研　究

汎用のデータ変換フレームワークを開発する

Developing a General-Purposed Data Converting Framework

名古屋大学情報連携基盤センター
Information Technology Center, Nagoya University

山本哲也
YAMAMOTO, Tetsuya

Abstract

This paper describes the development of a data convertion tool. It aims to simplify data converting processes, which consist of reading data, converting data, filtering data, and outputting the resulting data. Every phase of converting data can be easily customized by writing plug-in modules with simple rules.

Keywords: data conversion（データ変換），framework（フレームワーク），python（python），plug-in（プラグイン），reusability（再利用性）

1．はじめに

名古屋大学附属図書館では、国立情報学研究所による次世代学術コンテンツ基盤共同構築事業の委託事業（以下、委託事業）[1]において、平成18年度は多様なメタデータの［相互］交換というプロジェクトに取り組み、一定の成果をあげた。これのなかで、機関リポジトリに格納されているメタデータの中でも学位論文のもののみを抜き出して適切に変換した上、Networked Digital Library of Theses and Dissertations（NDLTD）[2]にOAI-PMHプロトコルを通じて提供し、また、junii2形式に変換してAIRWayプロジェクト[3]に提供するということを行った。これらはそれぞれ順調に稼動しているが、システムそのものについては、上に挙げたような変換ルール以外のものに流用しやすいといった汎用性においては、まだ改良の余地を残すものであった。

そもそも、連続したデータを変換するという仕事は、どんな業務においても相当頻繁に生じるものである。特に、外部からまとまった量のデータを受け取って自機関のサービスに統合しようとするときには必ず発生する種類の仕事である。そのたびにデータを変換するプログラムなどを個別に開発するのが通常であるが、これは同じような作業を何度も繰り返すことになり冗長である。このような負担を将来的に避けるために、委託事業で開発したいくつかのデータ変換プログラムをさらに一般化した形で整理しなおし、固有の変換ルールから独立した、より再利用性の高いツールとしてまとめ、自機関で使うとともに、外部にも公開・提供できるようになることが今年度の課題であった。この成果について以下に述べる。

2．設計

ここで問題にしている「連続したデータの変換」という仕事を分析すると、おおむね下のようなも

のであるとわかった。

(1) なんらかの形式で格納されたデータソースから順番にデータレコードを取り出す。
(2) それぞれのデータレコードについて、何らかの変換処理やフィルタ処理を施す。
(3) 生成されたデータを、しかるべき形式で書き出す。

変換作業の最初には (1) にあたるデータ入力の仕事が位置し、その次に必要なだけの (2) が順列につながり、最後に (3) によって結果データが出力されるというのが、データ変換作業を一般化したときのイメージ（図 1）である。

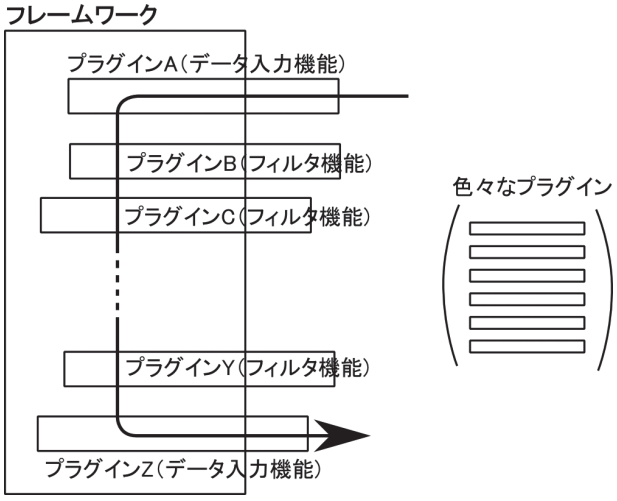

図 1　データ変換作業を一般的に表した図

また、一件のデータレコードは、ひとつ以上の項目名と、それらに対応する値から出来ているものとして一般化した。プログラミングにおいては連想配列・ハッシュ・辞書などという名前で呼ばれるものである。

