題があります。こちらは HTML としてルート要素で `lang` 属性を指定することが推奨されるので、気づかないまま不適切な言語タグを与えてしまう危険性が一層高くなります）。

## 3.3 メタデータの公開と交換・利用に関する指針の技術詳細

### 3.3.1 RDF によるメタデータの公開

メタデータの公開には、標準的なデータ表現方法として RDF を用いる。

メタデータ提供者 は、メタデータの公開、共有、交換に際しては、RDF を用いてください。

- RDF を記述する交換構文には、Turtle もしくは RDF/XML 構文を推奨する。
- 新 RDF 作業部会で非推奨となる予定[29]の、RDF コンテナモデル（`rdf:Seq`, `rdf:Alt`, `rdf:Bag` および `rdf:li`）、具体化（Reification）は、原則として使用を避ける。
- RDF は静的ファイルで公開しても、データベースなどから動的に生成しても、どちらでもよいが、リソースの URI を参照したときにメタデータ RDF を取得できるように設定する（ZIP ファイルなどでの提供よりも、参照解決可能な URI とする）。一般に、レコード数の多いメタデータの場合は動的生成が適しているケースが多いといえます。

**関係データベースと RDF の変換**　メタデータ提供者 は、関係データベースに保持しているメタデータを RDF として公開する場合、マッピングファイルを用意することで、変換ツールを利用して容易に RDF 化できます。

マッピングファイルによる変換は、D2R サーバー[30]が提供しているほか、W3C において R2RML[31]としてマッピング言語の標準化が進められています。

**TopicMaps と RDF の変換**　メタデータ提供者、メタデータ利用者 は、知識を記述し関連付けるための技術であるトピックマップ（TopicMaps）で記述したデータは、RDF と相互に変換できます。

両者のマッピングは、RTM（RDF to topic maps mapping）言語[32]を用いて記述できます。このマッピングを用いて、Omnigator などのツールで相互変換を行なうことができます。

[29] http://www.w3.org/2011/01/rdf-wg-charter の Changes/modification to the RDF concepts, model, or semantics
[30] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/ 参照。
[31] http://www.w3.org/TR/r2rml/ 参照。
[32] http://www.ontopia.net/topicmaps/materials/rdf2tm.html 参照。

### 3.3.2　スキーマを利用したメタデータ共有・活用

メタデータを正しく理解・利用するためにスキーマを参照し、必要に応じてプロパティの整合調整を行なう。

メタデータ提供者 は、メタデータ（のファイル）から、準拠する記述規則に dcterms:conformsTo を用いて関連付け、参照可能にしてください。
メタデータ利用者 は、次の手順でプロパティの整合調整（alignment）を行ない、異なるソースのメタデータを組み合わせて活用できます。

- メタデータファイルの dcterms:conformsTo で準拠記述規則を確認する。
- 未知の規則の場合は記述規則を参照し、項目記述規則（dsp:StatementTemplate）で dsp:propertyMapping が定義されているかどうかを調べる。
- 記述規則でのマッピング定義がなければ、語彙定義を参照し、プロパティの上位プロパティを調べる。
- 必要に応じてダムダウンを行ない、理解できる形に変換して利用する。

**元データと公開データの準拠規則**　メタデータ提供者 がメタデータを公開する場合、ある規則に準拠して記述された元データを、別の公開用規則に基づいて変換することもあり得ます。たとえば

- 元の書誌データは「日本目録規則　改訂 3 版」に準拠しているが、
- これを国立国会図書館のメタデータ公開規則である DC-NDL を用いて変換・公開する

といったケースが考えられます。本ガイドラインにおいては、コンピュータによるメタデータ変換利用を想定し、この場合は dcterms:conformsTo は DC-NDL を参照することを推奨します。元データの準拠規則は、データを利用する際に確認できるよう、コメントなどの形で記述します。

```
<public-metadata-file>
   dcterms:comformsTo
     <http://www.ndl.go.jp/jp/library/data/meta/2010/06/ndl-terms> ;
   dcterms:description "元データは日本目録規則　改訂 3 版に準拠" .
```

### 3.3.3　変換の際のデータ粒度とダムダウン

データを公開用などに変換する場合は、情報が失われないように構造と粒度を保ち、利用者がダムダウンする。主要プロパティはあらかじめ単純化値を提供する。

複数のソースからの、異なるプロパティとデータ構造、粒度をもつメタデータを集約して利用する場合、プロパティを整合調整し、粒度が同じレベルになるよう単純化する（ダムダウン）必要があります。

メタデータ提供者 は、ダムダウンのための情報をスキーマの定義に加える（3.1.3 を参照）か、あらかじめ単純化したプロパティを追加してメタデータを公開してください。
メタデータ利用者 は、スキーマ定義を参照して、必要なレベルにダムダウンを行ないます。

**基本プロパティの単純化値**　メタデータ提供者 は、たとえばシソーラスのラベルを構造化プロパティとして提供する場合、標目形を単純化プロパティとしても併せて提供してください。

```
<http://id.ndl.go.jp/auth/ndlsh/00566494>
  skosxl:prefLabel [
    skosxl:literalForm "交響曲" ;
    dcndl:transcription "コウキョウキョク"
  ] ;
  rdfs:label "交響曲" ;
  ...
```

**ダムダウンのためのプロパティ関係記述の利用**　メタデータ利用者 は、複数のメタデータを集約するために：

- スキーマを調べて得られたマッピングを適用する。
- マッピングが与えられない最上位プロパティに関しては、
  - 目的語がリテラルの場合は `dc:description` にマッピングする。
  - 目的語が URI の場合は `dc:relation` にマッピングする。
- 目的語が空白ノードの場合は、`dc:description` として次の構造化プロパティに準じて扱う。

**構造化プロパティの値の利用**　メタデータ利用者 は、取得したデータのプロパティ値が構造化されている場合、次の手順でプロパティ値をダムダウンできます。

- 構造化値の子プロパティとして `rdfs:label` があればその値を用いる。
- `rdfs:label` がなく、`rdf:value` があるときはその値を用いる。
- いずれもない場合は、構造化リソースが持つリテラルプロパティ値を連結する。

RDF においてはトリプルの順序は保持されありません。たとえば次の構造化値をダムダウンする際に、

```
<#xyz>
  dcterms:creator [
     foaf:familyName "バルトーク" ;
     foaf:givenName "ベーラ"]
  ] ;
  ...
```

連結がどの順序で行なわれるかはアプリケーションに依存します。この場合、利用アプリケーションは、連結する際にプロパティのローカル名をラベル（導入句）として加えても構いません。

```
<#xyz>
  dc:creator "familyName:バルトーク, givenName:ベーラ" ;
  ...
```

**ダムダウンのタイミング**　ダムダウンを行なうと、元のメタデータの情報の一部が失われることになります。通常ダムダウンは不可逆の変換なので、メタデータ利用者 は、

- 収集したデータは元の形で保持しておき、利用に際してダムダウンを行なう。もしくは
- 収集したデータはそのまま保持し、ダムダウンしたデータを追加して保存する

という形が望まれます。

また、マッピング定義がないプロパティを dc:description にダムダウンすると意味をなさなくなる場合があるので、元プロパティのローカル名を、カッコ付き（あるいは区切り文字付き）で値の先頭に付加しておくと解釈の手がかりとなります。例えば、

```
<#aProduct> ex:productionDate "2000" .
```

をダムダウンして、dc:description の値を”2000”としてしまうと、これは製品の価格なのか発売日なのか有効期限なのか区別がつかず、役に立たない情報になってしまいます。ダムダウンする場合に

```
<#aProduct> dc:description "productionDate:2000" .
```

とすることで、利用時に意味を確認できます。

## 3.4　運用に関する指針の技術詳細

### 3.4.1　スキーマのバージョン管理

スキーマの管理データを明示し、バージョン管理を行なう。

一般にスキーマの URI は安定していることが望まれます。メタデータ提供者 は、意味解釈の変更を伴うような重大な改定がない限り、最新スキーマは常に同じ URI で参照できるようにしてください。

各バージョンのスキーマを参照可能にするためには、日付もしくはバージョン番号を含む URI で各バージョンを公開する方法がしばしば用いられます。スキーマ本来の名前空間 URI を参照した場合は最新バージョンが取得できるようにサーバーを設定し、最新バージョンと直前のバージョンを、owl:priorVersion を用いて関連付けておきます。

スキーマの名前空間 URI が http://example.org/terms/ex#である場合、2011-02-14 版の最新バージョンには、2010-08-31 版の旧バージョンとの関係を次のように記述します。

```
<http://example.org/terms/ex#>
  a owl:Ontology ;
  owl:versionIRI <http://example.org/terms/ex-20110214#> ;
  owl:priorVersion <http://example.org/terms/ex-20100831#> ;
  owl:versionInfo "version 1.3, 2011-02-14" ;
  dcterms:created "2011-02-14" .
```

