2015 年 10 月版

OAI-PMH の要点

本資料に関するお問合せ先：

国立国会図書館　電子情報サービス課
情報アクセス提供係
ndl-search@ndl.go.jp

OAI-PMH の要点

イントロダクション

このドキュメントは OAI-PMH の概要と実装上の留意点などを説明した資料です。プロトコルの仕様については、Open Archive Initiative[1]のプロトコル本体を参照してください。プロトコル本体を読み解くにあたって本文書が理解を深める助けになると幸いです。

このドキュメントは上述のとおり説明資料であり、OAI-PMH の規定を新たに定めたり、開発ベンダや連携機関等に記載どおりの実装を強制したりする性質の文書ではありません。

また、国立国会図書館サーチでも本ドキュメント通りの実装となっていない個所があり、今後改善を目指していきます。

† 「OAI-PMH の概観」に書いてあること

汎用的な説明を記述していますが、OAI-PMH でリポジトリを作成しようとする機関向けの説明文書として利用することを想定しています。

- 設計や要件定義を行う上での第一歩となる、OAI-PMH の概念的な位置づけ
- ResumptionToken によるフロー制御（例外処理などは捨象しています）
- 差分更新の流れ
- Set によるレコードの分類

† 「実用的なリポジトリの実装」に書いてあること

第二部では実際のアプリケーションの構造について実例を交えて解説します。

- 設計・実装における注意点や Tips
- 実用的なリポジトリの設計・実装例

† ドキュメントの対象者

- OAI-PMH を実装しようとするシステムの開発・設計担当者
- 図書館・アーカイブ等職員
  - XML、HTTP-GET パラメータの初歩を前提知識としていますが、必須ではありません。第二部では技術的な用語が多く出現します。要件定義の担当者などでなければ全てを理解する必要はないと考えています。

---

[1] http://www.openarchives.org/OAI/openarchivesprotocol.html

1. はじめに

OAI-PMH は Open Archive Initiative によって開発されたメタデータ交換のプロトコルで、2002 年 6 月に現行の第 2 版が公表されて以降、国内外で広く採用されてきた[2]。DSpace[1][2]をはじめとした主要な機関リポジトリソフトウェアではパッケージの段階で実装されており[3]、国立情報学研究所が運営・提供する共用リポジトリサービスである JAIRO Cloud[4]においても実装されている。しかしながら日本国内の他の領域（デジタルアーカイブや公共図書館の目録システムなど）においては、個別既存システムの改修などで OAI-PMH のインタフェースを実装していることが多い。

OAI-PMH については、メタデータのみの交換を目的としたプロトコルであることからその限界も指摘されているが[3]、図書館システムなどメタデータのみを交換する上では必要十分であるともいえる。日本では、国内図書館の総合目録としての機能を担っている国立国会図書館サーチにおける平成 27 年から 5 年間の連携方針について示した実施計画[5]、「国立国会図書館サーチ連携拡張に係る実施計画」の中で OAI-PMH での連携を推進する姿勢を明確に打ち出しており、今後、都道府県立および政令指定都市立の図書館システムにおいても OAI-PMH インタフェースの実装が進んでいくものと考えられる。

そうした中で、これまで OAI-PMH についての日本語のドキュメントはあまり普及していなかった。実際の設計等の際に参照可能なものとしては、国立情報学研究所が公開している OAI-PMH のプロトコル日本語訳[6]および実装ガイドライン日本語訳[7]があるのみと言って良いほどである。

本稿では図書館職員やデジタルアーカイブ運用担当者のために、前半で OAI-PMH の概要と位置づけについて解説する。後半では要件定義担当者や開発者がリポジトリの設計や実装を行う上での注意点を挙げ、まとめとして図書館システムを想定した実用的なリポジトリの設計について述べる。

[2] 本ドキュメントでは、OAI-PMH2.0 を対象としている。

OAI-PMH の概観

2.1 検索サービスシステムの連携方式

検索サービスシステムの連携として、システム A（例：ディスカバリ）とシステム B（例：目録システム）が連携することを考える。連携を行うことで、利用者はシステム A の画面上でシステム A・B 双方のデータを検索できるようになる。一元的な検索が実現されることでユーザの利便性が高まるとともに、利用者動線が増えることで利用増を見込める。

検索サービスシステムの連携は、検索処理を連携元と連携先のどちらで行うかによって、2 つに大別することが出来る。本稿ではこの 2 つを**動的連携**と**静的連携**と呼ぶ。システム A・B の分担を(*a*)データベースの更新、(*b*)検索処理の実行、(*c*)画面表示、の 3 段階で考えたとき、(*b*)検索処理をシステム A とシステム B で分散する連携方式を動的連携、システム A で担う連携方式を静的連携と呼ぶ（図 1）。

図 1 連携方式の分類（動的連携と静的連携）

† 動的連携 … 別称：横断検索

動的連携では、利用者がシステム A の検索窓に検索語を入れて検索したときに、動的に検索式・クエリを生成して、システム B の API へリクエストし返戻結果をシステム A で表示する。動的連携は横断検索などとも呼ばれ、代表例として、SRU[8]などがある。

† 静的連携 … 別称：ハーベスティング

静的連携の場合、定期的（例：日次や週次）にシステム B にあるデータをシステム A に複製しておいて、検索処理はシステム A の内部のみで行う。検索処理が 1 システムのため、ランキングロジック[3]を統一することができるなどのメリットがある。OAI-PMH は静的連携のプロトコルである。静的連携は、収集(API)、ハーベスティング、などとも呼ばれる。

---

[3] 「関連度順」などを定めるスコアの算出方法。検索処理を行うシステムの中での事前計算が必要なため、統一するには検索される前から全データを持っている必要がある。

2.2 OAI-PMH の特徴とリクエスト

OAI-PMH では定期的な更新を行うことでデータベースの複製ができる。リクエストとレスポンスの形式についての特徴は以下の通り。

OAI-PMH の特徴

- HTTP 上に規定

OAI-PMH は HTTP 上のプロトコルとして規定されている。図書館システムの API としては Z39.50[9]など HTTP を使うことが規定されていないプロトコルもあるが、それらと比べて一般的な通信規約の上で開発ができ、環境構築などが容易であると言える。

- サーバクライアント型

データを取得する側と提供する側で振る舞いがはっきりと異なるサーバクライアント型のプロトコルである。ディスカバリサービスなどのデータを取得する側がクライアント、公共図書館の目録システムなど個別データを提供する側がサーバとなる。クライアント側はハーベスタ、サービスプロバイダなどと呼ばれ、サーバ側はリポジトリ、データプロバイダなどと呼ばれる。本稿においても、以降「ハーベスタ」「リポジトリ」の語を用いる。

- リクエストは URL でパラメータを指定

ハーベスタからのリクエストは HTTP-GET のパラメータとして、URL に付してリクエストされる。セッション管理[4]やユーザ識別等は行わない[5]ため、GET でのパラメータ指定[6]のみで全てのリクエスト情報が完結することが特徴である。（リポジトリの返戻は、メタデータに変更がない場合は、ハーベスタからのリクエスト URL のみによって一意に定まる。）

