biblatex-japanese パッケージ

前田一貴 †

2021 年 4 月 28 日

目次

1 はじめに 1
1.1 動機と目標 1
1.2 必要なパッケージ 2
1.3 ライセンス 2
1.4 バグレポート・要望など 3
1.5 謝辞 3
2 使い方 3
3 オプション 3
4 開発者向けコマンド解説 4
4.1 和文文字判定 4
4.2 置換コマンド 4
4.3 割り込み 6

1 はじめに

1.1 動機と目標

LaTeX においては文献引用の煩雑な作業の効率化のために BibTeX がよく用いられている．BibTeX を使うと，文献データベースを作成しておき，本文中の引用が必要な箇所に`\cite` コマンドを書くだけで自動的に，指定したスタイルでソートされた文献リストが指定箇所に，ソート後の文献番号が本文中の引用箇所に出力される．BibTeX は特に参考文献が膨大な数に及ぶ書籍や論文の執筆には欠かせない機構であるが，文献リストのスタイルを指定する`.bst` ファイルの作成には独自言語[1)]を用いなければならず，指定された様式に沿う`.bst` ファイルが用意されていない場合には

1） PostScript と同様の postfix stack-based programming language, いわゆる逆ポーランド記法に基づいた言語であり，不慣れな者には扱い難い．

門外漢には利用が困難であるという問題があった．

この困難の緩和を目的の一つとして，近年になって biblatex[2] というパッケージが開発されている．biblatex は文献のソートに biber と呼ばれる Perl スクリプトを用いる以外は全て TEX のマクロで記述されており，カスタマイズが（BibTEX に比べると）容易になっている．また biblatex では，従来は別のパッケージで提供されていた，脚注への文献情報の出力や章毎の文献リストの出力といった，人文系で要求されることの多い機能が充実しており[3]，欧米の人文系学界では BibTEX に代わるものとして普及が進んでいるとされている[4]．しかし，biblatex では和文における利用は現状全く考慮されていないため，日本国内では需要があるにも関わらずあまり活用されていないようである．需要に対して供給がない理由は，日本の LATEX 利用者の多くが理科系の人間であり，理科系においては論文といえば多くが英文によるものであること，理科系の和文論文を書く場合にも BibTEX の機能で事足りてしまうといったことがあると考えられる．

そこで，こうした問題に対処するために，biblatex-japanese というパッケージを作るプロジェクトを立ち上げることにした．「パッケージを作る」とは書いたが，立ち上げ時点では着地点がどこになるかはまだ定かではない．biblatex を使わなくとも BibTEX で問題を解決することも可能であるという話もある[5]ので，そうしたノウハウを集めて形にするといったことも目標として考えられる．いずれにせよ，これを書いている人（前田）自身は理科系の人間であるため，具体的にどういう要望があるのかを集約することが喫緊の課題であると考えている．

おぼろげながら考えているロードマップ（と呼べるほどのものではないが）は以下のようである．

1. GitHub の Issue 機能などを用いて要望を集める．
2. コードを書いて要望を実現する．ベースとしては biblatex や biblatex-chicago を用い，和文での文献引用における様々なニーズに柔軟に対応できるようなフレームワークを用意することを目指す．
3. 成果物は CTAN にアップロードし，TEX Live をインストールするだけで利用可能にする．

1.2 必要なパッケージ

biblatex v3.4 以上が必要．biblatex-chicago については，v1.0rc1 で動作を確認している．

1.3 ライセンス

二条項 BSD ライセンスとする．同梱の LICENSE を参照し，従うこと．

2） https://github.com/plk/biblatex
3） おそらく .bst が云々よりもこちらが移行する理由としては大きいと思われる．
4） 本当かどうかはちゃんと調べていないので知らない．
5） https://twitter.com/munepixyz/status/663199507392237568

1.4 バグレポート・要望など

パッケージの開発には GitHub[6] を利用している．バグレポートや要望はここの Issues に書き込めば，対応可能である（もちろん日本語で構わない）．また，パッチを投げるには Pull requests が利用できる．