### 3.4.2 メタデータ自身の管理情報の付与

メタデータには作者、作成日時、準拠スキーマなどの管理データを付与する。

メタデータ提供者 は、メタデータに（ファイル単位で）タイトル、作者、作成日時、準拠スキーマなどの管理データ（メタ・メタデータ）を付与し、長期の相互運用性を確保してください。

メタデータ提供者 が、メタデータに記述された作成日から適用スキーマのバージョンを知るためには、どのバージョンがどの期間有効だったかを把握する手段が必要になります。

個別に公開されているスキーマの場合、適用バージョンは次の手順で確認します。

1. 管理データの `dcterms:conformsTo` に示されたスキーマ URI にアクセスし、最新バージョンを取得。
2. 取得したバージョンの `dcterms:created` を確認し、メタデータ日付より古ければ、このバージョンを適用版とし、終了。
3. メタデータ日付より新しければ、`owl:priorVersion` の URI にアクセスして前バージョンを取得し、2. を繰り返す。

### 3.4.3 由来情報の保持

データを集約して格納する場合、由来情報とあわせて管理する。

複数の情報源の RDF トリプルは、いったん集約してしまうと情報源を区別できなくなってしまいます。メタデータ利用者 が RDF を集約、保管する場合は、情報源ごとに RDF トリプルを「グラフ」（データセット）としてまとめ、そのグラフ単位に情報源、収集日時などのメタデータを管理してください。

一般的な RDF データベースは、RDF トリプルに、トリプルが属するグラフ URI を加えた四つ組みを単位としてデータを格納します。このグラフ URI を主語として、収集日時や情報源名ラベルなどのメタデータを記述・保管することで、トリプルの由来情報を保持しておくことができます。

2.4.2 で示したメタ・メタデータが提供される場合、併せてグラフのメタデータとして保持しておくことができます。

グラフ URI は、同じ情報源から異なる日時にデータを集める可能性を考慮して、情報源 URI と日時を組み合わせたものとすると安全です。情報源 URI にクエリ文字列（?で結合された文字列）として日付を加えるほか、ある時点でのリソースを識別する URI スキームとして提案されている duri:[33]を用いる方法があります。

```
http://example.org/data/source?20110214
duri:2011-02-14:http://example.org/data/source
```

**由来情報の確認**　メタデータ利用者 は、RDF データをデータベースから取得するとき、SPARQL[34]の GRAPH キーワードを用いてグラフ URI を合わせて取り出すことができます。

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?g ?item ?creator ?date
WHERE {
  GRAPH ?g {
    ?item
      dc: creator ?creator ;
      dc:date ?date .
  }
}
```

取り出したグラフ URI を主語に、改めてメタデータを問い合わせることで、由来情報を確認できます。

### 3.4.4　レジストリへの登録

スキーマを公開レジストリに登録し、利用者の発見を助けるとともに、最新版、旧版を確認できるようにする。

メタデータ提供者 がスキーマを登録するとき、

- スキーマ自体をレジストリに登録し、更新、維持管理もレジストリの機能を利用する、または
- スキーマの管理情報（メタデータ）をレジストリに登録し、スキーマファイルは別途維持管理する

のいずれの方法を採っても構いません。

[33]http://datatracker.ietf.org/doc/draft-masinter-dated-uri/ 参照。
[34]http://www.w3.org/TR/rdf-sparql-query/ 参照。

### 3.4.5 メタデータの検証

メタデータを作成・公開する場合、スキーマの記述規則と矛盾がないか検証する。

記述規則定義言語（DSP）は、OWL のクラス制約を用いて記述規則を定義します。この規則に基づいて作成されたメタデータは、記述規則で定義するレコードクラスのインスタンスとして扱うことができます。

一般に、OWL は制約記述言語ではなく推論言語（定義と異なる記述は新事実、あるいは同じ実体の別名とみなす）であり、制約に対する妥当性検証を目的とはしていませんが、一定の前提のもとに、クラス制約記述を検証に用いることは可能で、そうしたツールも提案されています。

メタデータ提供者 は、メタデータファイルに記述規則をインポートし、メタデータを（一時的に）レコードクラスのインスタンスと明示することで、ツールによる検証ができます[35]。

記述規則で主レコード記述規則#MAIN を次のように定義しており、

```
<#MAIN> a dsp:DescriptionTemplate ;
    rdfs:subClassOf [
        owl:onProperty dcterms:title ;
        owl:minQualifiedCardinality 1 ;
        owl:onDataRange rdfs:Literal
    ], [
        owl:onProperty dcterms:creator ;
        owl:maxQualifiedCardinality 1 ;
        owl:onClass foaf:Agent
    ], [
        owl:onProperty dcterms:issued ;
        owl:minQualifiedCardinality 1 ;
        owl:onDataRange xsd:date
    ] .
    #dcterms:の title および issued が必須で、creator は最大 1 人
```

この規則に基づくメタデータが

```
<#mysample1>
    dcterms:title "メタデータについて" ;
    dcterms:created "2011-02-14"^^xsd:date .
```

と記述されているとします。記述規則が ex:で表される URI で定義されているならば、これを owl:import で取り込んだ上でこのメタデータを ex:MAIN クラスのインスタンス（<#mysample1> a ex:MAIN）として、OWL 推論検証ツールの pellet-ic で検証すると、

[35] OWL-DSP で定義した制約すべてが OWL 推論検証ツールで検証できるわけではありません