---

[4] Web ブラウザから Web サイトなどにアクセスしたときに設定情報などを保持するのに用いられている仕組み。多くのハーベスタは Web ブラウザなどを介さずサーバから直接リクエストすることから、リポジトリでセッション管理などを行うとハーベスタは対応できないことが多い。

[5] API キーなどを埋め込むことは考えられるが、これは OAI-PMH のアプリケーションとは別のレイヤーで実現されることである。

[6] URL の末尾に「?項目名 1=値 1&項目名 2=値 2 …」という形式で情報を付加してサーバと連携する方法

◦ レスポンスは XML としてのみ返戻

リポジトリからのレスポンスは基本的に XML 形式である。ハーベスタは引数を GET パラメータとして URL を生成しリポジトリにリクエストし、レスポンスされた XML を解析することで実装できるため、各 OS や言語標準の API を用いることで容易に実装可能である。

◦ 出力レコード（メタデータ）は XML 形式であれば OAI-PMH で出力可能

出力するレコードのメタデータは XML で表現することが規定されており、XML 形式であればどんなメタデータ形式でも OAI-PMH で提供することができる（ただし、当該形式のデータをハーベスタが扱えることが必要条件である）。

◦ 出力対象のメタデータを「更新・新規追加」、「削除」の 2 ステータスで出力

ハーベスタは、収集したデータが「更新・新規追加」のレコードであればハーベスタ内の DB で当該レコードを上書き、「削除」であればハーベスタ内の DB から当該レコードを削除する。定期的に収集を行うことで、古いデータは上書き、削除され、収集時点の最新データをハーベスタは常に保持しておくことができる。

OAI-PMH でのリクエスト

前述の通り、OAI-PMH では URL パラメータによってリクエストを行う。ハーベスタからの操作は Verb パラメータによって決定する。Verb パラメータの値は 6 種類の値しか取りえない。リポジトリは各 Verb へのレスポンスを実装することで実現される。

† Identify

- http://example.com/api/oaipmh?verb=Identify

- リクエスト
  Verb の他はなし
- レスポンス
  リポジトリの名称や、リポジトリでのタイムスタンプの精度など

† ListMetadataFormats

- http://example.com/api/oaipmh?verb=ListMetadataFormats

- リクエスト
  Verb の他はなし(2.3 で後述する ResumptionToken を持ちうる)
- レスポンス
  リポジトリから出力できる XML メタデータフォーマット名の一覧

† ListSets

- http://example.com/api/oaipmh?verb=ListSets

- リクエスト
  Verb の他はなし(ResumptionToken を持ちうる)
- レスポンス
  Set(対象データベースのテーブル名のようなもの)の一覧を返戻

上記の 3 つのオペレーション(Identify, ListSets, ListMetadataFormats)では、メタデータの各レコードを更新しても変化がないプロパティ情報であり、実行時の変換や条件分岐は不要なため、DB や設定ファイルなどから値を引き出せば充分であることが多い。

† GetRecord

- http://example.com/api/oaipmh?verb=GetRecord&...

- リクエスト
  Identifier(メタデータの ID)と MetadataPrefix(返戻フォーマット)を指定
- レスポンス
  指定された ID を持つメタデータ(1 件)を指定されたフォーマットで返戻

† ListIdentifiers

- http://example.com/api/oaipmh?verb=ListIdentifiers&...

- リクエスト
  From および Until（取得対象期間）と Set を指定
  （ResumptionToken も持ちうる）
- レスポンス
  指定された対象のメタデータの ID リストのみを返戻

† ListRecords

- http://example.com/api/oaipmh?verb=ListRecords&...

- リクエスト
  From および Until と Set、MetadataPrefix を指定
  （ResumptionToken も持ちうる）
- レスポンス
  指定された対象のメタデータを指定されたフォーマットで返戻

後者の 3 つ(GetRecord, ListIdentifiers, ListRecords)が実際にデータを取得するためのオペレーションであり、リポジトリによるアプリケーションでの制御が必要となる。表の通り、ListIdentifiers が ID リストを取得するオペレーション、GetRecord が ID を基にメタデータを取得するオペレーションである。ListRecords が両者の機能を兼ねたオペレーションで、対象となるメタデータのリストを返戻する。実際にシステム連携を行う際には、多くの場合 ListRecords のみが用いられる。ListRecords の設計が定まると、ListIdentifers、GetRecords の設計も自然と定まるため、本ドキュメントでは、以降 ListRecords を用いた収集について詳述する。

OAI-PMH の概観

2.2 OAI-PMH の特徴とリクエスト

2.1 & 2.2 の まとめ

☐ OAI-PMH は静的連携に分類される
☐ OAI-PMH は HTTP-GET のみでリクエストが完結するプロトコルである
☐ URL にアクセスするだけで情報を得ることが出来る
☐ 返戻された XML をハーベスタが上書き保存することでデータを更新していく
☐ 削除データも連携するため、リポジトリの削除レコードをハーベスタに反映できる
☐ OAI-PMH の各オペレーションでできることは限定的で、多くの場合 ListRecords オペレーションのみで収集が行われる

2.3 ListRecords によるデータ取得とフロー制御

ListRecords リクエストには以下のような 5 種類のパラメータがある。
?verb=ListRecords&metadataPrefix=…&from=…&until=…&set=…
?verb=ListRecords&resumptionToken=…

ListRecords のリクエスト

† MetadataPrefix

返戻データのメタデータ部分の XML フォーマットを指定する。リポジトリはハーベスタの要求に合わせた返戻を可能とすることで、様々なハーベスタに対応できる。特定のハーベスタと連携するのであれば、ハーベスタが要求する XML フォーマットでメタデータを返戻できる必要がある。

† From, Until

返戻対象となる期間範囲を指定する。リポジトリは、この期間内に最終更新日が含まれる（新規追加・更新・削除された）データを対象データとして返戻する。

† Set

リポジトリは、自身の持つメタデータがいくつかの種類に分けられる場合、Set（データ集合）を分けて提供すると、特定の種類のデータのみを利用するハーベスタにとって効率良く連携することができる。例えばアグリゲータ[7]を構築する場合は、収集元ごとに Set を分けることが考えられる。ハーベスタが Set を指定したとき、リポジトリは指定された Set のメタデータのみを対象とし返戻する。ただし、明確なカテゴリ分けがない場合、リポジトリが Set を分けるメリットはない。（Set の設計については 2.4 で詳述する。）

† ResumptionToken

OAI-PMH のリポジトリでは、取得結果が多い場合に、取得結果の一部のみ（不完全リスト）と、ResumptionToken を返戻する。ハーベスタは返戻された ResumptionToken をパラメータとしてリクエストすることで、残りの結果を取得することが出来る。