これをもとに、それぞれの処理単位をプラグインできる部品として準備し、あとは動作に必要な各種パラメータを与えることで処理全体を整然と実行することができるようなフレームワークを設計した。プログラム言語には Python を採用し、部品のひとつひとつは Python のクラスオブジェクトとして記述することにした。典型的な処理はあらかじめ準備されているものの組み合わせだけで実現できるよう、いくつかの機能のプラグインをあらかじめ初期配布に含むものとして準備した。また、プログラム言語をほとんど記述しないで、ある程度の処理が実現できるようにも工夫した。もちろん、Python を自由に記述できればより高度な処理を行うことが可能になるような余地も残した。

現時点で、これらは Python 用ライブラリとして実装されている。これの公開場所などは後述することとし、まずはライブラリの利用の流れを以下に示してゆく。

なお、これらの設計および実装の各段階においては、株式会社ナレッジサイエンス[4] の協力を得た。紙面を借りてお礼を申し上げる。

3．ライブラリ利用の流れ

このライブラリの利用方法は以下のとおりである。

(1) 動作環境を確認の上、システム上に配布物をインストールする。
(2) Python スクリプトを作成し、そこで ksconv という名前のライブラリをインポートする。
(3) 必要な機能が初期配布プラグインに含まれていればそれの利用を指定して、あわせて必要なパラメータも記述する。プラグインの指定は、複数つなげて行ってよい。
(4) 初期配布で足りない機能があるときは、仕様に従ってこれを作成する。
(5) 完成した処理手順を実行する。

3. 1　動作環境とインストール作業

このライブラリをインストールして動作させるには、Python2.3 以降が動作するマシン環境が必要である。Windows 上および Linux 上でそれぞれ動作確認済みである。ただし、使用するプラグインによってはそれぞれが依存する他のライブラリが必要になったり、より新しいバージョンの Python が必要になることもある。

必須ではないが、YAML による処理手順の記述を可能にするために、PyYAML[5] または PySyck[6] ライブラリがあらかじめ Python の実行環境上にインストールされていることが望ましい。これからサンプルとして挙げていくスクリプトも、特に断りがない限りは YAML が扱えるという前提で

書かれている。

インストールは、Python 標準の distutils を使って行う。配布するアーカイブファイルを展開してできるディレクトリ下で、

python setup.py install

とコマンドを入力すればよい。これで必要なライブラリが適切な位置にコピーされ、バイトコードに変換される。通常、この作業にはコンピュータの管理者権限が必要である。

アンインストールには、ライブラリがコピーされたディレクトリを見つけてこれを削除する。通常は、Python のライブラリ群が格納されるディレクトリ下にさらに site-package ディレクトリがあって、その下に ksconv ディレクトリが見つかるはずなので、これを削除してしまえばよい。

3.2　簡単な実行例：自動付番

ここまでの手順で、Python 実行環境上で ksconv ライブラリを利用できるようになった。下のようなテスト用スクリプトを test1.py という名前で作成し、実行すると、すべてが正常なら下のような結果が標準出力に現れる。

```
# test1.py
from ksconv import run_yaml
run_yaml(" " "
- module: ksconv.line.SegGenerator
  number: 5
- module: ksconv.dump.Dump
" " ")
```

（実行結果。python test1.py とコマンドラインから入力したとき）

```
{'sequence': '1'}
{'sequence': '2'}
{'sequence': '3'}
{'sequence': '4'}
{'sequence': '5'}
```

このサンプルで示したのは、初期配布のプラグインモジュールのうち、ksconv.line.SeqGenerator と ksconv.dump.Dump のふたつを使った、単純な処理手順である。「" " "」で囲んだ部分が、処理手順を YAML で記述したところである。書き方は単純で、ハイフン記号＋空白でモジュールの区切りをつけながら、コロン記号＋空白でパラメータ名とその値を記述していけばよい。これは YAML の記法であり、より詳しくは別に情報源をあたるとよい。module パラメータが、使用するプラグインを決めるために最も重要である。大文字と小文字は区別されるので、正確に書き分けなければならない。