```
Validating constraint: MAIN subClassOf (issued min 1 date)
Constraint violated  : Yes
Violating individuals (1): mysample1,
```

という検証エラー（issued の制約を満たしていない＝ created しか記述されていない）が出力されます[36]。

[36]2011 年 3 月現在のバージョンの pellet-ic にはバグがあるので十分利用できません

# 4　RDF について

## 4.1　データモデルの標準：RDF

**RDF**（Resource Description Framework）は、1999 年に W3C から勧告されたリソース一般の記述方法の枠組みです。2004 年には、実装の経験などを踏まえ、RDF で記述された情報の厳密な意味論を含めた改定が勧告され、メタデータ記述方式の国際的な標準となっています。

### 4.1.1　対象の記述とトリプル

RDF は、リソースについての記述を主語（対象）―述語（記述項目）―目的語（項目値）の三つ組み（**トリプル**）として表現するモデルです。トリプルは、主語リソースと目的語リソースを述語をラベルとした矢印で結ぶ図で表現できます（図 4）。

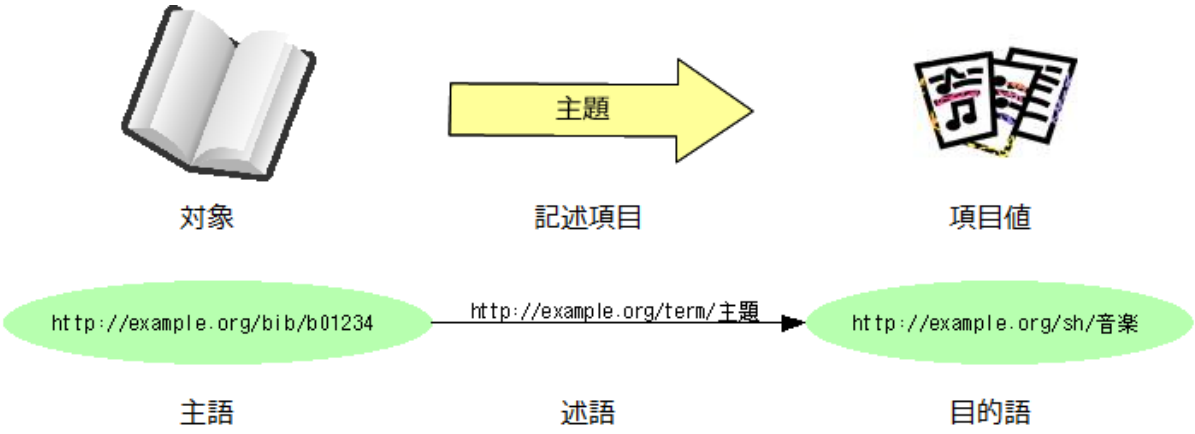


図 4: RDF は主語―述語―目的語トリプルでリソースを記述する

RDF のトリプルの集合を「**グラフ**」と呼びます。また、グラフ中の述語をしばしば「**プロパティ**」と呼びます。

RDF によるリソースの記述は、一つのデータベース内で保持されるだけでなく、ウェブを通して広く共有、交換されることを念頭においています。そのため、トリプルとして記述する主語、述語、目的語は、グローバルに識別が可能な形で表現されなければなりません。RDF はこれらの表現に Uniform Resource Identifier（**URI**）を用います。

URI、特に `http:` スキームによる URI はすでに WWW でのウェブページアドレスとして広く用いられてきています。ドメイン名システムとの組み合わせにより

- ウェブ全体で衝突の恐れのない識別子
- ユーザが任意の識別子を自由に作成できる分散型システム

という 2 つの重要な機能を持っており、どんな組織においても、確実にグローバルに一意な識別子を与えることができます。また、グローバルに一意であるということは、同じ URI を用いて名前付けされたリソースは、同一とみなせることを意味します。従って、異なる組織で別々に記述された RDF グラフであっても、共通の URI を介して連動させることが可能となります（図 5）。

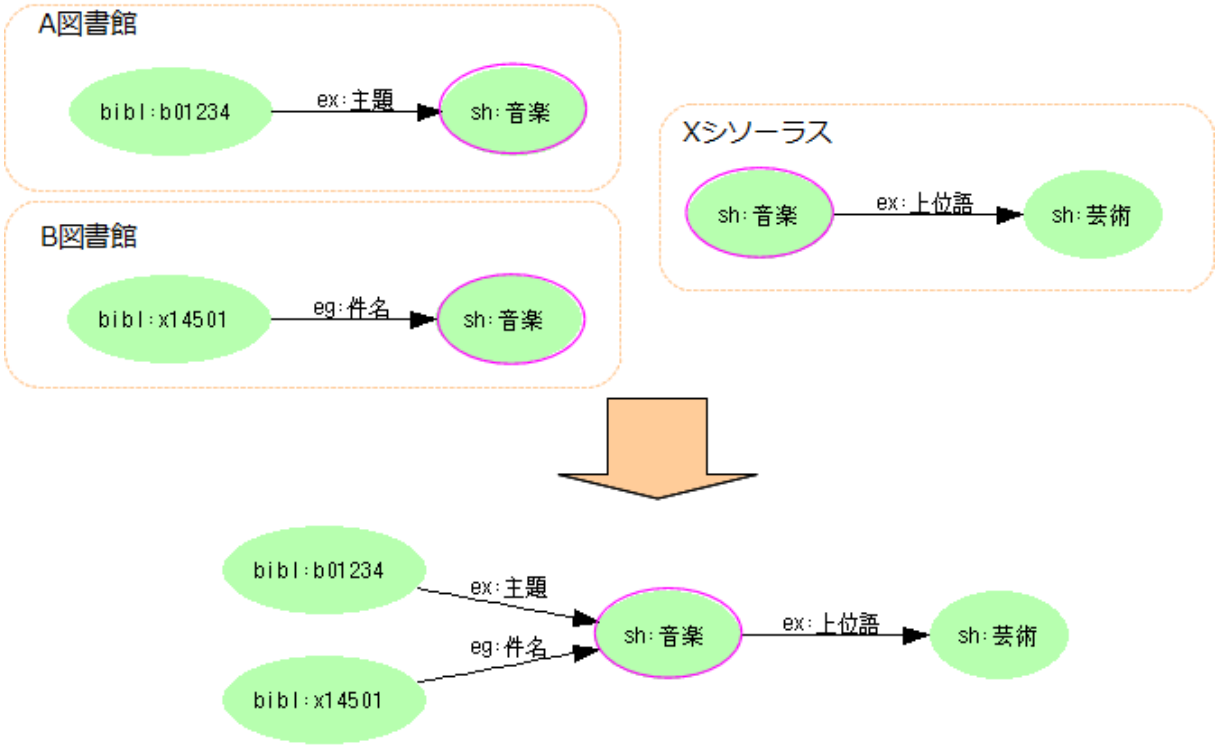


図 5: RDF グラフは同じ URI を介して連結（併合）できる

**リテラルと空白ノード**　また RDF トリプルの目的語は、URI 以外に**リテラル値**を持つことができます。これは、URI のように何らかのリソースを名前付けするのではなく、その値そのものがリソースであるタイプの目的語です。たとえばある人の「名前」プロパティの目的語（値）として「山田太郎」があるとき、この値は文字列そのものであって、何か別のものを表しているわけではありません。リテラル値は、URI と違って主語や述語に用いることはできません。

さらに、トリプルの主語と目的語は、URI で名前付けせずに、匿名の「あるリソース」とすることができます。グラフの図を描くとき、このリソースは空白の円で表されるので、通常これを「**空白ノード**」と呼びます。空白ノードは、数式における変数とよく似た役割を果たします。

### 4.1.2　URI とリソース

RDF の記述対象となる「リソース」は、URI によって名前付け（識別）されます。これは、ウェブ文書のようにネットワーク経由でアクセスできるものに限らず、書籍、人物、概念など、名前付け可能なものすべてを対象とできます。

（図 4）において、主語となる書籍を「`http://example.org/bib/b01234`」と URI で表していますが、この URI を辿っても本そのものがネットワークで送られてくるわけではありません。また、目的語となる主題件名を「`http://example.org/sh/音楽`」としているのも同様で、「音楽」のような具体的な形を持たない概念も URI で表現されます。

主語と目的語の関係（項目名）も、「`http://example.org/term/主題`」という URI となります。項目名を URI で表現することで、異なる組織の項目名を的確に共有したり区別したりできます。さらに、項目名 URI を主語にした RDF を記述すれば、項目名に関する説明や、異なる組織の項目名同士の関

連をも説明することが可能になります。

たとえば、A 図書館の「主題」項目と B 図書館の「件名」項目が、同じ意味を表しているということを、項目名同士の関係として表現できます（図 6）。

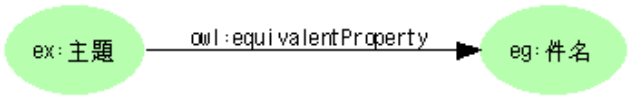


図 6: RDF は関係（項目名）も URI で名付けるので、項目名間の関係を記述することもできる

### 4.1.3 RDF スキーマによる語彙のマッピング

**RDF スキーマ**（RDF Schema）は、記述対象の分類を可能にする「**クラス**」と、記述の述語に用いる「**プロパティ**」を名前付けして定義します。RDF スキーマでは、クラス、プロパティの階層的定義、およびプロパティとクラスの関係定義を行ないます。

たとえば、「作曲家」と「作者」というプロパティを階層関係として（「作曲家」は「作者」の《**サブプロパティ**》である）定義すると、「A は B の作曲家である」というメタデータから、暗黙的に「A は B の作者である」という関係を導くことができます。

RDF ではプロパティも URI で名前付けされるため、異なる領域で定義されるメタデータ・スキーマを関連付け、複数の情報源からのメタデータを相互変換したり連動させることが可能になります。たとえば「x:作曲家」「y:著者」などの異なる語彙のプロパティをダブリンコアの「dc:creator」と関連付けておけば、音楽メタデータと書籍メタデータを、同じ項目名を用いて横断検索できます（図 7、4.2 参照）。

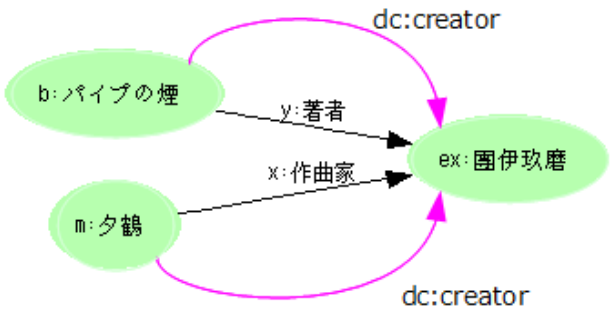


図 7: サブプロパティ関係を用いて異なるメタデータを同じ検索項目で横断検索できる

このようなマッピングは、`rdfs:subClassOf` を用いて記述します。

### 4.1.4 プロパティの値域と項目値の制約

RDF スキーマでは、プロパティの「**値域**」を定義できます。値域は、そのプロパティの目的語となるリソースが暗黙的に属するクラスを示します。例えば、プロパティ「x:作曲家」の値域を「x:音楽家」と定義すると、「A x:作曲家 B.」というトリプルから、B は音楽家であることが分かります。

プロパティの値域は、メタデータの記述規則における項目値の制約条件と裏表の関係になります。値制約は項目値として許される値の範囲（グループ）を限定する規則であるのに対し、値域は目的語（項目値）がどのクラス（グループ）に属するかを推論して導くものです。

値域は制約ではありませんが、矛盾を見つけることはできます。たとえば値域が `rdfs:Literal`（リテラル値）であるプロパティの目的語に人物などを記述すると、その人物がリテラル文字列でもあることになり、矛盾が生じます。こうした性質を利用して、一定の条件下で、プロパティの値域を用いて制約チェックを行なうことも可能です。

プロパティは「**定義域**」も持つことができます。これは値域とは逆に、主語となるリソースの属するクラスを示す働きをします。

### 4.1.5 RDF バス

RDF のグラフは柔軟なモデルを表現できます。表形式データはもちろん、関係データベースの複数テーブルに格納した複雑なデータであっても、RDF の記述に変換することが可能です。いったん RDF に変換したデータは、URI によって記述対象やデータ値を集約でき、また項目名（プロパティ）間の関係も定義できるので、データの交換が容易になります。

異なる組織間でのデータ交換のためには、それぞれのデータ項目や構造のマッピングが必要です。この交換が、1 対 1 ではなく複数対複数になると、必要なマッピングは幾何級数的に増加します。

RDF の多様なデータ表現と URI を介したデータ集約・関連付けを利用し、各組織のデータ交換の中間項に RDF を用いれば、変換マッピングは各組織と RDF の間だけでよいことになります。RDF がデータの交換と共有のための橋渡しを行ないます。RDF はデータ流通の共通基盤です「バス」として機能するのです（図 8）。

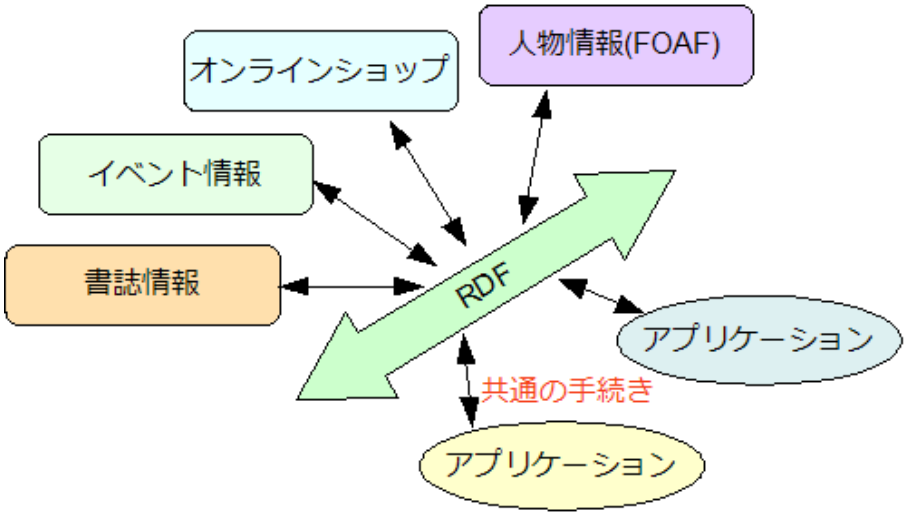


図 8: 異なる組織やアプリケーションのデータを交換・共有するための「バス」として RDF が機能する

## 4.2 ダムダウンのための定義

### 4.2.1 語彙定義におけるサブ・プロパティ関係

新たに定義するプロパティをシンプル DC などの標準汎用語彙に結びつけるためには、「サブ・プロパティ」の関係を用います。ex:著者が dc:creator のサブ・プロパティであることは、次のように記述します。

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix ex: <http://example.org/terms#>.

ex:著者 rdfs:subPropertyOf dc:creator .
```