---

[7] 複数の機関などから、ハーベスタとしてデータを収集し、さらにそのデータをまとめてリポジトリとして提供する役割を持つサービスのこと。

ListRecords のレスポンス

ListRecords のレスポンスは図 2 の形式で返戻される（XML を図式化したもの）。

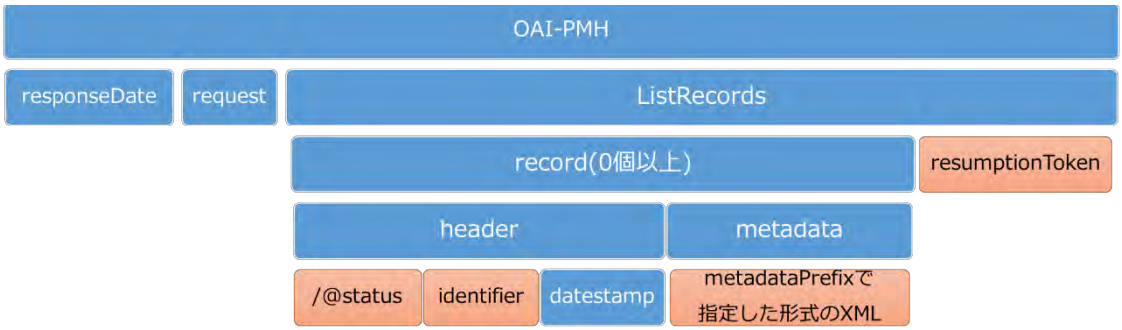

図 2 ListRecords の返戻データ構造イメージ

リポジトリからのレスポンスのうち、

・OAI-PMH/ListRecords/record/header/@status

・OAI-PMH/ListRecords/record/metadata

の 2 項目はレコードが新規追加・更新か、削除かによってタグの有無や値が異なる。

表 1 レコードの種別とタグの値

タグ名(図 2)	status	identifier	datestamp	metadata
新規追加	タグ無し	レコードの ID	追加日付	メタデータが入る
更新	タグ無し	レコードの ID	更新日付	メタデータが入る
削除	“deleted”	レコードの ID	削除日付	タグ無し

ハーベスタでは、identifier の値をキーとして、メタデータの更新を行う。ListRecords の返戻からは、各レコードが新規追加か更新かを区別することはできないが、ハーベスタでは identifier をキーとして一致するメタデータがある場合は上書き、ない場合は新規作成とすれば、リポジトリの最新メタデータ集合と同内容のメタデータ集合を得ることができる。また、status が”deleted”であるレコードを収集した場合には、当該 identifier を持つレコードを削除する。

ResumptionToken によるフロー制御

返戻対象となるレコードが多い場合には返戻対象のうち一部のレコードしか返戻されず、返戻値に ResumptionToken が付与される。ResumptionToken を用いてフロー制御を行うことで、複数回のリクエストによって、完全な対象メタデータのリクエストをやり取りできる。

ResumptionToken による制御のイメージ図を示す（図 3）。

図 3 ResumptionToken によるフロー制御の流れ

1. ハーベスタは、初回のリクエスト時は、ResumptionToken をリクエストパラメータに含めず、MetadataPrefix,From,Until,Set 等を指定してリクエストを行う。
2. リポジトリは、返戻対象件数が一定以下（例えばここでは 200 件とする）でなければ、そのまま返戻する。返戻対象件数が 200 件以上の場合は、record を 200 件返戻して、末尾に 201 件目以降を取得するための ResumptionToken を付してレスポンスする。
3. ハーベスタは返戻された ResumptionToken を付して再リクエストし、201 件目から 400 件目までを取得する。
4. リポジトリは(3)のリクエストを受け付け、201 件目から 400 件目までのレコードと 401 件目以降を取得するための ResumptionToken を付して、返戻する。

全件の取得が完了するまで上記手順の 3.～4.をリポジトリとハーベスタ間で繰り返すことで、OAI-PMH では全件データの取得を行うことができる。

定期的な差分更新

また、OAI-PMH では、ResumptionToken でのフロー制御のみでなく、From,Until の指定による差分更新を組み合わせることよって、ハーベスタは定期的にリポジトリの全件メタデータ情報を更新できる。ResumptionToken と差分更新のイメージ図を示す（図 4）。

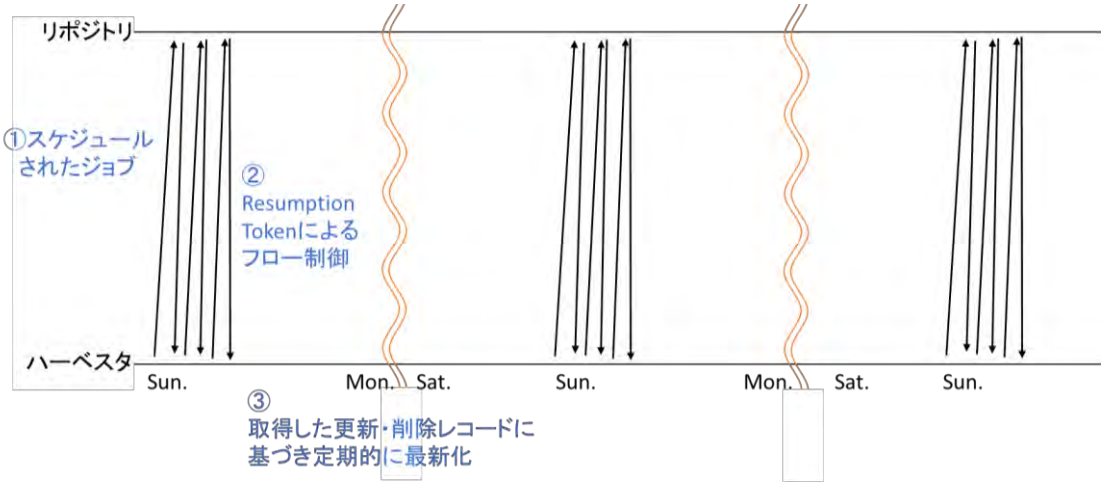

図 4 差分更新のイメージ（日曜に週次更新する場合）

例えば週次で OAI-PMH による更新を行う場合、ハーベスタは from に一週間前の日付 until に当日の日付を指定し「一週間前から今日まで」というような指定をすることで、欠けることなくデータ更新を行うことができる。1 週間分のデータは ResumptionToken によるフロー制御で、すべての更新・削除を取得する。

リポジトリはこれら ResumptionToken による制御と、差分更新によってハーベスタが完全なデータ集合を取得できるよう、設計・実装されなければならない。

トランザクションログではなくダンプ

「差分更新」という言葉や、追加更新削除が出力される、時間単位で切って定期的に更新する、ということから、OAI-PMH はデータベースのトランザクションログ[8]のような理解・設計をされることがある。しかし比喩的な理解としては、データベースのダンプ[9]の方が適切である。トランザクションログのような振る舞い[10]を実装すると、レコード数や計算量上の問題が発生しやすい。トランザクションログではないため、各レコードの最新状態以外を返戻する必要は無く、返戻の順序にとらわれる必要はない。

◦ 各レコードの最新の状態以外は返戻しなくてよい

OAI-PMH で返戻するのは最新のメタデータのみである。例えば From, Until の対象期間内に 2 度更新されていたとしても、中間状態を保持・返戻する必要はない。また、新規登録を 2001-01-01 に行い、2001-01-10 に更新されたデータ A があるとして、ハーベスタから From=2001-01-01, Until=2001-01-07 のリクエストを受け付けたとき、リポジトリはデータ A を返戻しない（してはいけない）。

◦ 返戻の順序に規定はない

返戻順序についてはプロトコル上の規定はないため、最終更新日順である必要はない。すなわち、新規追加・更新レコードと削除レコードを一元的に並べる必要はなく、分けて管理してよい。