さきの図１に倣ってこの処理手順を図示したものが、図２である。ここでは外部からのデータ入力のかわりにデータ生成ということになるが、位置づけはほぼ同じである。

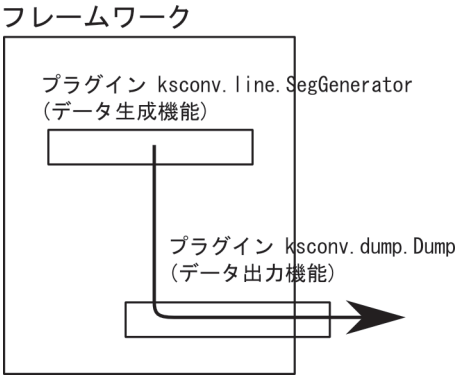

図２　test1.py の動作を図示したもの

test1.py は、YAML を扱うためのライブラリがあらかじめ準備されていることを前提に書かれたものだが、これを YAML を使わない形に書き直すこともできる。ここでは test2.py とする。実行結果は上と同じなので省略する。この書き方はやや煩雑になるので、この点でも YAML を扱えるライブラリを準備しておくのがよい。

```
# test2.py
import ksconv
c = ksconv.Chain()
c.addLink(module="ksconv.line.SeqGenerator", number=10)
c.addLink(module="ksconv.dump.Dump")
c.run()
```

3. 3　簡単な実行例：csv 読み込み

もうひとつの例（test3.py）として示すのは、csv ファイルを読み込んで、一定の値を持つものだけを表示するというスクリプトである。

```
# test3.py
from ksconv import run_yaml
run_yaml(" " "
- module: ksconv.text.CSVReader
  filename: sample.csv
- module: ksconv.filter.Match
  field: name
  match: “^a”
- module: ksconv.dump.Dump
" " ")
```

あらかじめ、スクリプトと同じディレクトリに sample.csv という名前で下のようなテキストファイルを準備した上で実行する。

```
name, age, address
aries, 20, east
taurus, 25, west
gemini, 34, north
cancer, 19, south
```

実行結果は下のようになる。

```
{‘age’: u’20’, ‘name’: u’aries’, ‘address’: u’east’}
```

ksconv.text.CSVReader は csv を読み込むためのプラグインで、現在の実装では、ファイルの一行目をヘッダ行とみなし、ここからフィールド名を読み込む。ksconv.filter.Match は、あるフィールド（field パラメータで指定する）が、match パラメータで指定したパターンに合致するかをチェックする。デフォルトの動作では合致したデータを通し、合致しないものを破棄する（オプション設定から、逆の動作にすることもできる。）「^a」とは「a で始まる」という意味であり、ここは一般的な正規表現の書法に従って書くことができる。

４．初期配布プラグイン

初期配布に含まれるプラグインモジュールは、下に挙げるとおりである（リビジョン 30 現在。）ただし、各モジュールの完成度にはまだばらつきがある状態であり、今後も仕様が変化する可能性がある。

■ ksconv. dump. Dump

このプラグインは、受け取ったデータをすべて出力して動作を確認するために使う。データは変更されずに直後のプラグインに渡されるので、これの下にさらに他の処理をつなげることができる。出力形式は Python 独特のものであり、あまり再利用性はない。主にデバッグに使うことができる。

パラメータ：なし

■ ksconv. dump. YAML

上の Dump モジュールとほぼ同じだが、出力形式が YAML である点が異なる。

パラメータ：encoding（出力エンコーディング。省略時は ’utf-8’）

■ ksconv. dump. FieldList

プラグインが受け取るデータは Python の辞書オブジェクトであるが、それらのキー一覧をリスト形式で得るために使う。

パラメータ：fieldname（リストを含むフィールド名。省略時は ’fields’）

■ ksconv. dump. Storer

このプラグインは、受け取ったデータをすべて、その構造を保ったままファイルに格納するために使う。ここで生成されたファイルは、次に挙げる Loader プラグインを使って読み込むことができる。