この関係を用いて、一般的な RDF 推論システムは、

```
<#mybook> ex:著者 <#author> .
```

というデータから次のデータを導くことができます。

```
<#mybook> dc:creator <#author> .
```

この推論を利用して、元メタデータには dc:creator が使われていなくても、dc:creator を対象にした検索が可能になります。

### 4.2.2 データ構造の変換を含むマッピング

プロパティの対応関係だけでは、異なる構造を持つデータをマッピングできません。たとえば作者と作成日を「作成イベント」という形でまとめて記述するメタデータがある場合、

```
@prefix ex: <http://example.org/terms#>.

<#myitem>
  ex:creation_event [
    ex:creator <#author> ;
    ex:date "2011-02-14" ] .
```

サブ・プロパティ関係を定義しても、作者、作成日を<#myitem>の直接のプロパティとして表現することができません。フラットな構造のメタデータとして利用可能にするためには、構造の変換も含めたマッピングが必要です。

**SPARQL の CONSTRUCT**　RDF データのクエリ言語 SPARQL には、検索したデータを特定の構造の RDF グラフとして返すための CONSTRUCT 構文が用意されています。

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ex: <http://example.org/terms#>
CONSTRUCT {
  ?item
    dc:creator ?creator ;
    dc:date ?date .
} WHERE {
  ?item
    ex:creation_event [
      ex:creator ?creator ;
      ex:date ?date ] .
}
```

というクエリを用いることで、前のデータを次の RDF に変換できます。

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.

<#myitem>
  dc:creator <#author> ;
  dc:date "2011-02-14" .
```

## 4.3 RDF の留意点

### 4.3.1 順序の表現

RDF では、トリプルの順序は意味を持たありません。Turtle や RDF/XML の構文において特定の順序で記述されているデータも、いったん RDF データベースに取り込んで再度出力した場合に、異なる順序となる可能性があります。

**RDF リスト** RDF のデータモデルとして、順序を持つ閉じたリストを表現する RDF リストがあります。Turtle 構文では、() の中にリストの項目をスペース区切りで列挙して表現します。

```
<#ISBN-4-3090-0110-6>
  dc:title "小林秀雄をこえて" ;
  ex:creator_list (
    <http://id.ndl.go.jp/auth/ndlnh/00026849>
    <http://id.ndl.go.jp/auth/ndlnh/00050918>
  ) ;
  ...
```

RDF/XML 構文では、プロパティ要素に属性 `rdf:parseType="Collection"`を加え、要素内容にリスト項目を列挙します。

```
<rdf:Description rdf:about="#ISBN-4-3090-0110-6">
  <dc:title>小林秀雄をこえて</dc:title>
  <ex:creator_list rdf:parseType="Collection">
    <rdf:Description rdf:about="http://id.ndl.go.jp/auth/ndlnh/00026849"/>
    <rdf:Description rdf:about="http://id.ndl.go.jp/auth/ndlnh/00050918"/>
  </ex:creator_list>
  ...
</rdf:Description>
```

この場合、ex:creator_list の目的語は閉じたリストとなり、順序も保持されます。

ただし、ここでプロパティとして dcterms:creator を用いると、その値域が dcterms:Agent と定義されていることと矛盾してしまうので、リストを用いる場合はプロパティの定義に注意を要します。また、プロパティの目的語をリストと定義したら、著者が 1 名であってもリストを用いることになる点にも注意が必要です。

**コンテナモデル（非推奨）** RDF/XML 構文には、HTML のリスト項目要素（<li>）に類似した<rdf:li>要素が用意されており、これを rdf:Seq 型付ノードの子要素として列挙することで、順序を表現するという方法が使われてきました。

```
<rdf:Description rdf:about="#myitem">
  <ex:creator>
    <rdf:Seq>
      <rdf:li>Alice</rdf:li>
      <rdf:li>Bob</rdf:li>
      <rdf:li>Cathy</rdf:li>
    </rdf:Seq>
  </ex:creator>
</rdf:Description>
```

rdf:li は RDF のプロパティではなく、RDF/XML 構文の独自要素で、処理アプリケーションはこれを特殊なプロパティに変換して扱います。上記の RDF/XML は、Turtle 構文では次のようになります。

```
<#myitem>
    ex:creator [
        a rdf:Seq ;
        rdf:_1 "Alice" ;
        rdf:_2 "Bob" ;
        rdf:_3 "Cathy"
    ] .
```

モデルとしてトリプルの順序が保持されるのではなく、RDF 処理アプリケーションが rdf:_1、rdf:_2、rdf:_3…というプロパティを番号と解釈することで、結果的に順序を保持する形になります。

このコンテナモデルは、XML 構文とアプリケーションの解釈に依存しており意味論的な裏付けがないこと、また順序表現はリストで可能であることなどから、非推奨（新規データでは用いるべきではありません）とされる見込みになっています。

### 4.3.2 XML の属性付きテキスト要素と RDF

XML では、記述対象のラベルをテキスト要素として、その属性を用いて関連情報を記述する方法がしばしば持ちいられます。

```
<item id="i001">
  <creator id="p301" title="Mr.">
    John Smith
  </creator>
</item>
```

これを RDF で表現する場合、属性の情報を適切に扱うために、テキスト内容を本ラベル（たとえば name）とした実体を考えるとうまくいきます。

```
<item rdf:about="#i001">
  <creator>
    <rdf:Description rdf:about="#p301">
      <title>Mr.</title>
      <name>John Smith</name>
    </rdf:Description>
  </creator>
</item>
```

## 4.4 メタデータのモデリング例

### 4.4.1 フラットな構造のメタデータ

リソースの特徴を、そのリソースを直接の主語としたプロパティの集合で表現するモデル。シンプルで扱いやすく、メタデータ同士の相互変換もスムーズです。一方で、プロパティをグループ化する手段はないため、特徴を細分化した表現には向きません。

```
@prefix ex: <http://example.org/terms#>.

<#myitem>
  ex:title "断章 I" ;
  ex:title_alternative "Fragment I" ;
  ex:creator "John Smith" ;
  ex:created "2000-10-30" ;
```

```
  ex:digitized "2011-02-14" ;
  ex:series_title "変容" ;
  ex:series_alternative "Transfiguration" ;
  ex:series_published "2001-02-30" .
```

### 4.4.2 構造化データ

関連するプロパティをまとめて構造化する形のメタデータモデル。情報はよく整理される一方、どのプロパティを構造化するかは設計者次第なので、複数のメタデータを統合する際には、スキーマを理解したうえで同じ構造に変換する必要があります。

```
@prefix ex: <http://example.org/terms#>.

<#myitem>
  ex:title [
    ex:value "断章 I" ;
    ex:alternative "Fragment I" ;
  ] ;
  ex:creator "John Smith" ;
  ex:created "2000-10-30" ;
  ex:digitized "2011-02-14" ;
  ex:isPartOf [
    ex:title [
      ex:value "変容" ;
      ex:alternative "Transfiguration"
    ] ;
    ex:published "2001-02-30"
  ] .
```

