# 日本語文献史料の構造化記述のための軽量マークアップ言語の開発

橋本 雄太（国立歴史民俗博物館）
宮川 真弥（慶應義塾大学・日本学術振興会）

歴史文献史料を構造化されたデータとして扱うためには，何らかの機械処理可能なマークアップ言語を用いて翻刻文を記述することが望ましい．加えて人文学研究者やクラウドソーシング参加者が利用するためには，そのような言語は簡潔かつ可読性の高い文法で記述される必要がある．そこで，古文書や古記録，古典籍といった日本語文献史料を記述するための軽量マークアップ言語を開発した．この言語の文法を形式文法のひとつである解析表現文法によって定義し，また縦書き入力やシンタックスハイライトに対応したオンラインエディタを開発した．このような入力負荷の低いマークアップ言語が普及することで，クラウドソーシングによる史料翻刻や，文献史料のデータベース化が効率的に進むことが期待される．

# A Light-weight Markup Language for Structured Description of Japanese Historical Documents

Yuta Hashimoto (National Museum of Japanese History)
Shinya Miyagawa (Keio University / Japanese Society for the Promotion of Science)

In order to process historical documents as semi-structured data, it is desirable that the transcriptions of those documents are encoded with some sort of machine-readable markup languages. In addition, for the use by humanities scholars and crowdsourcing project participants, such a language needs to have concise and readable grammars. *Koji* is a light-weight markup language that we have developed for the encoding of pre-modern Japanese documents. It has a simple and readable syntax prescribed with Parsing Expression Grammars (PEG), a type of analytical formal grammars. We also developed a web-based editor for *Koji* that supports syntax highlighting, snippet insertion, and so on.

## １．はじめに

気象学や地震学などの自然科学諸分野において，過去の自然現象を解明するために前近代の文献史料を「データ」として活用する動きが進んでいる[1]．たとえば京都大学古地震研究会が主催する「みんなで翻刻」[1]では，クラウドソーシングによって 530 万文字もの災害史料をテキスト化することに成功した．同研究会では，翻刻テキストから災害研究に有用な情報の抽出を試みているが，非構造化データである翻刻文から有用な情報を収集するには，自然言語処理を含む様々な工夫が必要である．

文献史料から効率的な情報抽出を可能にするには，その翻刻文はプレーンテキストではなく，何らかのマークアップ言語を用いて記述されていることが望ましい．たとえば史料中に現れる地理名称が機械処理可能な形式でマークアップされていれば，座標情報を含む歴史地理データベースを併用することで，史料中に出現する地名を地図上にプロットするといった操作が機械的に実行可能になる．こうした操作は史料内容を自然科学的見地から検討する際に特に有用である．

現在，日本語の文献史料の翻刻の入出力には，もっぱら MS Word や一太郎などのワードプロセッサーや InDesign などの DTP ソフトウェアが使用されている．これらのソフトウェアは縦書きの文書編集に対応し，振り仮名や割書きなどの表記法にも対応することから，文献史料の翻刻に広く利用されている．しかしこれらは基本的に紙媒体に印刷される文書の編集を目的としたソフトウェアであり，文献史料の論理構造や意味内容を機械可読形式で記述する用途には適していない．

人文学資料を構造化して符号化するための規格には，1980 年代から欧州を中心に策定されている TEI（Text Encoding Initiative）がある．TEI は XML を用いて資料テキストを構造化するフォーマットであり，国際標準として人文学資料のテキストデータベース構築に広く利用されている．TEI コミュニティによって整備されている TEI ガ

[1] たとえば伊藤[2]，弘瀬・中西[3]など．

イドライン[4]には，テキストに付随する多種多様な情報を構造化して記述するための XML 要素や属性が定義されている．

一方で，TEI を日本語の文献史料に適用するには課題も多い．作業者には XML の文法やスキーマに関する知識が不可欠であり，効率的に XML を編集するためには oXygen[5]など有償の高機能なエディターによる支援も欠かせない．こうした教育コストや導入コスト，作業負荷を考慮すると，現実的には TEI が日本語文献史料の専門家の間で短期間のうちに普及するとは考えにくい．また，技術知識を持たないクラウドソーシング参加者が TEI を使用することも困難である．

TEI の入力負荷の問題については，海外の DH 分野でもすでに意識されており，Web ブラウザ上で容易にテキストのアノテーションとアノテーション結果の TEI 出力を可能にする Web アプリケーションの CATMA（Computer Aided Textual Markup and Analysis）[6]も開発されている．