パラメータ：filename（出力先のファイル名。省略時は ’/dev/stdout’）

■ ksconv. dump. Loader

先の Storer モジュールで格納したデータファイルを開いて、データを復元しながら一件ずつ取

り出して処理することができる。ファイル名は、filename パラメータで指定することができる。

パラメータ：filename（入力元のファイル名。省略時は ’/dev/stdin’）

■ ksconv. text. CSVReader

CSV 形式のデータからデータを一件ずつ取り出すためのモジュールである。データの一行目はヘッダ行として扱われ、ここからフィールド名が決められる。

パラメータ：filename（入力元のファイル名。省略時は ’/dev/stdin’）
encocing（入力エンコーディング。省略時は ’utf-8’）

■ ksconv. text. CSVWriter

データを csv 形式で書き出すときにこのプラグインを使う。

パラメータ：filename（出力先のファイル名。省略時は ’/dev/stdout’）
encoding（出力エンコーディング。省略時は ’utf-8’）
fieles（出力するフィールドをリスト形式で指定）

fields はフィールド名のリストという形で記述する。YAML で処理手順を書くときには下のような形になる。

```
- module: ksconv.text.CSVWriter
  fields:
    - field1
    - field2
      ...
```

■ ksconv. field. Select

このプラグインは、指定したフィールドとその値だけを残して、他を破棄する。抽出のルールを指定する方法は二通りあり、fields パラメータにフィールド名をすべて列挙するか match パラメータに正規表現を記述して、これにマッチするフィールド名をまとめて指定する。

パラメータ：fields（選択するフィールド名のリスト）
match（選択フィールド名のマッチ条件）

■ ksconv. field. Remove

このプラグインは、指定したフィールドとそれに対応する値を除去する。上の Select プラグインの同様に、fields パラメータか match パラメータのどちらかを与えて処理対象になるフィールドを指定する。

パラメータ：fields（選択するフィールド名のリスト）
match（選択フィールド名のマッチ条件）

■ ksconv. field. Rename

このプラグインは、フィールドの名前を変更するときに使う。

パラメータ：from（置き換えるフィールド名）
to（置き換え後のフィールド名）

■ ksconv. filter. Flatten

与えられたデータが単純な Python 辞書オブジェクトでないときに、これを強制的に単純な辞書として扱えるように展開する。例を下に示す。

```
# test4.py
from ksconv import run_yaml
class NameGenerator:
  def h_convert(self, data):
    return {‘name’:
              {‘last’: [‘testuya’, ‘2’],
               ‘first’: ‘yamamoto’}}

run_yaml(" " "
- module: NameGenerator
- module: ksconv.line.Head
  number: 1
- module: ksconv.dump.Dump
- module: ksconv.field.Flatten
- module: ksconv.dump.Dump
" " ")
```

ここで行った class 宣言の意味は後述するが、ともかくこれを実行してみると、下のような結果を得る。最初の行で表示されるようなやや複雑なデータが、次の行では単純なキーと値だけの構造をもつものになっていることが見て取れる。

```
{‘name’: {‘last’: [‘testuya’, ‘2’], ‘first’: ‘yamamoto’}}
{‘name.first’: ‘yamamoto’, ‘name.last’: ‘testuya,2’}
```

パラメータ：delimiter（フィールド名を連結するときの文字。省略時はピリオド。）

■ ksconv. filter. Match

指定したフィールドの値が決まったパターンに合致したときに、フィルタ処理を行う。フィルタの動作モードが ”allow” のときは合致するデータを通過させ、”deny” のときは合致するデータを遮断する。マッチパターンは正規表現に従った書きかたで指定する。