2.3 の まとめ

□ ListRecords を用いて、定常収集を実現できる
□ ResumptionToken によるフロー制御と定期的な差分更新によって、差分更新の度にハーベスタではリポジトリの DB を複製することができる
□ リポジトリはフロー制御と差分更新によって、メタデータコレクションの完全な複製ができるように返戻を行う必要がある
□ OAI-PMH はリポジトリの最新断面を提供するもので、トランザクションログではない

---

[8] DBMS(データベース管理ソフトなど)において、どういう操作（レコードの編集・新規追加・削除など）が行われたかを記録するもの。

[9] データベースにある全データをファイルに書き出すこと。またそのファイル。

[10] リポジトリ側で行った操作を、ハーベスタでも再現できるように、操作順に提供するような応答。そうではなく、変更箇所のみのダンプファイルとして、リポジトリが吐き出したデータを、ハーベスタはそのまま上書きする、と捉えるとシンプルに設計できる。

2.4 Set の概要と設計

OAI-PMH において、リポジトリは各レコードを Set という分類で分けることができる。例えば、図書資料と博物資料の Set を分けておくことで、一方のみを利用するユーザにとって利便性が増し、リポジトリにとっても負荷が減ることとなる。リポジトリが"books"の set を用意することで、"books"のメタデータのみを取得したいハーベスタは Set を指定してリクエストして、"books"のデータのみを収集することができる。

http://example/oai?verb=ListRecords&set=books&metadataPrefix=oai_dc

リポジトリは Set が指定された場合はその Set のデータのみを、指定されなかった場合は、リポジトリ全体を対象としてレコードを返戻する。

† Set はレコードに対して変更不可能

OAI-PMH のプロトコル上に明記されていないが、変更されやすい可変のプロパティを基に Set を設定できない。ここで可変のプロパティとは、例えば図書の分類や排架室などを指す。Set はレコードに対して不変のプロパティとして設定する必要がある。また、あるレコードの Set を変更することはできない。Set 変更を行う場合には、新規で OAI-PMH の ID を付与する必要がある。以下に理由を述べる。

良くない例として排架場所を基に Set で分けられるようにしたと仮定する。開架資料と閉架資料で Set=open, Set=close の 2 つの Set で分類することとしたとする。ここで資料 A が開架から閉架に移され、open から close に変わったとき、Set=open として収集しているハーベスタから見ると、資料 A は deleted として扱われなければならない(資料 A の id に対して、delete レコードとして出力する)。そして次に資料 A が閉架から開架に移され close から open に移されると、ハーベスタは一度 deleted として収集した ID に対して更新がかかることになってしまい、収集データの整合性が取れなくなる。

したがって、Set は変更不能なプロパティ[11]に対して設定する必要があり、どうしても Set を変更する必要がある場合には、新規に ID を採番しなおす必要がある。

2.4 の まとめ

□ 適切に Set を設定すれば、利便性向上、リポジトリのサーバ負荷低減を望める。
□ ただし、Set はレコードに対して不変のプロパティとして設定する必要がある。
□ 強いニーズがあり、かつ、データベースや業務フロー上で完全にレコードが分けられる場合には Set による分類が有効だが、むやみに Set を設定するべきではない。

[11]例えば DB テーブル名など、システム上も変更が想定されていない項目。

2.5 正確性と速度

正確性

OAI-PMH は静的連携の API である。差分のみを更新していくため、更新レコードの出力漏れがあると、再度レコードの更新が行われない限り、完全なコレクションに追いつくことはできなってしまう。したがってデータの一部にでも不完全があると、データベース全体のクオリティに長期的な影響を与える。返戻は正確でなければならない。特にフロー制御や差分更新で完全なリストが取得できること（バッチ処理によるデータベースの更新や、バックアップ処理、負荷分散処理による切替えなどを行った場合も、整合した完全なリストが得られること）、エラーが発生したときにエラーとして適切に応答すること（エラー発生時にハーベスタが検知できる形でエラーを返戻しないと、ハーベスタでは不完全なリストであることを検出できないまま収集を完了してしまうため）が重要である。

速度

また、OAI-PMH を設計するにあたっては、常に計算量を意識する必要がある。OAI-PMH のプロトコルドキュメントは、あくまでプロトコル上の交換規約についてのドキュメントであるため、入出力フォーマットしか記載されていないが、特に大規模件数のデータを処理する場合には計算量への注意が必要である。OAI-PMH ではリポジトリのレコード数に比例して、線形[12]のリクエスト数が必要である（レコード数が 10 倍になれば、10 倍の回数 ResumptionToken を用いてアクセスする必要がある）。そのため、1 度の返戻に係る時間計算量がレコード数に対して線形時間のオーダーとなってしまうと、取得全体ではレコード数の自乗に比例する時間が必要となってしまう[13]。理想的には定数時間での返戻が求められ、多くても log(n)程度の計算量で 1 度の返戻が行われないと、データ数が多い場合には実際の運用が困難となってしまう。また、定数部分（XML パース[14]して返戻する部分）なども速度に大きく影響するため、どのようにデータを保持し返戻するかも重要となる（3.5 で詳述する）。

---

[12] 正比例とほぼ同義と捉えてよい

[13] 1 回(リクエスト)あたりの処理時間が対象レコード数に正比例してしまうと、全体では「1 回あたり処理時間(正比例)×リクエスト回数(正比例)」となりレコード数の 2 乗に比例した時間がかかる。データ量が多い場合は、1 回あたりの処理時間を正比例ではなく、例えばレコード数の対数に比例（レコード数が大きくなっても、処理時間は緩やかにしか増加しない）するような設計をしないと実運用に影響がある。

[14] 主に XML について、データ構造を変換すること。ここでは内部データ構造と OAI-PMH での提供用データの変換を指している。

2.5 の まとめ

□　OAI-PMH は完全なリストを得るためのプロトコルであり、リポジトリは完全なリストを出力しなくてはならない
□　入出力がプロトコルに沿うように動作するだけでは実用にならない
□　データ量が多い場合は大量のデータを高速で処理できる設計が必須である

実用的なリポジトリの実装

3. 実用的なリポジトリとは何か

これまで述べてきたとおり、OAI-PMH の長所は、最終更新日を持つことで差分更新を実現できることと、Delete レコードを保証することによってリポジトリ側でのデータ削除をハーベスタ側に反映できることにある。また、2.5 で述べた通り、OAI-PMH では完全性と速度が重要である。ただし、データモデルの設計が悪いと最終更新日・削除データを保持することができない場合や、効率が悪くなってしまう場合がある。

図書館システムなどにおいては、収録件数も多く、期間あたりの更新データ量も大きい。連携開始時には初期データとして数百万件の書誌情報出力が必要となる場合もある。リポジトリからの返戻ボリューム[リクエストを受けてからリポジトリが返戻するデータ量/時間]がデータベースの更新ボリューム[リポジトリ上でのデータベース更新量/時間]を下回ってしまうと、当然全件のデータ取得はできなくなってしまう。

本章では、高速に完全なデータを返戻し、また、Delete レコードを永続的に保持し返戻するために、どのように OAI-PMH リポジトリを実装したらよいかのポイントを解説する。