本研究では，教育コストや作業負荷を抑えつつ文献史料の構造化記述を可能にするために，日本語文献史料に特化した軽量マークアップ言語とその処理系を開発する．

CATMA のような Web アプリケーションではなく，新しいマークアップ言語を開発する理由は，それが可搬性に優れているからである．たとえば言語処理系をオープンソースの JavaScript ライブラリとして公開すれば，「みんなで翻刻」のようなクラウドソーシングシステムや，文献史料のテキストデータベースなど，様々な Web システムに言語を組み込むことが可能になる．

## ２．研究のねらい

軽量マークアップ言語とは，簡潔な文法を持ち，データ記述言語としての整合性と，可読性と記述の容易さを両立させたマークアップ言語である．代表的な軽量マークアップ言語には Markdown や Wiki 記法，AsciiDoc，reStructuredText などがある．HTML や XML のソースの可読性の低さや文法の冗長性を補うために，これらはソフトウェアのドキュメンテーションやブログ記事の記述など，さまざまな場面において利用されている．

本研究の基本的アイデアは，前近代の日本語文献史料を記述対象とした，次の条件を満たすマークアップ言語を開発することである．

1. 簡潔な文法を持ち，可読性が高く記述が容易である．
2. 日本語の文献史料の論理構造や，種々の表記法を記述することができる．
3. 史料中に出現する日時や地名など，歴史研究上の重要情報をマークアップ可能である．
4. ソーステキストを構文解析した抽象構文木から，TEI/XML や HTML など他のデータフォーマットへと変換が可能である（図 1）．

このようなマークアップ言語とその処理系が，史料翻刻に携わる人文学研究者の間で普及することによって，文献史料の機械可読データとしての利用可能性が高まることが期待される．また，「みんなで翻刻」のような史料翻刻のクラウドソーシングシステムに言語を組み込むことで，技術的バックグラウンドのない市民も史料の構造化に参加することが可能になる．

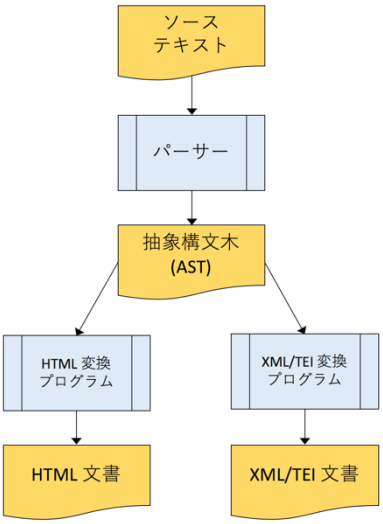


図 1 ソーステキストから他フォーマットへの変換フロー

## ３．言語の設計

Koji は，先述の目的のために筆者らが開発している軽量マークアップ言語である[1]．Koji は，日本語の文書編集環境で容易に利用できるように，①縦書きで，②全角文字のみを使用して，③Web ブラウザ上で入力できるように設計されている．以下では Koji の基本的な言語仕様を簡単に述べる．

### ３．１．文字セット

Unicode が定める文字集合のうち，次の範囲にある文字を，Koji のソーステキストを構成する文字として受け付ける：ASCII 文字（0020-007F），ラテン 1 補助（0080–00FF），CJK の記号及び句読点（3000-303F），平仮名・片仮名（3040-30FF），漢文用記号（3190-319F），全角 ASCII 文字（FF01-FF5E），CJK 統合漢字（4E00-9FEA），変体仮名（1B000-1B12F），および CJK 統合漢字拡張 A〜F．

[1] https://github.com/yuta1984/koji

基本的に Koji のソーステキストは全角文字で記述されることを前提としているが，半角の ASCII 文字やラテン 1 補助を含めたのは，史料中に欧文が現れることを想定したためである．なお，全角の（）｛｝［］《》【】といった括弧記号類は，後述するようにタグを構成する特殊記号として使用され，マークアップ対象のテキスト中では使用することはできない．

### ３．２．ソーステキストの構成要素

Koji のソーステキストは，次の 4 種類の組み合わせによって構成される．

1. 章段など複数行にわたるテキストの範囲をマークアップするブロック要素
2. 人名や日時，振り仮名など，比較的短い範囲のテキストをマークアップするインライン要素
3. ブロック要素やインライン要素の略記表現である糖衣構文
4. 要素や糖衣構文を含まない単純なテキスト

### ３．３．ブロック要素

ブロック要素は，複数行にわたる文章のまとまりのマークアップに使用されるもので，

［要素名］テキスト［／要素名］

という構文で記述される．上記の大括弧（[]）はいずれも全角文字である．たとえば，複数行にわたって字下げされる文章や，文中に挿入された絵図はブロック要素で記述される．