パラメータ：mode（フィルタの動作モード。省略時は ’allow’）
field（マッチを試みるフィールド）
match（マッチ条件）

■ ksconv. line. Head

データを決まった行数だけ通過させてから動作を終了するときに使う。UNIX でいう head コマンドに動作を似せている。

パラメータ：number（通過するデータ行数。省略時は 10）

■ ksconv. line. Tail

受け取ったデータの最後の数行だけを通過させる。UNIX でいう tail コマンドに動作を似せている。

パラメータ：number（通過するデータ行数。省略時は 10）

■ ksconv. line. SeqGenerator

データに行番号をつけるときに使う。行番号が 1 から順に付与される。

パラメータ：fieldname（行番号を付与するフィールド名。省略時は ’sequence’）
number（行番号の最大値。省略時は無限とみなされる）

■ ksconv. line. Random

確率を指定して、データレコードをランダムに間引きするときに使う。例えば 0.3 のときは、受け取ったデータがそれぞれ 30％の確率で次のプラグインに渡され、残りは破棄される。

パラメータ：probabilty（通過する確率で、0 から 1 の間の小数。省略時は 0.5）

■ ksconv. line. Multiply

与えられたデータレコードをそれぞれ数回ずつ繰り返すときに使う。

パラメータ：number（繰り返す回数。省略時は 2）

■ ksconv. lib. BFormatReader

NII が NACSIS-CAT 参加組織に提供している個別版サービス[7] で得られるデータを扱うデータ入力プラグインとして試作したものである。ここでの実装では、B フォーマット形式・EUC エンコードのものだけを扱う。◆ Uxxxx ◆といった形式で記された文字は、対応するユニコード文字に置き換えられる。また、EXC 文字は対応するユニコード文字に置き換えられる。

パラメータ：filename（入力元のファイル名。省略時は ’/dev/stdin’）

■ ksconv. net. feed. Feed

RSS フィードを読み込んで扱うためのデータ入力用プラグインとして試作したものである。これを利用するには、Python の外部モジュールである、Universal Feed Parser[8] がインストールされている必要がある。

パラメータ：url（フィードの URL。必須）

■ ksconv. net. oaipmh. OAIPMH

OAI-PMH プロバイダからメタデータを取得す

るための入力用プラグインとして試作したものである。データ取得には、OAI-PMH で決められた verb のうち、ListRecords を使っている。baseurl パラメータに、OAI-PMH プロバイダの BaseURL を指定する。その他、metadataPrefix（省略時は "oai_dc"）、from、until、set パラメータを必要に応じて指定する。これらのパラメータの意味については、OAI-PMH についての各種情報源[9,10] を参照されたい。

取得されたメタデータ一件について、datestamp フィールド、deleted フィールド（ある場合）、identifier フィールド、sets フィールド（ある場合）、metadata フィールドを含む辞書オブジェクトが得られる。metadata フィールドの中はパース前の XML がそのまま含まれており、このプラグインではその処理を問題にしていない。

パラメータ：interval（HTTP リクエスト間の秒数。省略時は 5）
baseurl（OAI-PMH の BaseURL。必須）
これ以外に、OAI-PMH の各種パラメータを必要に応じて指定する。

■ ksconv. misc. DirWalk

あるディレクトリの下にあるファイル名をフルパスですべて列挙する入力プラグインとして試作したものである。指定されたディレクトリの下を再帰的に探索し、すべてのファイル名をデータとして入力する。

パラメータ：path（探索開始パス。省略時は '.'）
fieldname（表示フィールド名。省略時は 'filepath'）

■ ksconv. misc. dspace. DSpaceDump

このモジュールは、機関リポジトリなどを構築するためのソフトウェアである DSpace[11] に収録されたメタデータを列挙するための入力プラグインとして試作したものである。このプラグインは、DSpace のバックエンド DB である PostgreSQL のダンプファイルを読み込む形で作動する。動作確認した DSpace のバージョンは 1.2.1 であり、最新のバージョンの DSpace についてはここでは対応していない。参考的な実装例として収録する。