GitHub が利用できない場合は，メールで報告していただいても構わない．メールアドレスは最初のページの脚注に書いてある．

1.5 謝辞

2 使い方

biblatex-japanese 一式を TEXMF ツリーの適切な場所に配置したうえで，プリアンブルに次を入れる：

```
\usepackage[backend=biber]{biblatex-japanese}
```

これで biblatex 本体および日本語用の設定ファイルが読み込まれる．biblatex に渡したいオプションは biblatex-japanese に渡せばよい．例えば，引用時の番号をソートしたければ

```
\usepackage[backend=biber,sortcites=true]{biblatex-japanese}
```

のようにする．

注意 まだまだ開発は始めたばかりのため，とても実用に耐えるものではない．気長に待っていただければ幸いである．

3 オプション

chicago=true, false default: false

biblatex-chicago を使いたいときに指定する．このときは，biblatex-chicago が自動的に読み込まれ，これに渡すオプションも通るようになる．

yearsuffix=true, false default: false

和文文献の年号のあとに「年」をつけるかどうか．

nameorder=true, false default: false

和文文献でも欧文文献と同じように「姓, 名」という形式（順番）で名前を指定している bib ファイルを利用する場合に true にする．

6) https://github.com/kmaed/biblatex-japanese

4 開発者向けコマンド解説

biblatex-japanese では，日本語対応ファイルの開発に用いるコマンド（マクロ）を biblatex-japanese.def というファイルの中に記述している．使用例は japanese.lbx 内で見つけられるだろう．

なお，以下に説明するコマンドの名前・文法・仕様はまだ流動的であり，今後のバージョンで変更がある可能性が高いので注意を要する．

4.1 和文文字判定

欧文文献なのか和文文献なのかを判定するための一つの方法は，文献データベース自体に langid といった言語情報を付加しておくことである．しかし，このような情報をいちいち付加するのは面倒であるし，フィールド毎に和文文字を含むかどうかで動作を変更できると便利である．このためのテストコマンドを準備しておく．

なお，Unicode であることを前提にしているので，pTEX の場合は怪しい動作になる．どうなるのかをちゃんと理解したら対策するかもしれない．

\ifCJKstr{⟨*tokens*⟩}{⟨*true code*⟩}{⟨*false code*⟩}

⟨*tokens*⟩ に U+2E80 以上[7)] のコードポイントの文字が含まれていれば ⟨*true code*⟩ を実行し，そうでなければ ⟨*false code*⟩ を実行する．テストは一度 ⟨*tokens*⟩ を完全展開してから行われる．

\ifbeginwithCJKchar{⟨*tokens*⟩}{⟨*true code*⟩}{⟨*false code*⟩}

⟨*tokens*⟩ の先頭に U+2E80 以上のコードポイントの文字があれば ⟨*true code*⟩ を実行し，そうでなければ ⟨*false code*⟩ を実行する．テストは一度 ⟨*tokens*⟩ を完全展開してから行われる．

\ifendwithCJKchar{⟨*tokens*⟩}{⟨*true code*⟩}{⟨*false code*⟩}

⟨*tokens*⟩ の末尾に U+2E80 以上のコードポイントの文字があれば ⟨*true code*⟩ を実行し，そうでなければ ⟨*false code*⟩ を実行する．テストは一度 ⟨*tokens*⟩ を完全展開してから行われる．

4.2 置換コマンド

和文論文の参考文献に欧文文献と和文文献がある場合，欧文文献については既存のマクロで処理し，和文文献の場合は別のマクロに置き換えて処理したいという状況がよくある．こうしたマクロの置換処理を効率的に記述するために，以下の一連のコマンドを準備しておく．

\replacecommand{⟨*command*⟩}{⟨*replace rule name*⟩}{⟨*code*⟩}

もし ⟨*replace rule name*⟩ の条件に合致するならば，⟨*command*⟩ の中身を ⟨*code*⟩ に置き換える．

\replacebibmacro{⟨*name*⟩}{⟨*replace rule name*⟩}{⟨*code*⟩}

\replacefieldformat[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*replace rule name*⟩}{⟨*code*⟩}