### ３．４．インライン要素

インライン要素は，資料中の文字や語句などより短い文章単位をマークアップするための要素である．インライン要素は全角棒線（｜）で区切られた，単一または複数のテキストを囲むことができ，

《要素名：テキスト１｜テキスト２｜…》

という構文で記述される．振り仮名や割書きなどの表記，また人名や地理名称などの事物名はインライン要素でマークアップされる．

### ３．５．糖衣構文

これらの要素を簡単に記述するために，複数の糖衣構文（シンタックスシュガー）が用意されている．たとえば，連続する漢字の後に置かれた括弧書きは，**《振り仮名》**のインライン要素として解釈される．この記法は，青空文庫の注記記法を参考にしたものである．

### ３．６．識別子

任意の要素は，HTML の ID 属性やクラス属性に相当する識別子をもつことができる．

**(1) ブロック要素の例**

［字下げ］
（本文テキスト）
［／字下げ］

**(2) インライン要素の例**

《日時：弘化四年三月廿六日》
《場所：東海道筋三島宿》
《振り仮名：般若｜はんにゃ》

**(3) 糖衣構文の例**

般若（はんにゃ）経（きょう）
→《振り仮名：般若｜はんにゃ》《振り仮名：経｜きょう》
月［ニ］有［リ］｛レ｝陰
→月《送り仮名：ニ》有《送り仮名：リ》《返り点：レ》陰

**(4) 識別子の例**

《人物＃ID 1 ：川上金五郎》
《場所＊クラス 1：信州中之条》

図 2 Koji によるマークアップの例

**(1) ブロック要素**

論理構造：表題，尚々書き，勘返状，一つ書き
位置情報：字下げ，地付き
絵図：花押，印，表，系図，地図，絵

**(2) インライン要素**

事物：人物，場所，地理，日時，期間
字体情報：略字，合字，難読文字，虫損
位置情報：振り仮名，振り漢字，訂正，傍注，迎え仮名，送り仮名，頭注，脚注，割注
記号：返り点，合符，朱引，傍線，ヲコト点，文字囲，彫り残し，棒引き

図 3 Koji がサポートする要素の例

要素名の後に全角の「＃」に続けて書かれた文字列は ID 属性として識別される．ID 属性の値は，同一のソーステキスト中でユニークである必要がある．また要素名の後に全角の「＊」に続けて書かれた文字列は，クラス識別子として解釈される．クラス識別子は，複数の要素に同じラベルを適用することを目的とした識別子で，同一のソーステキスト中に複数回出現することができる．

図 2 に Koji によるマークアップの例を，図 3 に Koji がサポートする要素の例を示す．

## ４．解析表現文法による構文定義

本研究では，Koji の厳密な構文定義を，形式文法の一種である解析表現文法 (Parser Expression Grammar, PEG) [7]によって記述する．PEG は文脈自由言語の記述文法である BNF と一見類似しているが，BNF と異なり文法記述に曖昧性を許さない．このため解析表現文法の構文解析木は一意に定まる．

解析表現文法はソーステキストの解析にあたり字句解析の必要がなく，形式文法の記述言語として広く利用されている．青空文庫の注記記法を解析表現文法で記述した取り組みもある[8]．

解析表現文法で記述した文法は，再帰下降パーサーに自動変換することができる．しかし解析表現文法のパーサーは無限長の先読みが可能であるため，最悪の場合解析に指数関数時間を要する．ただし解析過程の結果をメモ化して再利用する Packrat Parser[9]に変換すれば，解析を線形時間で完了することができる．

図 4 に解析表現文法で記述した Koji の文法の一部を示す．”->” の左辺にある英字は，右辺によって定義される非終端記号を示している．右辺に現れる記法の意味は次の通りである：

- e1 e2　　…記号 e1 と e2 の並びにマッチする
- e1 / e2　　…記号 e1 へのマッチを試み，失敗すれば e2 へのマッチを試みる
- e*　　…記号 e の 0 回以上の繰返し
- e+　　…記号 e の 1 回以上の繰返し
- e?　　…記号 e の 0-1 回の出現
- "文字列"　…指定文字列にマッチする
- [文字範囲] …指定文字範囲にマッチする

本研究では，JavaScript 製の解析表現文法のパーサージェネレータである PEG.js[10]を使用して，Koji の文法を記述し，パーサーを生成した．文法は GitHub（https://github.com/yuta1984/koji）で公開されている．出力したパーサー自体も JavaScript で構成されるため，Web ブラウザ上で動作可能である．