パラメータ：filename（入力元のファイル名。省略時は '/dev/stdin'）

5．独自のプラグイン

現在、初期配布のプラグインだけで完全に用が足りるとは、必ずしも期待できない。今回のデータ変換システムを開発するにおいては、簡単に機能を拡張したり、既存のロジックを再利用しやすいよう設計されている。それには新しいクラスを定義し、そこに一定の決まりに従ったメソッドを記述すればよい。

5. 1　スクリプト内に直接記述

先に ksconv.filter.Flatten の使用例で行ったような方法が、プラグインを新しく作るやりかたのひとつである。例では、NameGenerator という名前の Python クラスを定義し、その中に、self 引数以外に一つ引数を取る h_convert メソッドを作成して、Python の辞書オブジェクトを返り値とするようにした。これが、下の処理手順ではそのままプラグインの名前として使われている。これだけで、完全に動作する拡張プラグインをひとつ作ったことになる。

一般的な手順として、プラグインを作成するとは、下のような条件を満たすクラスを作成することである。

- h_convert という名前のメソッドを持つ。
- h_convert メソッドは、一つの辞書オブジェクトを引数として受け取り、一つの辞書オブジェクトを返り値として返す。
- データ入力型のプラグインなどで引数のデータを参照する必要がなければ、引数を無視するように記述してもよい。
- 受け取ったデータを破棄するときは、返り値には None を返す。
- データの入力を行う種類のプラグインで、データが尽きた状態でさらに h_convert を呼び出された場合は、このメソッドは EOFError 例外を発生させて、そのことをフレームワーク側に伝える。

これらのルールに従ったものを作成すれば、あとはクラスの名前をそのままプラグインとして使

うことができる。

プラグインの動作のために前処理や後処理（例えばデータファイルのオープンやクローズ）が必要な場合は、プラグイン用クラスには、h_start メソッドと h_end をそれぞれ定義してもよい。どちらも引数はとらない。h_start と h_end は、フレームワーク側から適切なタイミングで呼び出されるので、実行のタイミングを意識する必要はない。

変換手順に記述されたパラメータは、対応するクラスのコンストラクタである __init__ メソッドにキーワード引数として与えられるので、必要に応じてこれも書くことができる。

標準配布のプラグインはすべてこれらの仕様どおりに書かれているので、参考資料としても使えるだろう。

5.2　ライブラリとして記述

新しく作ったクラス記述を頻繁に使うようになった場合は、それを独立の Python モジュールとして別のファイルに格納すると、何度も同様のコードを書く手間が軽減されて、スクリプトそのものもプラグインの記述と変換手順の記述がはっきりと分けられるので、より望ましい。

モジュールファイルは、Python のライブラリ探索パスに入っていればどこに置いてもよい。プラグインの名前は単に Python の名前空間を表しているだけのことであるから、変換手順の記述では、プラグインの名前として、そのクラス定義をドット区切りで書けばよい。

5.3　標準配布プラグインの拡充

このツールは、プラグインモジュールを作りやすいことを意識して設計された。Python の初歩的な知識があれば簡単にこれを作ってみることができるので、ぜひ試していただきたい。データ入力、変換、出力とも、実際の仕事では様々なバリエーションがあることだろう。これらの一部をうまく扱えるようなよいモジュールができたときは、これを将来的に標準配布に含めることができるよう、本プロジェクトに提供いただけるとたいへんありがたく、また、既存のプログラムの再利用を促すという視点からも価値あることと思われる。