7) 変更可能にすることが今後の課題．

\replacenameformat[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*replace rule name*⟩}{⟨*code*⟩}

それぞれ\replacecommand の bibmacro, field format, name list format に対応するもの．

\replacefieldformat, \replacenameformat では，replace rule の定義中にある\blxja@format が冒頭で ⟨*format*⟩ と定義される．このことを利用すると，⟨*format*⟩ 毎に replace rule を用意する手間を省ける．

⟨*replace rule name*⟩ には以下のいずれかを指定する．

iflangidisjapanese　文献出力処理中に，もしその文献の langid が japanese ならば真になる．

iffieldisCJK　\replacefieldformat で用いると，対象の field が和文文字を含む時に真になる．

\replacecommand*{⟨*command*⟩}{⟨*replace rule code*⟩}{⟨*code*⟩}

\replacebibmacro*{⟨*name*⟩}{⟨*replace rule code*⟩}{⟨*code*⟩}

\replacefieldformat*[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*replace rule code*⟩}{⟨*code*⟩}

\replacenameformat*[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*replace rule code*⟩}{⟨*code*⟩}

⟨*replace rule name*⟩ の代わりに置換ルールのコードを直接引数に取るバージョン．置換ルールコード中では，#1 が置換前のコードを，#2 が置換後のコードを表す．例えば，

```
\replacecommand{\command}{%
  \iffieldequalstr{langid}{japanese}
                  {#2}
                  {#1}}{...}
```

と書けば，スター無し版で ⟨*replace rule name*⟩ に iflangidisjapanese を指定したのと同じことになる．

よく使う置換ルールは，次の\newreplacerule で定義した方が便利である．

\newreplacerule{⟨*replace rule name*⟩}{⟨*replace rule code*⟩}

\renewreplacerule{⟨*replace rule name*⟩}{⟨*replace rule code*⟩}

置換ルール ⟨*replace rule name*⟩ を ⟨*replace rule code*⟩ で定義する．⟨*replace rule code*⟩ の書き方は既述の通り．

置換コマンド使用上の注意点　\DefineBibliographyExtras などの引数内で使用する場合は注意が必要である．以下でこのことについて説明する．

一連の置換コマンドは etoolbox パッケージ[8)]のパッチコマンドを用いて実装されている．このパッチコマンドはパラメータ文字（#1 など）を取り扱うために，一度#のカテゴリコードを 12 (other) に変えてから引数のコードを取り込み，後で\scantokens で読み直すという処理を行っている．このため，引数のコードを一度別のマクロに保存してから実行するような処理と相性が悪い．このことを理解するには次のコードを試してみるとよい：

8）https://github.com/josephwright/etoolbox

```
\def\a#1{\detokenize{#1}}
\a{#2}
```

処理すると結果は「##2」となるはずである．このように二重#に化けてからパッチコマンドに渡すと，うまくパッチが当てられないということになる．この現象と同じことが\DeclareBibliographyExtras や\DefineBibliographyExtras などの引数にパラメータ文字を含む⟨*code*⟩を持つ置換コマンドを与えた場合に起こり，パッチ処理が失敗するという結果になる．

この問題の回避策としては，前もって#のカテゴリコードを 12 (other) に変えてしまうという方法がある．実例は japanese.lbx にある．もちろん，問題部分を読み取らせた後はカテゴリコードを元に戻す必要がある．なお，置換コマンド以外（定義関係のコマンド）の引数で#を含むものがある場合は，カテゴリコード 12 で読ませると当然意図通りにならないので，この場合は処理を分割する必要がある．

4.3 割り込み

\printfieldhook[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*code*⟩}

\printfield の実行前に割り込んで ⟨*code*⟩ を実行する．割り込みが実行されるのは実際に print すべきものが存在するときのみである．ここで，割り込み箇所は\newunit での約物出力処理 (\blx@begunit) よりも前なので，この約物出力を抑制することが可能である．現在のところ，用途はこれ以外に思いつかない（し，あまり筋が良い方法とも思えない……）．