† この章で記載する内容

設計・実装する上でのポイントとなる項目として、以下の 5 つを解説する。

- 最終更新日の持ち方と、レコードの整列
- 削除レコードのデータの持ち方と管理
- 返戻結果の完全性の担保
- Set に基づく対象レコードの抽出
- 出力する XML データの生成タイミング

また、最後にまとめとして、上記を踏まえたデータ構造とアルゴリズムを示す。

本章では図書館等、中程度以上のデータ量を持つシステムを想定している。データ量が数千件未満のリポジトリや、データ更新が発生しないなどの制約があれば、目的限定的なアドホック実装を行ったほうが効率的な場合もありうることにご留意されたい。

3.1 最終更新日の持ち方と、レコードの整列

まず、最終更新日（OAI-PMH/ListRecords/record/header/datestamp）の保持方法について述べる。この値が From-Until パラメータの指定範囲内にあるかどうかで返戻要否を決定するため、この値は OAI-PMH テーブルでの最終更新日としないとリポジトリとしての整合性が取れなくなってしまう。例えば OPAC システム上の書誌レコードの最終更新日のように、遡及して変更可能な値を入れられるようにしてしまうと、データ出力に抜けや漏れが生じることとなる。

また、最終更新日の値が From-Until パラメータでの検索キーとなるため、高速な処理を行うためには、文字列型などではなく日付型のデータとして管理する必要がある。OAI-PMH での最終更新日はあくまで OAI-PMH のために提供される項目であり、GUI を経由して利用者が閲覧用項目や業務担当者の管理用項目として併用されるべきでない。過去の値を遡及して入れることもないため、UNIX time[15]として保持しても充分である。

例えばリレーショナルデータベース(RDB)[16]を OAI-PMH のレコード管理に用いた場合に、OAI-ID をキーとし、Datestamp にインデクス[17]を設定せずに(図 5 左)From-Until の範囲に含まれる件数を取得するには、N 件(N：レコード数)の走査が必要となるが、インデクスを張っておく(図 5 中)と、探索回数は O(log N)に低下する。

ただし、フロー制御中に DB 更新が発生しても整合性を保って返戻を行う必要があるため、datestamp にインデクスを張るだけの場合は ResumptionToken の中に「次に取得開始する datestamp の値」を含める必要が発生する。主キーとなるレコード更新ごとにユニークな番号を発行し、ResumptionToken のパラメータとすると制御を簡略にできる(図 5 右:青字は物理削除[18]されたレコードを示している)。

なお、RDB を用いる場合は中図と右図で速度は変わらないが、RDB を用いない場合は右図のように datestamp 昇順でレコードを保持すると末尾挿入しか発生しなくなる。このため、平衡二分木[19]を容易に構成でき、各レコード操作は定数オーダーでの処理が可能となる。

---

[15] 1970 年 1 月 1 日を起点として、そこから何秒経過しているか秒単位での時刻を整数で表すコンピュータ内での時刻表現方式。

[16] MySQL, PostgreSQL, SQLite, Oracle など

[17] リレーショナルデータベースにおいて、インデクスとは後から検索や並び替えがしやすいよう、あらかじめ整列情報を付与しておくこと。登録・更新時かかる時間は少し増えるが、検索時にはインデクスを元にあたりを付けて効率よく探索できる。

[18] 物理削除…通常のデータ削除。対義語として論理削除があり、論理削除では実際にデータを行わず、削除されたとマーキングのみを行う方法。

[19] データを高速に検索できるデータ構造。

OAI-ID	datestamp	書誌データ
Oai:example:1	1980-01-01	・・・
Oai:example:2	1980-01-01	・・・
Oai:example:3	2013-09-21	・・・
Oai:example:4	2000-03-04	・・・
Oai:example:5	1999-09-09	・・・
Oai:example:6	2014-11-18	・・・
Oai:example:7	2012-10-05	・・・
Oai:example:8	1989-12-20	・・・
Oai:example:9	2004-12-23	・・・

OAI-ID	datestamp	書誌データ
Oai:example:1	1980-01-01	・・・
Oai:example:2	1980-01-01	・・・
Oai:example:8	1989-12-20	・・・
Oai:example:5	1999-09-09	・・・
Oai:example:4	2000-03-04	・・・
Oai:example:9	2004-12-23	・・・
Oai:example:7	2012-10-05	・・・
Oai:example:3	2013-09-21	・・・
Oai:example:6	2014-11-18	・・・

#	OAI-ID	datestamp	書誌データ
1	Oai:example:1	1980-01-01	・・・
2	Oai:example:2	1980-01-01	・・・
3	Oai:example:3	1980-01-01	・・・
4	Oai:example:2	1989-12-20	・・・
5	Oai:example:4	1990-03-05	・・・
6	Oai:example:5	1999-09-09	・・・
7	Oai:example:6	2000-03-04	・・・
8	Oai:example:7	2004-12-23	・・・
9	Oai:example:4	2012-10-05	・・・

図 5 データ整列のイメージ

3.2 削除レコードのデータの持ち方と管理

削除レコードの管理方法ついて、直観的には(1) OAI-PMH 用の最終更新日などのレコード情報を管理しているテーブルにフラグとなる列を追加して論理削除を行う方法と、(2)Delete レコードを別テーブルで管理する方法が考えられる。このうち、(1)は処理が複雑となるため、(2)での実装を行った方が潜在リスクを下げることができ実装も容易となる。

(1)の場合は、データベース等からデータを取得した後、レコードのステータスを見て処理が分岐することとなり、制御が複雑である。また、カラム[20]が増え、書誌データへの参照を持つことが必須ではなくなるなど、潜在的リスクが増す要因となってしまう。

#	OAI-ID	datestamp	書誌データ	Deleted
1	Oai:example:1	1980-01-01	・・・	
2	Oai:example:2	1980-01-01	・・・	
3	Oai:example:3	1980-01-01	・・・	
4	Oai:example:2	1989-12-20	null	Deleted
5	Oai:example:4	1990-03-05	・・・	
6	Oai:example:5	1999-09-09	・・・	
7	Oai:example:6	2000-03-04	・・・	
8	Oai:example:7	2004-12-23	・・・	
9	Oai:example:4	2012-10-05	・・・	

図 6 削除データの持ち方（良くない例）

(2)の場合はテーブルごとに処理を単線化でき、制御フローをシンプルにすることが出来る。2.3 で述べた通り、レコードの返戻順序に規定はないため、先に削除されていないレコードを返戻し、後から削除レコードの返戻を行ってよい。また、削除されていないレコードと削除レコードは特性が大きく異なる。削除されていないレコードについては更新の可能性があるが、Delete レコードの場合は一度発行されたらその後は固定となるべきである。そのため RDB 等の場合はテーブルを分けた方が DB チューニング[21]なども行いやすい。

#	OAI-ID	datestamp	書誌データ
1	Oai:example:1	1980-01-01	・・・
2	Oai:example:2	1980-01-01	・・・
3	Oai:example:3	1980-01-01	・・・
4	Oai:example:2	1989-12-20	
4	Oai:example:4	1990-03-05	・・・
5	Oai:example:5	1999-09-09	・・・
6	Oai:example:6	2000-03-04	・・・
7	Oai:example:7	2004-12-23	・・・