また、現時点での標準配布プラグインについて、完成度が充分に高い状態とは必ずしも言えない所がある。これらの改良についてもご意見をいただけると幸いである。

6．今後の展望等

現時点では、このツールはコマンドライン上で動作する種類のものである。プラグインの選択やパラメータの設定は、すべてスクリプトを直接記述するといった方法で行わなくてはならない。もちろんこれらをできる限り簡易に行えることを目指したのではあるが、それでもなお、こういった形よりも、プログラム経験の少ない人にとっては、GUI を使ってより直感的にモジュールを配置してこのツールを使えるほうがより望ましいと感じるかも知れない。例としては、WEB ブラウザ上で RSS などの情報源を加工するためのグラフィカルなツールを提供する、Yahoo! Pipes[12] などのサービスがとても参考になるイメージをもたらす。これらの実現は今後の課題として残る。

また、このシステムはまだ個々のコンピュータ上にインストールして利用するという形態のみを想定している。近年、様々なツールが Web 上で提供され、個々のコンピュータ上では煩雑なソフトウェア導入作業が必要なくなるということが起こっているが、こういった形のサービス提供が非常に有用であることからも、今回のデータ変換システムを何らかの WebAPI を介して提供できるようになることが、今後の発展の方向のひとつとして考えられるだろう。これが実現すると、たとえば手元のデータファイルを、ある決まったデータ変換ルールを提供するコンピュータ上にネットワークを通じて投入すると、即時に変換結果が得られるといった利用法が現れてくるかも知れない。また、サービス利用者に見えないレベルでも、異なったコンピュータシステム間でデータをやりとりする際には、このツールが柔軟な変換ルールを提供する「糊」として役に立てるかもしれない。

ところで、この仕組みを設計するにあたっては、RSS フィードを様々なルールで変換して扱おうとする Plagger[13] などに多くのインスピレーションを受けているが、今回扱おうとした情報は必ずしもネットワーク上の情報とは限らないこともあり、これがいわゆる「車輪の再発明」ではなかったと信じている。

７．配布情報

ここで挙げたデータ変換フレームワークのソースコードは、subversion を使って管理しており、同時に一般にも公開している。subversion が動作するコンピュータ上で、

svn co http://lab.nul.nagoya-u.ac.jp/svn/repos/ksconv/trunk

とコマンドを入力することで、最新のソースを取得できる。また、取得したディレクトリの下で、svn update とすることで、随時最新のものにアップデートできる。これ以上の情報については、一般的な subversion の情報源を参照されたい。

原稿執筆時現在（リビジョン 30）のアーカイブは、

http://info.nul.nagoya-u.ac.jp/pubwiki/index.php?ksconv

から取得できる。

参考文献・URL

１）学術機関リポジトリ構築連携支援事業．http://www.nii.ac.jp/irp/，（参照 2008-2-1）

２）Networked Digital Library of Theses and Dissertations. http://www.ndltd.org/，（参照 2008-2-1）

３）AIRway peoject．http://airway.lib.hokudai.ac.jp/，（参照 2008-2-1）

４）株式会社ナレッジサイエンス．http://www.knowledge-sci.co.jp/，（参照 2008-2-1）

５）PyYAML．http://pyyaml.org/，（参照 2008-2-1）

６）PySick．http://pyyaml.org/wiki/PySyck，（参照 2008-2-1）

７）個別版サービス．http://www.nii.ac.jp/CAT-ILL/contents/options.html#jump7（参照 2008-2-1）

８）Universal Feed Parser．http://www.feedparser.org/，（参照 2008-2-1）

９）The Open Archives Initiative Protocol for Metadata Harvesting．http://www.openarchives.org/OAI/openarchivesprotocol.html，（参照 2008-2-1）

10）国立情報学研究所．OAI-PMH の NII メタデータ・データベースへの適用について．http://www.nii.ac.jp/metadata/oai-pmh/，（参照 2008-2-1）

11）DSpace．http://www.dspace.org/，（参照 2008-2-1）

12）Yahoo! Pipes．http://pipes.yahoo.com/，（参照 2008-2-1）

13）Plagger．http://plagger.org/，（参照 2008-2-1）