### 4.4.3 ライフサイクル型メタデータ

多くのメタデータモデルでは、作者や日付はリソースの直接のプロパティとして記述されますが、美術館、博物館などでは、作成、発掘、購入などのイベント単位での記述を重視したモデルも用いられます。

```
@prefix ex: <http://example.org/terms#>.

<#myitem>
  ex:title [
    ex:value "断章 I" ;
```

```
  ex:alternative "Fragment I" ;
] ;
ex:creation [
  ex:contributor "John Smith" ;
  ex:date "2000-10-30"
] ;
ex:digitization [
  ex:contributor "山本 徹" ;
  ex:date "2011-02-14"
] ;
...
```

# 5　メタデータ記述に用いられる代表的語彙

広範な資源記述に用いられる RDF 語彙としては、人物を中心とした表現のための FOAF（Friend of a friend）、シソーラス表現の SKOS（Simple Knowledge Organization System）、再利用のためのライセンスを記述する Creative Commons などがあります。これらの語彙は、ダブリンコアや各領域の語彙と組合せることで、より豊かなメタデータの表現を可能にします。

以下にこれらの語彙で定義されている主要なプロパティ、クラスを示します。

## 5.1　ダブリンコア

ダブリンコアの語彙には、異なる領域を通じて共通に使える語彙として、シンプルで基本的（コア）なものに限定した「ダブリンコア・メタデータ要素」と、基本要素を精緻化し、詳細なメタデータ記述が可能な拡張要素を含めた「ダブリンコア・タームズ」があります。

### 5.1.1　ダブリンコア・メタデータ要素（シンプル DC）

1995 年 3 のワークショップでは、できるだけ幅広い領域のリソースを記述できる「コア」なメタデータ要素、**ダブリンコア**（DC, Dublin Core）の策定が目指され、その成果として定義されたのが、ダブリンコア・メタデータ要素（**シンプル DC**）です。

シンプル DC は、さまざまな分野に共通するプロパティを目録専門家でなくても使えるように整備し、分野を越えた意味の相互運用性（semantic interoperability）を得ることが狙いです。そのため要素はシンプルで基本的（コア）なものに限定しており、必要ならこの共通項を基盤にして拡張ができる柔軟性を提供しています。

シンプル DC には表 4 の 15 のプロパティが含まれます。

シンプル DC は、次の名前空間で定義されています。

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

### 5.1.2　ダブリンコア・タームズ（DC タームズ）

シンプル DC は、「コア」であることを目指して 15 のプロパティを定義するにとどめ、より精緻な記述のためには、利用者がプロパティ拡張できるようにしました。しかし「作成日」「更新日」といった頻繁に用いる記述を利用者ごとに拡張していては、「意味の相互運用性」という重要な目的が果たせません。そこで、ダブリンコア自身が基本要素を精緻化し、詳細なメタデータ記述が可能な拡張要素を導入し、シンプル DC の 15 要素とあわせて DCMI Metadata Terms（**DC タームズ**）を定義しました。

表 5、表 6 に、この DC タームズから主要なプロパティを挙げます。

DC タームズは、次の名前空間で定義されています。

```
http://purl.org/dc/terms/
```

表 4: DCMES のプロパティ

| プロパティ | 定義 |
|---|---|
| title | リソースに与えられた名前 |
| description | リソースに関する説明 |
| date | リソースのライフサイクル中の出来事に関連する日時もしくは期間 |
| creator | リソースの作成＊に主たる責任を持つ実体 |
| contributor | リソースへの協力、貢献に責任を持つ実体 |
| publisher | リソースを利用可能にすることに責任を持つ実体 |
| type | リソースの性質もしくはジャンル |
| format | ファイル形式、物理メディア、リソースのサイズなど |
| language | リソースの言語 |
| identifier | ある文脈におけます、リソースへの曖昧さのない参照 |
| rights | リソースに適用される権利に関する情報 |
| relation | 関連するリソース |
| source | リソースの派生元リソース |
| subject | リソースのトピック |
| coverage | リソースの空間的あるいは時間的トピック、あるいは適用対象、リソースが有効となる地域など |

表 5: タイトル、著者などに関するプロパティ

| プロパティ | 定義 |
|---|---|
| title | リソースに与えられた名前 |
| alternative | 代替となるタイトル |
| description | リソースに関する説明 |
| tableOfContents | リソースのサブ単位のリスト（目次） |
| abstract | リソースの要約 |
| date | リソースのライフサイクル中の出来事に関連する日時もしくは期間 |
| created | リソースが作成された日 |
| valid | リソースが有効となる期日もしくは期間 |
| available | リソースが利用可能となる期日もしくは期間 |
| issued | リソースの公式発行日 |
| modified | リソースが変更された日 |
| creator | リソースの作成に主たる責任を持つ実体 |
| contributor | リソースへの協力、貢献に責任を持つ実体 |
| publisher | リソースを利用可能にすることに責任を持つ実体 |
| language | リソースの言語 |
| identifier | ある文脈におけます、リソースへの曖昧さのない参照 |
| rights | リソースに適用される権利に関する情報 |

表 6: フォーマットや関連を記述するプロパティ

| プロパティ | 定義 |
|---|---|
| type | リソースの性質もしくはジャンル |
| format | ファイル形式、物理メディア、リソースのサイズなど |
| extent | リソースのサイズもしくは長さ |
| medium | リソースの素材もしくは媒体 |
| relation | 関連するリソース |
| isVersionOf | 主語リソースは目的語リソースのバージョン、版、翻案 |
| hasVersion | 目的語リソースは主語リソースのバージョン、版、翻案 |
| isPartOf | 主語リソースは目的語リソースの物理的もしくは論理的一部 |
| hasPart | 目的語リソースは主語リソースの物理的もしくは論理的一部 |
| isReferencedBy | 主語リソースは目的語リソースから参照、引用、あるいはポイントされている |
| references | 目的語リソースは主語リソースから参照、引用、あるいはポイントされている |
| source | リソースの派生元リソース |
| subject | リソースのトピック |
| coverage | リソースの空間的あるいは時間的トピック、あるいは適用対象、リソースが有効となる地域など |
| spatial | リソースの空間的な特徴 |
| temporal | リソースの時間的な特徴 |

## 5.2 FOAF

メタデータには著者をはじめ人物に関する記述がしばしば登場します。人とその活動に関する情報の記述には、 FOAF（Friend of a friend）語彙がよく用いられています。FOAF 語彙は、次の名前空間で定義されます。

```
http://xmlns.com/foaf/0.1/
```

FOAF は人物だけでなく、人物に関連する幅広いリソースを記述することを念頭に置いているため、人物だけではなく、その制作物である文書、写真、あるいは所属する組織などを表現するためのクラス（表 7）が用意されています。

表 7: FOAF が定義するクラス

| クラス | 説明 |
|---|---|
| Agent | 人間、グループ、ソフトウェアなど、「ある行為をする能力のある人（もの）」を総称するクラス |
| Person | 人物を表すクラス。Agent のサブクラス |
| Group | Agent の集まりであるグループを表すクラス。Agent のサブクラス |
| Organization | 会社、協会など、社会的な Agent をあらわすクラス。Agent のサブクラス |
| Document | 文書を表すクラス |
| Image | 画像を表すクラス。Document のサブクラス |

人（もしくは Agent）を主語として、そのプロフィールを表現するプロパティには、表 8 のものが含まれます。

FOAF には、人（Agent）ばかりでなく、人が関心を持つ一般的なリソースなどを記述するためのプロパティも用意されています。主要なものを表 9 に挙げます。

表 8: Agent のプロフィールを表現するプロパティ

| プロパティ | 定義 |
|---|---|
| name | 名前（人に限らず使えます） |
| surname | 姓 |
| givenname | 名 |
| nick | ニックネーム |
| gender | 性別 |
| title | 敬称（Mr, Ms, Dr など） |
| mbox | 特定の所有者に結びつけられたメールボックスで、家族などと共有していないもの |
| homepage | ホームページ |
| weblog | ウェブログ |
| openid | OpenID |
| knows | 知っている人 |
| interest | 関心を持っていることに関するページ |
| topic_interest | 関心を持っているトピック |
| publications | 主語人物の出版、発表物リストのページ |
| made | 主語エージェントが作ったもの |
| schoolHomepage | 母校のホームページ |
| workplaceHomepage | 勤務先のホームページ |
| currentProject | 現在手がけているプロジェクト（のページ） |
| pastProject | 以前手がけたプロジェクト（のページ） |

表 9: 人が関心を持つ一般的なリソースなどを記述するためのプロパティ

| プロパティ | 定義 |
|---|---|
| depiction | 主語を描いたもの。写真やイラストなど |
| depicts | 主語画像が描いている内容（depiction の逆） |
| maker | リソースの作者（made の逆） |
| based_near | 目的語の近くにいます（あります）ことを示す |
| page | 主語リソースに関するページ、文書 |
| topic | 主語ページ、文書のトピック（page の逆） |
| primaryTopic | 主語ページの中心トピック。topic のサブプロパティ、かつ関数型プロパティ |
| isPrimaryTopicOf | primaryTopic の逆の関係を表す IFP |
| member | 目的語は主語グループのメンバーである |