## ５．抽象構文木

上述のパーサーは，Koji のソーステキストを解析し，その抽象構文木（Abstract Syntax Tree, AST）を出力する．抽象構文木とは，ソーステキストの文法構造をツリーのデータ構造で表現したものである．抽象構文木はソーステキストの解析結果をそのまま表現する構文解析木とは異なり，ソーステキストの意味に関係する情報のみを抽象化して保持する．一般に，抽象構文木はプログラムのソースコードから最適化された実行ファイルを出力する前段階の中間表現として利用されるが，markdown などの軽量マークアップ言語をパースした際の中間表現としても利用される．

```
# 文書全体
Document
 -> (Block / Inline / SyntaxSugar / Text)+

# ブロック要素
Block
 -> FrontCover / BackCover
  / Front / Back / Section / Colophon
  / Indent / Waka / Figure / …

# 表紙（ブロック要素のひとつ）
FrontCover
 -> "［表紙" Attrs "］"
     BlockContent
    "［／表紙］"

# ブロック要素の内部
BlockContent
 -> (Block / Inline / SyntaxSugar / Text)*

# インライン要素
Inline
 -> Person / Place / Date / Furigana
   / Kaeri / Okuri / Warigaki / Box / …

# 人物（インライン要素のひとつ）
Person
 -> "《人物" Attrs "：" InlineContent "》"

# インライン要素の内部
InlineContent
 -> (Inline / SyntaxSugar / Text)*

# 属性
Attrs -> Id? Class*

# ID・クラス識別子
Id -> "#" [A-z0-9]+
Class -> "＊" [A-z0-9]+

# 糖衣構文
SyntaxSugar
 -> SSFurigana / SSKaeriten / ...

# ふりがな要素の糖衣構文
SSFurigana
 -> Kanji+ / Kana+ / Alphabet+
    "（" Text "）"

# 送り仮名要素の糖衣構文
SSOkuri
 -> Kanji " {" Kana+ "} "
```

図 4 解析表現文法で記述した Koji の文法の一部

Koji のパーサーは，ソーステキストを解析し，抽象構文木を出力する．たとえば図 5 は，次のブロック要素のみからなる Koji のソーステキストから出力された抽象構文木を，JSON 形式で出力したものである．

```
[字下げ三]字下げされるテキスト[／字下げ三]
```

ルート要素の `document` の子要素として，字下げのブロック要素を表す `indent` 要素が格納されており，また `indent` 要素の子要素として，ブロック要素内のテキストが格納されている．`location` プロパティは，エラーメッセージ出力のために，各要素に対応するソーステキスト中の位置を示している．

こうした抽象構文木から HTML や XML を生成するには，木構造を走査し，各要素に対応する HTML や XML を構成する訪問器（visitor）プログラムを作成すればよい．

## 6．オンラインエディタ

Koji のソーステキストは一般的なテキストエディタで記述可能である．しかし利用者にとっては，シンタックスハイライトや文法エラーの表示に対応した専用エディタが提供されることが望ましい．そこで JavaScript フレームワークの Vue.js を使用し，Koji のソーステキストを記述するための Web エディタ（http://koji-lang.org/）を試作した．

この Web エディタはいわゆるライブプレビューに対応し，ソーステキストを解析した抽象構文木，また抽象構文木を変換した HTML や XML をリアルタイムに表示することができる．

ソーステキストの入力用エディタは，縦書き編集に対応している．また，ブロック要素やインライン要素など，Koji の文法要素をハイライト表示し，ソーステキストに文法エラーが含まれていた際にはエラー箇所を強調表示する．

この機能は Web エディタの CodeMirror と同様のアプローチによって実現されている．つまり，ユーザーには不可視状態にした `textarea` 要素にソーステキストを入力させ，入力イベントが発生する度に `textarea` 要素と同位置に配置した `div` 要素にハイライト済みのテキストを逐一表示することで，擬似的にシンタックスハイライトを実現しているのである．

一般に DOM 操作はブラウザにとって負荷の高い処理であるが，仮想 DOM を利用して描画前後の差分を計算することで，ソーステキストの変更にともなう再描画にかかる処理を低く抑えている．

現在このエディタは Vue.js に依存しているが，依存性を排し，他の Web ページに埋め込み可能な JavaScript ライブラリとしてオープンソース公開することを予定している．

```
{
  "type":"document",
  "content":[
    {
      "type":"indent",
      "size":3,
      "attrs":{},
      "content":[
        {
          "type":"text",
          "value":"字下げされるテキスト"
        }
      ],
      "location":{
        "start":{
          "offset":0,
          "line":1,
          "column":1
        },
        "end":{
          "offset":25,
          "line":3,
          "column":8
        }
      }
    }
  ]
}
```