#	OAI-ID	datestamp
1	Oai:example:2	1989-12-20

図 7 削除レコードの持ち方（青字灰字は物理削除されたレコードを指す）

[20] RDB でのデータ項目。データ項目が増えれば、バグ発生率も高くなる。

[21] 特性に合わせて、操作や検索の順番などを変更し効率化すること。

3.3 返戻結果の完全性の担保

OAI-PMH での返戻結果は完全性を持つ必要がある。完全なリストを出力するために気を付けるべき点として、(1)ResumptionToken によるフロー制御の完全性、(2)データ更新を跨ぐ場合における返戻の完全性、(3)定期収集での返戻における完全性、がある。

(1) ResumptionToken によるフロー制御の完全性

リポジトリは ResumptionToken によるフロー制御の際に、データを漏れなく返戻する必要がある。そのためにはデータの返戻順は一意である必要がある。例えば更新日順で返戻をする場合、同一更新日のデータがあっても一意な順番で返戻が実現されなければならない。取得対象の指定内容によって、リスト中の同一返戻順のスコア要素が連続した箇所[22]でリストを切断して ResumptionToken を出力した場合でも、次の（ResumptionToken で要求された）リクエストに対して確実に完全なリストを提供する必要がある。RDB 等を利用せず、検索エンジン[23]などで OAI-PMH を実装する場合も同様は特に注意が必要である。

(2) データ更新を跨ぐ場合における返戻の完全性

ResumptionToken での返戻開始位置取得は返戻順位中の一意な箇所を指し示す必要がある。これは、「対象となるレコードのうち、X 件目からを取得する」というような指定方法だと、取得済みデータが取得対象レコードから外れたときに取得データに漏れが生じるためである。

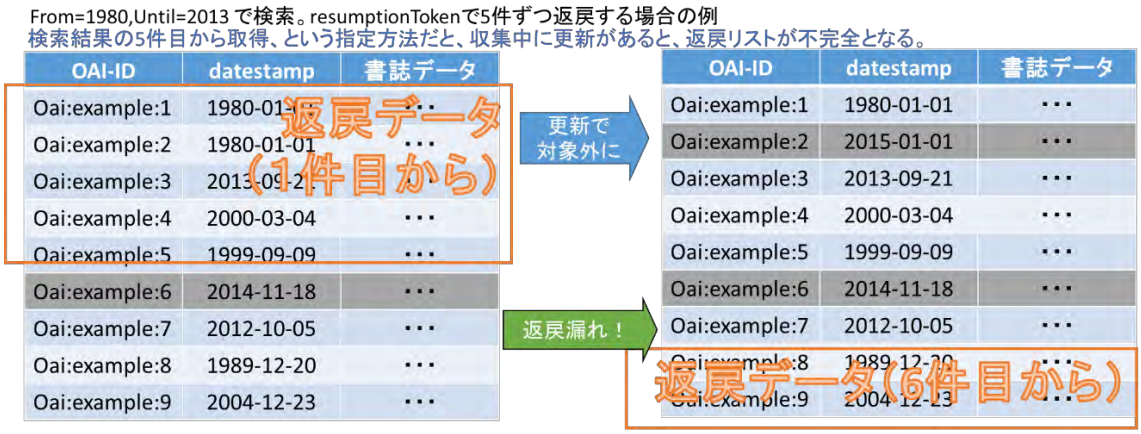

図 8 データ更新によって、出力漏れが発生してしまう例

---

[22] 出版年で排列したときの、同一出版年の本など。対象データを抽出するたびに順番が入れ替わってしまうと、ResumptionToken で指定したときに漏れが生じうる。

[23] ここでは Apache Lucene や Apache Solr、Sedue といった、システムに組み込んで使う検索プログラムのことを指している。

（3）定期収集での返戻における完全性

ハーベスタが From-Until パラメータで収集するときに漏れが発生しないように、実際に OAI-PMH でメタデータが返戻され始める日時と、datestamp の値は一致している必要がある。基本的に datestamp の値が遅い（新しい）分には完全性への影響は少ないため、datestamp の値が先行しないように特に気を付けなければならない。例えばリポジトリ内部でのデータ更新がバックアップのためリアルタイムではなく、1 日遅れて OAI-PMH では提供される、と言った場合には、OAI-PMH で提供される日付が設定される必要がある。

3.4 Set に基づく対象レコードの抽出

Set には可変なプロパティ（メタデータ中のキーワードなど）で設定してはいけない（2.4 で前述）。Set を実装する際には、Set の包含関係から集合を排他的に分割して、それぞれ異なるテーブル管理を実装することで、制御をシンプルにすることができる。仮に 2 つの Set を提供し、かつそれぞれが排他的ではない場合には、4 つの集合（A のみに属する, B のみに属する, A・B 両者に属する ,いずれにも属さない）に分割し、4×2(削除レコードの分) = 8 テーブルに分けることが望ましい。例えば Set として、A が指定された場合には、リポジトリは A のみに属するレコードのテーブルと、A・B 両者に属するレコードのテーブルから値を返戻すればよい。

3.5 出力する XML データの生成タイミング

OAI-PMH は全件データを提供するためのプロトコルであり、OAI-PMH を実装する目的を考えれば、すべてのデータが少なくとも一度は出力されることになるだろう。もしメタデータの項目を RDB で管理し返戻時に XML を作成する手法を採用していると、出力 XML を何度も生成しなくてはならなくなってしまう。特にリッチかつ複雑なメタデータ（DC-NDL（RDF）など）で返戻する場合は、リクエスト毎に各レコードの XML を毎回生成すると応答速度低下の大きな要因となる。OAI-PMH 用のテーブル等に登録する段階であらかじめ XML を生成し、返戻時にはテキスト操作のみで返戻を行うようにすれば、速度は大きく向上する。

OAI-PMH/ListRecords/record ノードに対応する XML をファイルに格納しておけば、ファイルを連続して出力することで record 部を出力できる（XML 宣言の一行を読み飛ばして連続出力することで ListRecords の返戻の一部としてよい）。

XML ファイル生成時に必要なバリデーション[24]が行われれば、OAI-PMH で出力するときに XML としてパースする必要はないため、リポジトリが ListRecords の返戻を行う時には XML ベースで実装するのではなく、プレーンテキストとしてファイル処理を行うことでサーバ負荷を下げ、返戻を高速化できる。

3.5 の おまけ

上記の通り、ListRecords での書き出しは XML として処理する必要はないが、ListRecords の読み込みは XML として処理する必要がある（ResumptionToken を抽出する必要があるため）。この点はプロトコルの線が悪く、連続して収集する場合に一度ハーベスタ側で XML をパースする必要があり、返戻が大容量だとボトルネックになりえる。また、本ドキュメントでは触れていないが、ネットワークボトルネックを解消するために OAI-PMH では圧縮応答の方法を用意しているが、これはレコードのみではなく返戻 XML 全体を圧縮しなければならないことから、リクエストの度に XML を生成・圧縮しなくてはならず、応答速度に影響が出る場合がある。

[24] 提供用 XML として正しいフォーマットで生成されているかのチェック

4. 実用的なリポジトリの設計例

ここまでの点を踏まえ、実用的なリポジトリのデータ構造とアルゴリズムについて整理する。前提としては、公共図書館のディスカバリサービスシステムの OAI－PMH リポジトリの設計を想定する。この想定では Set は “lib”（図書資料）と “mus”（博物資料）の 2 つを持ち、lib ∩ mus = φ (空集合)および lib ∪ mus = U(全体集合)を満たすものとする。