## 5.3 SKOS

シソーラス、タクソノミー、分類表や件名標目表などの知識組織化体系をウェブを通じて共有、リンク付けするための語彙として、SKOS（Simple Knowledge Organization System）が W3C から 2009 年に勧告されています。SKOS では分類や件名を概念（Concept）クラスで表現し、それらの属性や関係を体系的に記述します。

SKOS の主要なプロパティとしては、概念に名前を与えるもの（表 10）、概念の関連を表現するもの（表 11）、概念記述の注を与えるもの（表 12）が定義されています。

表 10: SKOS のラベルプロパティ

| プロパティ | 定義 |
|---|---|
| prefLabel | 優先ラベル（標目） |
| altLabel | 代替ラベル（同義語） |
| hiddenLabel | 非表示ラベル |

表 11: SKOS の概念関連付けプロパティ

| プロパティ | 定義 |
|---|---|
| broader | 広義語 |
| narrower | 狭義語 |
| related | 関連語 |
| broadMatch | 別のシソーラスにおける広義語 |
| narrowMatch | 別のシソーラスにおける狭義語 |
| relatedMatch | 別のシソーラスにおける関連語 |
| closeMatch | 別のシソーラスにおける類似語 |
| exactMatch | 別のシソーラスにおける同義語 |
| inScheme | 概念が属するスキーム（シソーラス） |
| hasTopConcept | シソーラスの最上位概念 |
| topConceptOf | 最上位概念として属するシソーラス |

SKOS は次の名前空間で定義されます。

```
http://www.w3.org/2004/02/skos/core#
```

また、読みを持たせるなど構造的なラベルを表現するために、SKOS 拡張ラベル語彙も SKOS 勧告で定義されています。これは拡張ラベルを `Label` クラスのインスタンスとして表現し、表 13 のプロパティで関連付けるものです。

SKOS 拡張ラベルは次の名前空間で定義されます。

```
http://www.w3.org/2008/05/skos-xl#
```

表 12: SKOS の注釈プロパティ

| プロパティ | 定義 |
|---|---|
| notation | 表記法 |
| note | 注 |
| editorialNote | 編集注 |
| historyNote | 編集履歴 |
| changeNote | 変更注 |
| scopeNote | スコープノート（適用範囲） |
| example | 例 |
| definition | 定義 |

表 13: SKOS 拡張ラベルのプロパティ

| プロパティ | 定義 |
|---|---|
| literalForm | ラベルの参照形 |
| prefLabel | 優先拡張ラベル |
| altLabel | 代替拡張ラベル |
| hiddenLabel | 非表示拡張ラベル |
| labelRelation | 拡張ラベル間のリンク |

# 6　メタデータ・スキーマ定義言語

## 6.1　記述規則定義言語

記述規則定義言語（OWL-DSP）のオントロジー記述を以下に示します。

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dsp: <http://purl.org/metainfo/terms/dsp#> .
@base <http://purl.org/metainfo/terms/dsp>.

<> a owl:Ontology ;
    rdfs:label "Description Set Profile Definition Language" ;
    rdfs:comment """メタデータの記述規則を表現するためのメタ言語。メタデー
    タのレコード記述規則は、OWL のクラスで表現し、各レコードの項目記述規
    則を OWL のクラス制約として表現する。メタデータをレコード記述規則クラス
    のインスタンスとして記述することで、推論ツールの拡張による整合性検証
    を可能にする。このメタ言語では、制約記述を表現するためのプロパティの
    ほか、OWL クラス、OWL クラス制約がレコード記述規則、項目記述規則である
    ことを明示するためのメタクラスも定義する。""" ;
    dct:created "2010-12-20" ;
    dct:modified "2011-02-13" ;
    owl:versionInfo "ver. 0.30" .

##クラス定義
dsp:DescriptionTemplate a owl:Class ;
    rdfs:subClassOf owl:Class ;
    rdfs:label "Description Template" ;
    rdfs:comment """レコード記述規則を表すメタクラス。
    レコード記述規則 D は OWL のクラス（例えば ex:CD）として表現し、その記述
    規則 D にしたがって書かれたメタデータ（例えば ex:MD）は記述規則クラス ex:
    CD のインスタンスとなる（すなわち、ex:CD a owl:Class. であり、ex:MD a
    ex:CD. となる）。
    クラス ex:CD がレコード記述規則であることを分かりやすくするため、OWL ク
    ラスの代わりに DescriptionTemplate のインスタンス（つまり ex:CD a dsp:
    DescriptionTemplate.）として表現してよい。""" ;
    rdfs:subClassOf
        #項目記述規則をサブクラス関係で結ぶ。
        [ a owl:Restriction ;
```

```
        owl:onProperty rdfs:subClassOf;
        owl:onClass dsp:StatementTemplate;
        owl:minQualifiedCardinality 0 ] .


dsp:StatementTemplate a owl:Class ;
    rdfs:subClassOf owl:Restriction ;
    rdfs:label "Statement Template" ;
    rdfs:comment """項目記述規則を表すメタクラス。
    項目記述規則 S は OWL のクラス制約（例えば ex:RS）として表現し、その記述
    規則 S を持つレコード記述規則 D のクラス（例えば ex:CD）は、ex:RS のサブク
    ラスとなる（すなわち、ex:RS a owl:Restriction. であり、ex:CD rdfs:
    subClassOf ex:RS. となる）。レコード記述規則 D1 が項目記述規則 S1、S2、
    S3 を持つならば、それぞれのクラス、クラス制約の関係は ex:CD1 rdfs:
    subClassOf ex:RS1, ex:RS2, ex:RS3. となる。レコード記述規則 D1 に従うメ
    タデータは、項目記述規則の制約 S1、S2、S3 をすべて満たすもの（ex:RS1,
    ex:RS2, ex:RS3 の共通部分）だからである。
    クラス制約 ex:RS が項目記述規則であることを分かりやすくするため、OWL ク
    ラス制約の代わりに StatementTemplate のインスタンス（つまり ex:RS a dsp:
    StatementTemplate.）として表現してよい。""" ;
    rdfs:subClassOf
        #対象となるプロパティを必ず 1 つ定義する。
        [ a owl:Restriction ;
        owl:onProperty owl:onProperty;
        owl:cardinality 1 ] .


##プロパティ定義
dsp:valueURIOccurrence a owl:DatatypeProperty ;
    rdfs:label "Value URI Occurrence" ;
    rdfs:comment """レコードを空白ノードとできるかどうかを示す。プロパティ
    値が mandatory なら URI 必須、optional なら空白ノード可、disallowed なら常
    に空白ノード。このプロパティを持たない場合は、optional であるのと同等。
    """ ;
    rdfs:domain dsp:DescriptionTemplate ;
    rdfs:range [ a owl:DataRange ;
        owl:oneOf("mandatory" "optional" "disallowed")
    ] .

dsp:inScheme a owl:ObjectProperty ;
    rdfs:label "In Scheme" ;
```

```
    rdfs:comment """クラスのメンバーが、目的語で示される語彙（シソーラス
    など）の概念の集合で構成されることを表す。たとえば、国立国会図書館件
    名標目表（NDLSH）に含まれるそれぞれの件名を、「NDLSH 語彙に属する」と
    いう制約で表現される匿名クラスのインスタンスと考えると便利な場合があ
    る。このプロパティは、このクラス制約を [dsp:inScheme ndlsh: ] と簡易に
    表現するために用いる。このプロパティで表現されたクラス制約について、
    推論 {ex:CR dsp:inScheme ndlsh: .} => {ex:CR owl:onProperty skos:
    inScheme; owl:allValuesFrom ndlsh: .} が成り立つ。
    項目記述規則において、dc:subject の値制約として NDLSH 語彙を指定する場
    合、[dsp:StatementTemplate; owl:onProperty dc:subject; owl:onClass
    [dsp:inScheme ndlsh:]] という表現ができる。複数語彙が許される場合、
    制約に用いるクラスを [owl:unionOf([dsp:inScheme ndlsh:] [dsp:inScheme
    bsh:])] のように和集合クラスとする。
    DCMI-DSP の vocabularyEncodingScheme に近い。""" ;
    rdfs:domain rdfs:Class ;
    rdfs:range skos:ConceptScheme .


dsp:resourceClass a owl:ObjectProperty ;
    rdfs:label "Resource Class" ;
    rdfs:comment """レコード記述規則によって記述したメタデータインスタン
    スは、このクラスのメンバーとなることを示す。DCMI-DSP の resourceClass と
    同等。""" ;
    rdfs:domain dsp:DescriptionTemplate ;
    rdfs:range rdfs:Class .

dsp:cardinalityNote a owl:DatatypeProperty ;
    rdfs:label "Cardinality Note" ;
    rdfs:comment """項目記述規則の出現回数制約のうち、「推奨」「あれば必
    須」など数値表現できない制約を記述する。このプロパティがある場合、owl
    :minCardinality は 1 と解釈される。""" ;
    rdfs:domain dsp:StatementTemplate .