図 5 抽象構文木の JSON 出力


図 6 Koji のオンラインエディタ

## 7．議論

ここまでに Koji の文法とその処理系，編集環境について概略を述べた．以下ではいくつかの重要と思われるポイント，特に TEI との関連について議論する．

### ７．１．Koji の適用対象について

Koji は可読性と書き易さに重点を置いたマークアップ言語であり，その代償として表現力の面でさまざまな制約を抱えている．たとえば，史料の言語学的特性を詳細に捉えたコーパスを記述する言語としては，Koji の表現力は不十分だと言わざるをえない．

しかしながら，「みんなで翻刻」などのクラウドソーシング翻刻プロジェクトの参加者や，XML 関連技術に不案内な人文学研究者が文献史料を構造化して記述する際には，Koji のような軽量マークアップ言語は，予備知識を必要としない記述言語として有用性を発揮するはずである．

ただし，互換性のない異なるマークアップ言語が複数乱立する状況は，学術研究のエコシステムにとって望ましくない．そこで，言語間の互換性を築くことが非常に重要な課題となる．

### ７．２．TEI への変換について

Koji のソーステキストは抽象構文木を介して XML に変換可能である．しかしながら，TEI のスキーマで妥当な XML を出力するためには，いくつかの課題がある．

第一は日本語史料のマークアップ方法に未確定事項が多いことである．たとえば振り仮名や返り点，ヲコト点といった日本語特有の表記を，TEI の枠内でどのように表現するべきであるのかについては，TEI コミュニティの中でも議論が始まったばかりである．Koji のソーステキストから TEI へのマッピング方法を確定するには，日本語史料のマークアップについて TEI の枠内で議論を深める必要があるだろう．当面は Koji の抽象構文木から TEI への変換について，さまざまな手法を試すことになるが，これは日本語史料についての TEI コミュニティの議論を促進することに寄与すると思われる．

第二は文法制約の差である．TEI は XML 要素の入れ子関係や出現順序について XML スキーマにより制約を課しているが，Koji の文法には現時点でそのような制約は存在しない．このため単純な木構造の変換では，妥当な TEI 文書が出力できないケースが予想される．これについては Koji の側でも同様の文法制約を追加することで解決する可能性がある．

## ８．おわりに

本稿では，前近代の日本語文献史料のための軽量マークアップ言語 Koji の設計の概略を述べた．Koji の言語仕様には未確定な部分も多く，日本語文献史料をどの程度妥当に記述できるかは明らかではない．したがって日本語史料の専門家と対話を重ねながら，長期的に開発を継続する予定である．

## 参考文献


[1] 京都大学古地震研究会 : みんなで翻刻，入手先〈https://honkoku.org〉（参照 2018-09-06）.

[2] 伊藤啓介 : 藤木久志『日本中世災害史年表稿』を利用した気候変動と災害史料の関係の検討，気候適応史プロジェクト成果報告集 I , p.65-75（2016）.

[3] 2）弘瀬冬樹，中西一郎 : 1854 年安政南海地震による愛媛県最南端（愛南町）での地震動 · 津波被害 · 地下水位変化，地震，Vol. 68, No. 4，pp.107-124（2015）.

[4] Text Encoding Initiative : TEI P5 Guidelines, 入手先〈http://www.tei-c.org/guidelines/p5/〉（参照 2018-09-06）.

[5] Syncro Soft : oXygen XML Editor, 入手先〈https://www.oxygenxml.com/〉（参照 2018-09-06）.

[6] University of Hamburg: CATMA 5.0, 入手先〈hhttp://catma.de/〉（参照 2018-09-06）.

[7] Ford, Bryan: Parsing expression grammars: a recognition-based syntactic foundation, ACM SIGPLAN Notices. Vol. 39. No. 1. ACM, 2004.

[8] 漢字データベースプロジェクト : 青空文庫の注記記法，入手先〈hhttp://kanji-database.sourceforge.net/aozora/grammar.html〉（参照 2018-09-06）.

[9] Ford, Bryan: Packet parsing: a practical linear-time algorithm with backtracking, PhD Thesis, Massachusetts Institute of Technology, 2002.

[10] Futago-za Ryuu: PEG.js,入手先〈https://pegjs.org/〉（参照 2018-09-06）.