4.1 テーブルの設計

まず、テーブルの設計を示す（図 9）。青地の表は OAI-PMH 以外の既存のデータベーステーブル（検索や GUI での情報提示に用いているもの）で、緑地が OAI-PMH 用のテーブルである。なお OAI-PMH テーブル中の XML ファイル参照先カラムは必須ではない。(OAI-ID から機械的に生成可能な Path にメタデータファイルを置き、カラムを省略しても実装可能。インデクスが減る分動作が高速化する。)

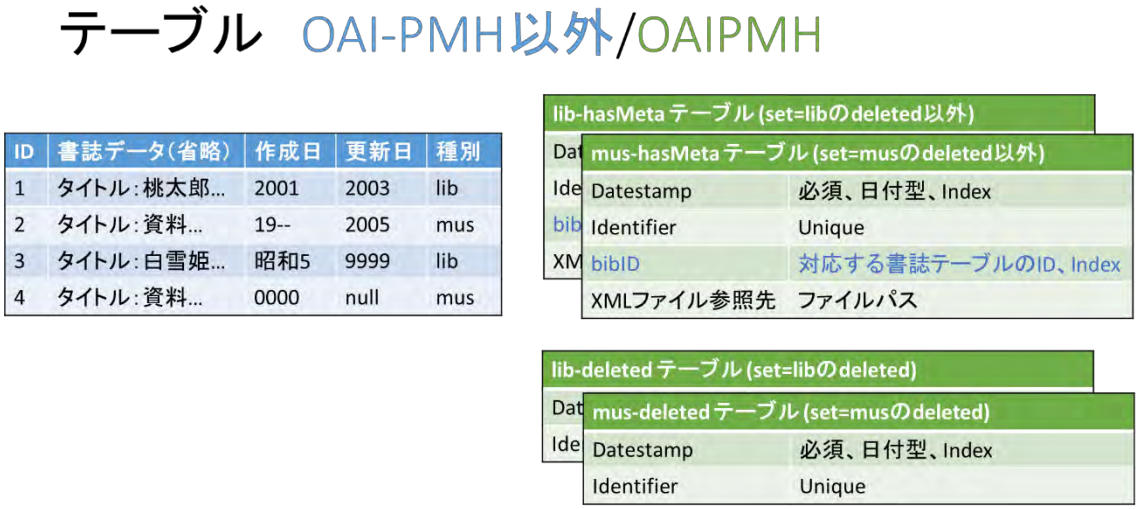

図 9 テーブル設計の概要

4.2 ListRecords への返戻

ListRecords リクエストを受け取った場合には、OAI-PMH 用のテーブルのみを参照し、図 9 左の青地で示された、元のデータベーステーブルへの参照は行わない。

Set の指定がある場合は、当該 Set の OAI-PMH テーブル 2 つ（生きているメタデータテーブルと deleted のテーブル）からそれぞれ Datestamp で範囲検索して返戻する。Set の指定がない場合には、テーブル 4 つ(2 つの Set × 2 種類(生きているデータと deleted))から範囲検索して返戻する。

4.3 テーブル更新のフロー

データを新規追加した場合には、まず GetRecord や ListRecords で返戻するための XML を生成する。その後 OAI-PMH テーブルの当該 Set のテーブルに行を追加する（図 10）。データ更新が行われた際には、新規追加と同様に先に出力 XML を更新し、対象 OAI-PMH テーブルを更新する（図 11）。

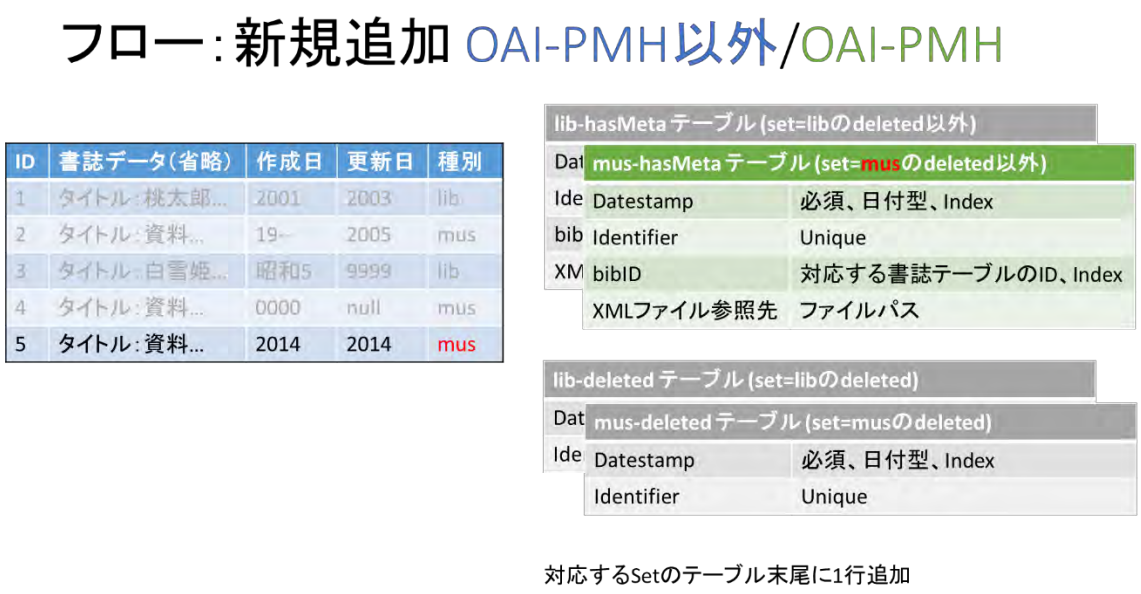

図 10 レコード新規追加のフロー

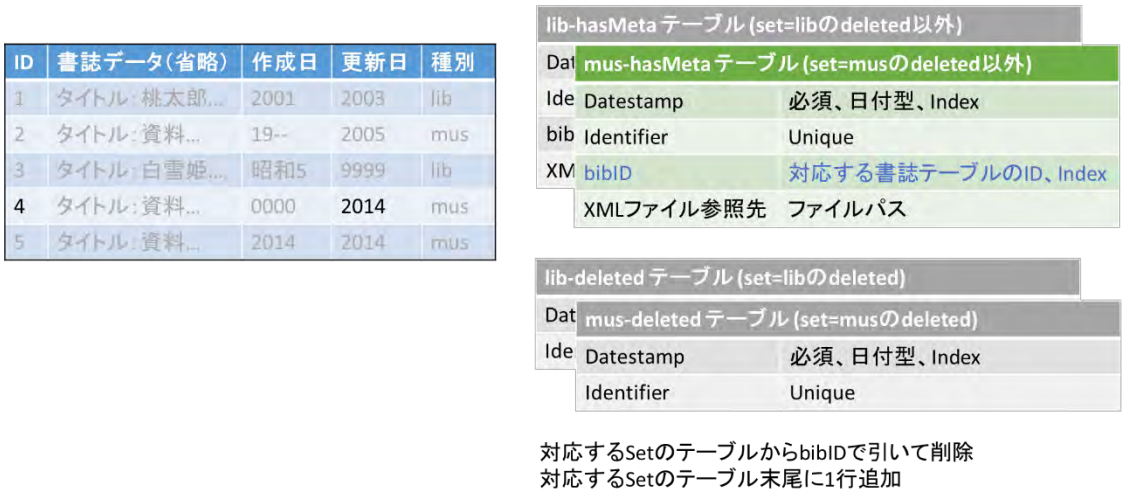

図 11 レコード更新時のフロー

OAI-PMH テーブルからデータを削除するときには、生きているメタデータテーブルから対象となる行を削除し、deleted 用のテーブルに行を追加する（図 12）。また、Set 変更の場合は、データを一度削除して新規追加する、というフローをとる。また、2.4 に記載した通り、ID は異なるものを新規で採番する必要がある。