##Testing

dsp:langTagOccurrence a owl:DatatypeProperty ;
    rdfs:label "Language Tag Occurrence" ;
    rdfs:comment """項目記述規則の値制約がプレーンリテラルの場合、言語タ
    グが必須（mandatory）か、任意（optional）か、不可（disallowed）かを示
    す。値がプレーンリテラルとならない項目規則で用いた場合はエラー。""" ;
```

```
    rdfs:domain dsp:StatementTemplate ;
    rdfs:range [ a owl:DataRange ;
        owl:oneOf("mandatory" "optional" "disallowed")
    ] .

dsp:perLangMaxCardinality a owl:DatatypeProperty ;
    rdfs:label "Per Language Max Cardinality" ;
    rdfs:comment """項目記述規則のプロパティが、1 つの言語タグあたり最大
    何回出現できるかを制約する。特にプロパティ値の読みを言語タグを用いて
    表現する場合、この値を 1 と制約することで、プロパティ値文字列と読みが 1
    対 1 に対応することを保証する。値がプレーンリテラルとならない項目規則で
    用いた場合はエラー。""" ;
    rdfs:domain dsp:StatementTemplate ;
    rdfs:range xsd:nonNegativeInteger .


dsp:propertyMapping a owl:ObjectProperty ;
    rdfs:label "Property Mapping" ;
    rdfs:comment """項目記述規則のプロパティP を、ダムダウン（単純化）用
    に汎用上位プロパティQ と関連付ける。通常は P rdfs:subPropertyOf Qp. 関係
    を用いてダムダウンを行なうが、この規則においては Qp とは異なるプロパティ
    とマッピングしたい場合や、P は外部で定義された語彙から利用しており勝
    手に subPropertyOf 関係を加えられない場合などに用いる。したがって、{
    [a dsp:StatementTemplate; onProperty any:P; dsp:propertyMapping ex:Q]
    } => {any:P rdfs:subPropertyOf ex:Q} が暗示される。""" ;
    rdfs:domain dsp:StatementTemplate ;
    rdfs:range rdf:Property .
```

## 6.2 簡易 DSP ファイルの記述方法

### 6.2.1 ファイル構成

- 簡易 DSP は、タブ区切りによるテキストファイルとして記述する。ファイルの文字コードは UTF-8 とする。
- テキストファイル内は、1 つ以上の記述規則ブロック（およびオプションの名前空間宣言ブロック）で構成する。
- ブロックは [] でブロック ID（レコード記述規則 ID）を示して開始する。名前空間宣言ブロックは、予約 ID として@NS を用いて示す。
- 区切りなどのために空行をおいてよい。空行は処理上無視する。

- 行頭が#で始まる行はコメント行とする（行の途中に#があっても、それ以降をコメント扱いはしない）。

最初のレコード記述規則 ID は、MAIN とする。
記述規則定義において修飾名（名前空間接頭辞）を用いる場合、標準接頭辞以外の接頭辞が必要であれば、先頭（[MAIN] ブロックの前）に名前空間宣言ブロック [@NS] を置く。

### 6.2.2 記述規則ブロック

レコード記述規則 ID（以下 ID）を [] 内に記し、続く行でレコード内に含まれる項目記述規則を、1 項目 1 行で列挙する。

- レコード規則 ID には、英数字もしくはカナ漢字を用いることができる（ただし最初の文字を数字にすることはできない）。空白文字、記号は ID に用いない。

```
[MAIN]
#項目規則名 プロパティ...
（以下に書誌の項目記述規則）\\
```

※項目記述規則の 1 行目を各列の名前（ラベル）とする場合は、行頭に#を置いてコメントとする必要があることに注意

### 6.2.3 名前空間宣言ブロック

6.2.6 で定義する標準名前空間以外の名前空間を用いる場合は、記述規則ブロックの前に名前空間宣言ブロックを置いて、接頭辞と名前空間 URI のマッピングを宣言する。
名前空間宣言ブロック ID（[@NS]）を最初に記し、続く行で接頭辞と URI の組を TAB で区切って列挙する。宣言の 1 行目には、各列のラベルをおいてよい。

```
[@NS]
#接頭辞　名前空間 URI
sioc     http://rdfs.org/sioc/ns#
rda      http://RDVocab.info/ElementsGr2/
```

- 接頭辞には英数字のみを用いる。最初の文字を数字にすることはできない。
- 記述規則自身の名前空間を定義する場合は、接頭辞を@base として記述する。@base がない場合の記述規則名前空間は、レジストリが割り当てる。

6.2.6 で定義する標準名前空間 URI を、標準設定と異なる接頭辞に結びつける宣言をしてもよい。
※@base で指定する URI は、「記述規則の名前空間」であって、記述規則を用いて生成する「メタデータインスタンス」のデフォルト名前空間ではないことに注意。

### 6.2.4 項目記述規則

項目記述規則は 1 行 1 項目とし、各行は次の要素を TAB 区切りで並べる。

- 項目規則名
- プロパティの修飾名
- 最小出現回数
- 最大出現回数
- 値タイプ
- 値制約
- コメント（説明）

値タイプを `ID` とする項目記述規則は、レコード自身の扱いを定める特殊な規則とする。

**項目規則名（規則ラベル）**　項目記述規則に与える識別名で、データに対する「タイトル」「著者」などといった項目名（列名、フィールド名）と同等。

項目記述規則を URI として表現するために用いるので、レコード規則 ID と同様の文字で構成する（英数字もしくはカナ漢字を用いることができるが、最初の文字を数字にすることはできない。空白文字、記号は用いない）。

※項目規則名に空白文字を用いた場合は、URI では_に変換する。記号は`%HH` の形に URL エンコードされる（ただし・については、_に変換する）。

**プロパティの修飾名**　規則に従って記述された項目を RDF に変換するために用いるプロパティを、修飾名で記述する。たとえば「タイトル」項目をシンプル・ダブリンコアの `title` とする場合は、ここに「`dc:title`」と記す。

この修飾名に用いる接頭辞は、6.2.6 の標準接頭辞、もしくは 6.2.3 で定義したものとする。

※接頭辞のないプロパティを記述した場合は、独自語彙によるプロパティとして、レジストリ登録時に語彙も合わせて定義・登録するものとする。

**出現回数制約**　レコード内における項目の出現回数について、最小回数、最大回数の 2 欄に記述する（値は 0 または正の整数、またはキーワード）。最大回数の制約がない場合は-とする。

出現回数が「あれば必須」「推奨」などの場合は、そのキーワードを最小回数欄に記述し、最大出現回数を-とする（ただしコンピュータによるチェック上は、任意と同じ）。

**値タイプおよび値制約**　項目の値は、表 15 のいずれかのタイプとする。

それぞれのタイプについて、以下の規則に基づく値制約を記述する。ただし値タイプ＝ **ID** については、値制約欄を異なる目的で使用する（次項 6.2.4 を参照）。

（値制約で修飾名を用いる場合の接頭辞も、プロパティ欄と同様、6.2.6 の標準接頭辞、もしくは 6.2.3 で定義したものとする）

表 14: 出現回数制約の記述例

| 最小出現回数 | 最大出現回数 | 意味 |
|---|---|---|
| 0 | 1 | 記述は任意で、最大 1 回 |
| 1 | - | 必須で、何回記述してもよい |
| 1 | 1 | 必須で、必ず 1 回 |
| 0 | - | 制約なし（任意で、何回出現してもよい） |
| 推奨 | - | 推奨（検証上は任意と同等） |

表 15: 簡易 DSP で用いる値タイプ

| 値タイプ | 記述する値 |
|---|---|
| ID | レコードの ID を記述する |
| 制約なし | 任意の値を記述してよい |
| 文字列 | リテラル値を記述する |
| 構造化 | 入れ子記述に相当するリソース |
| 参照値 | URI による外部参照リソース |

**値タイプ＝文字列の場合**　値制約欄は次のように記述する：

- プレーンリテラル（任意の文字列）の場合は空欄。
- リテラルのデータ型（datatype）を制約する場合は、データ型修飾名。複数のデータ型を使ってよい場合は、修飾名を空白区切りで列挙。
- 特定の文字列から選択する場合は、選択肢を引用符""で囲み、空白区切りで列挙。

表 16: 文字列タイプ値制約の記述例

| 値制約 | 意味 |
|---|---|
| | 任意の文字列 |
| `xsd:decimal` | 十進数 |
| `"バナナ" "りんご" "みかん"` | 「バナナ」「りんご」「みかん」のいずれか |

**値タイプ＝構造化の場合**　値制約欄は次のように記述する：

- 値が同じ記述規則ファイル内のレコード記述規則で定義されるレコード（入れ子を含む）である場合は、「`#規則 ID`」で示す。
- 値が特定のクラスインスタンスであればよい場合は、「クラス修飾名」を直接記す。この場合は、構造化値内の項目記述は制約を持たない（制約を与える場合は注参照）。
- 複数のクラスから選んでよい場合は、クラス修飾名を空白区切りで列挙。

表 17: 構造化タイプ値制約の記述例

| 値制約 | 意味 |
|---|---|
| #構造化タイトル | 同じファイル内の[構造化タイトル] ブロックで定義される値 |
| foaf:Agent | 値はfoaf:Agent クラスのインスタンス |
| foaf:Person rda:Family | foaf:Person もしくはrda:Family のインスタンス |

※構造化値にクラスと項目記述両方の制約を与えたい場合は、#規則 ID 型を用い、6.2.4 に従って [規則 ID] ブロックの ID 型記述規則のプロパティ欄にクラスを指定する。[規則 ID] ブロックに ID 型規則持たせずに（空白ノードのまま）クラスを与える定義は、この簡易 DSP ではサポートしない（DSP 定義言語で直接記述することはできる）。

**値タイプ＝参照値場合**　値制約欄は次のように記述する：

- 任意の実体（URI）を値とする場合は空欄。
- 特定の語彙に属する URI を用いる場合は、語彙の「名前空間接頭辞:」で示す（接頭辞は最後に :を置く）。
- 複数の語彙から選んでよい場合は、語彙の「名前空間接頭辞:」を空白区切りで列挙する。
- 特定の URI（語彙ではなく個別名）から選択する場合は、修飾名もしくは<URI>を空白区切りで列挙する。

表 18: 参照値タイプ値制約の記述例

| 値制約 | 意味 |
|---|---|
| | 任意の URI |
| ndlsh: | NDLSH **語彙**に属する URI が記述できる |
| ndlsh: bsh: | NDLSH もしくは BSH **語彙**に属する URI |
| card:VISA card:AMEX | URI（修飾名）card:VISA、card:AMEX のいずれか |

※値制約を ndlsh:とした場合、実際のデータには「ndlsh:図書館」など、その名前空間 URI で表される語彙に属する URI が記述できる（データが ndlsh:となるのではないことに注意。また、一般に個々の URI は ndlsh:名前空間に属すると考えられるが、名前空間が異なる場合もある）。
※特定の URI を記述する場合は<>で囲むことに注意。

**ID 項目規則**　値タイプ= **ID** の項目記述規則は、ID 欄を定義すると同時に、レコード自身の扱いも定義する。

- 項目規則名：レコードの ID を収める列とする。
- プロパティの修飾名：レコードの型（クラス）を示す。たとえばこの値が foaf:Document である場合、レコードは文書（foaf:Document クラスのインスタンス）であることを表す。
- 最小出現回数：常に 1

- 最大出現回数：常に 1
- 値タイプ：常に ID
- 値制約：レコード URI の名前空間を示す。たとえばこの値が ndlbooks:であるとき、実際に記述されたデータの ID の値が b01234 ならば、このレコードの URI は ndlbooks:b01234 となる。※この値を用いて、表形式のデータから URI を持つ RDF データを生成できる。ただしこのを指定すると、記述規則の使い方が固定されるので注意。
- コメント（説明）

各レコード記述規則ごとに、ID 型の項目記述規則は 1 つしか指定できない。[MAIN] 規則には、原則として ID 型項目を持たせるものとする。

### 6.2.5 簡易 DSP 記述例

```
[@NS]
dcndl       http://ndl.go.jp/dcndl/terms/
ndlsh       http://id.ndl.go.jp/auth/ndlsh/
bsh         http://id.ndl.go.jp/auth/bsh/
ndlbooks    http://iss.ndl.go.jp/books/
@base       http://ndl.go.jp/dcndl/dsp/biblio

[MAIN]
#項目規則名 プロパティ       最小  最大  値タイプ  値制約           説明
書誌 ID      foaf:Document   1     1     ID        ndlbooks:        文書の ID
タイトル     dcterms:title   1     1     構造化    #構造化タイトル  文書の表題
著者         dcterms:creator 0     1     構造化    foaf:Agent       文書の作者
発行日       dcterms:issued  1     1     文字列    xsd:date         文書の発行日
主題         dcterms:subject 0     -     参照値    ndlsh: bsh:      文書の主題

[構造化タイトル]
#項目規則名 プロパティ       最小  最大  値タイプ  値制約           説明
リテラル値   xl:literalForm  1     1     文字列                     タイトル自身
読み     dcndl:transcription 0     1     文字列                     タイトルの読み
```

### 6.2.6 デフォルト設定

**標準名前空間接頭辞**　頻繁に用いられる名前空間として、表 19 の標準接頭辞マッピングを用意する。

標準マッピングの接頭辞のみを用いる場合、名前空間宣言ブロックは省略してよい。

標準マッピングと異なる接頭辞、URI の組み合わせが名前空間宣言ブロックに記述されている場合は、その記述を優先する（デフォルト宣言を上書きする）。

表 19: 標準接頭辞マッピング

| 接頭辞 | 名前空間 URI |
|---|---|
| dc | http://purl.org/dc/elements/1.1/ |
| dcterms | http://purl.org/dc/terms/ |
| foaf | http://xmlns.com/foaf/0.1/ |
| skos | http://www.w3.org/2004/02/skos/core# |
| xl | http://www.w3.org/2008/05/skos-xl# |
| rdf | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs | http://www.w3.org/2000/01/rdf-schema# |
| owl | http://www.w3.org/2002/07/owl# |
| xsd | http://www.w3.org/2001/XMLSchema# |

**レコード記述規則 ID の省略**　名前空間宣言がない場合、主規則はレコード記述規則 ID[MAIN] の行を省いて、項目規則のみを列挙できる（つまり、入れ子規則がない記述規則は、項目規則の表のみで示すことができる）。

## 6.3 OWL 記述例

6.2.5 の簡易 DSP 記述例を OWL-DSP に変換し、メタデータを加えた RDF は次のようになる。

```
@prefix bsh: <http://id.ndl.go.jp/auth/bsh/>.
@prefix ndlbooks: <http://iss.ndl.go.jp/books/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix dsp: <http://purl.org/metainfo/terms/dsp#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix dcndl: <http://ndl.go.jp/dcndl/terms/>.
@prefix reg: <http://purl.org/metainfo/terms/registry#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix xl: <http://www.w3.org/2008/05/skos-xl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ndlsh: <http://id.ndl.go.jp/auth/ndlsh/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@base <http://ndl.go.jp/dcndl/dsp/biblio>.

<> a owl:Ontology ;
    rdfs:label "国立国会図書館書誌記述" ;
    reg:created "2011-01-15" ;
    reg:creator ex:aRegistoredUserId ;
    reg:version "第 1.1 版" .
```

```
<#MAIN> a dsp:DescriptionTemplate ;
    dsp:valueURIOccurrence "mandatory" ;
    dsp:resourceClass foaf:Document ;
    reg:idField "書誌 ID" ;
    reg:resourceNsURI ndlbooks: ;
    rdfs:subClassOf <#MAIN-タイトル>, <#MAIN-著者>, <#MAIN-発行日>, <#MAIN-主題> .

<#MAIN-タイトル> a dsp:StatementTemplate ;
    rdfs:label "タイトル" ;
    owl:onProperty dcterms:title ;
    owl:qualifiedCardinality 1 ;
    owl:onClass <#構造化タイトル> ;
    rdfs:comment "文書の表題" .

<#MAIN-著者> a dsp:StatementTemplate ;
    rdfs:label "著者" ;
    owl:onProperty dcterms:creator ;
    dsp:cardinalityNote "あれば必須" ;
    owl:onClass foaf:Agent ;
    rdfs:comment "文書の作者" .

<#MAIN-発行日> a dsp:StatementTemplate ;
    rdfs:label "発行日" ;
    owl:onProperty dcterms:issued ;
    dsp:cardinalityNote "あれば必須" ;
    owl:maxQualifiedCardinality 1 ;
    owl:onDataRange xsd:date ;
    rdfs:comment "文書の発行日" .

<#MAIN-主題> a dsp:StatementTemplate ;
    rdfs:label "主題" ;
    owl:onProperty dcterms:subject ;
    owl:onClass [owl:unionOf(
        [dsp:inScheme ndlsh:]
        [dsp:inScheme bsh:]
    )] ;
    rdfs:comment "文書の主題" .


<#構造化タイトル> a dsp:DescriptionTemplate ;
    rdfs:subClassOf <#構造化タイトル-リテラル値>, <#構造化タイトル-読み> .
```

```
<#構造化タイトル-リテラル値> a dsp:StatementTemplate ;
    rdfs:label "リテラル値" ;
    owl:onProperty xl:literalForm ;
    owl:qualifiedCardinality 1 ;
    owl:onDataRange rdfs:Lieral ;
    rdfs:comment "タイトル自身" .

<#構造化タイトル-読み> a dsp:StatementTemplate ;
    rdfs:label "読み" ;
    owl:onProperty dcndl:transcription ;
    owl:maxQualifiedCardinality 1 ;
    owl:onDataRange rdfs:Lieral ;
    rdfs:comment "タイトルの読み" .
```