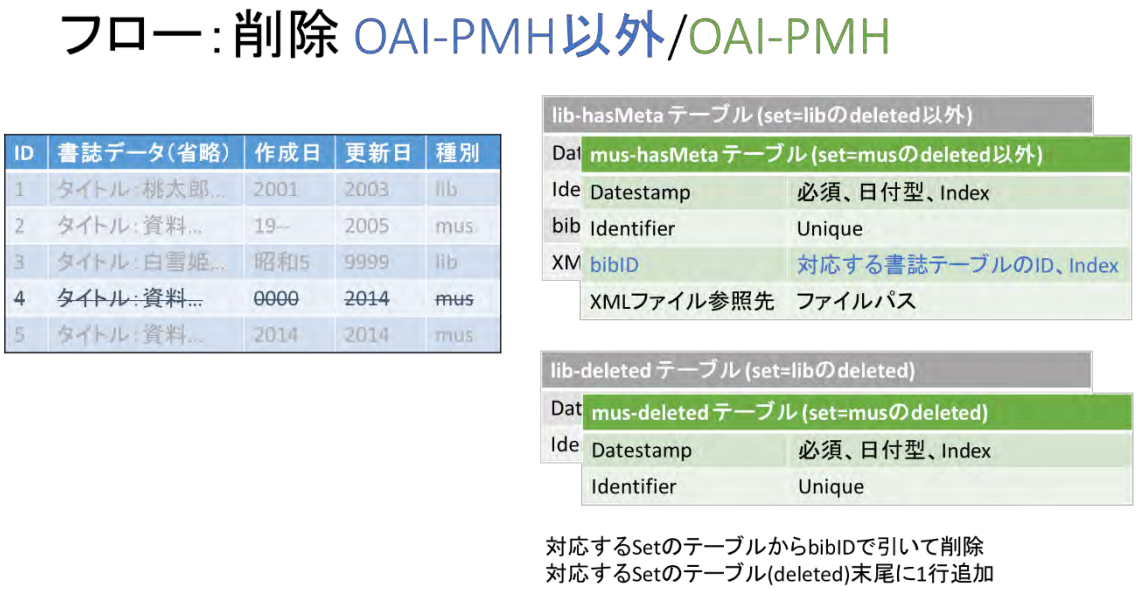

図 12 レコード削除時のフロー

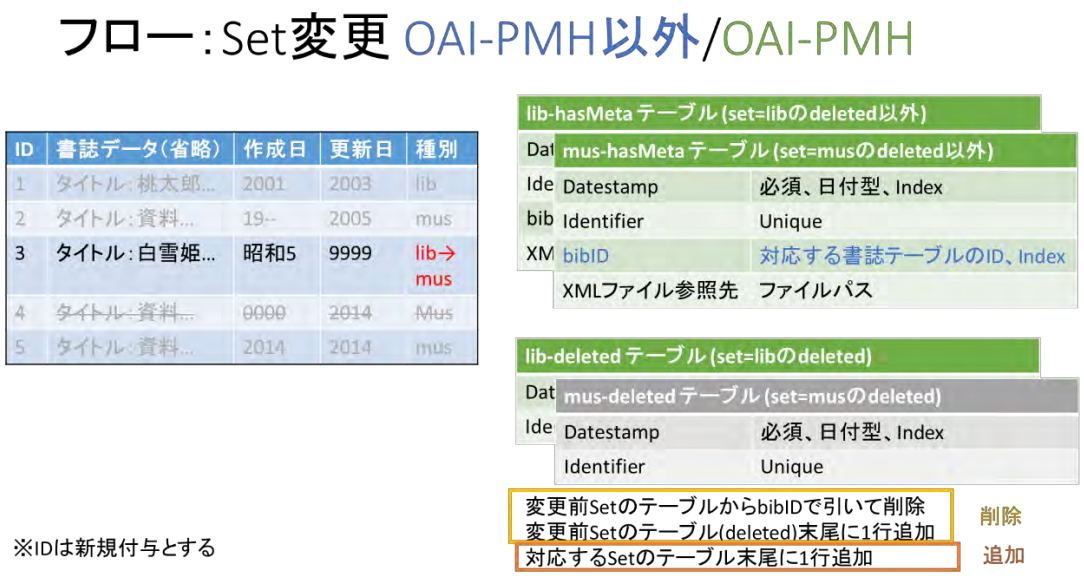

図 13 セット変更時のフロー

5. まとめ

本稿では OAI-PMH の概要を解説し、リポジトリ実装で陥りやすいリスクについて要点を並べた。また、リポジトリ全体のデータ構造とアルゴリズムについて述べた。日本国内において OAI-PMH が今以上に普及し、メタデータ交換を通じた知の交換がさらに進んでいく中で、本稿がその一助となれば幸いである。

参照文献

[1] “DSpace”. DSpace is a turnkey institutional repository application. http://dspace.org/, (accessed 2015-08-04).

[2] Smith, MacKenzie. et al. “DSpace: An Open Source Dynamic Digital Repository”. D-Lib Magazine. 2003-01. http://www.dlib.org/dlib/january03/smith/01smith.html, (accessed 2015-08-04).

[3] 林豊. ResourceSync : OAI-PMH の後継規格. カレントアウェアネス. 2015, 323, p. 17-21.

[4] “トップページ”. JAIRO Cloud コミュニティサイト. https://community.repo.nii.ac.jp/, (参照 2015-08-04).

[5] 国立国会図書館. “国立国会図書館サーチ連携拡張に係る実施計画”. 国立国会図書館デジタルコレクション. 2015-04-03. http://dl.ndl.go.jp/info:ndljp/pid/9207570, (参照 2015-08-04).

[6] “OAI-PMH2.0 日本語訳”. 2004-11-30. https://www.nii.ac.jp/irp/archive/translation/oai-pmh2.0/, (参照 2015-08-04).

[7] Lagoze, Carl. et al., eds. “Open Archives Initiative メタデータ・ハーベスティング・プロトコル実装ガイドライン”. 2002-06-14. https://www.nii.ac.jp/irp/archive/translation/oai-pmh2.0/guidelines.htm, (参照 2015-08-04).

[8] “SRU: Search/Retrieval via URL -- SRU, CQL and ZeeRex (Standards, Library of Congress)”. Standards (The Library of Congress). 2013-12-26. http://www.loc.gov/standards/sru/, (accessed 2015-08-04).

[9] ISO 23950:1998. Information and documentation -- Information retrieval (Z39.50) -- Application service definition and protocol specification.