分散レイヤーとしての
デジタル テキストに向けて

ウィッテルン　クリスティアン[1]

はじめに

中国学研究者として、過去 35 年間の主たる関心は、研究環境で利用するための漢籍のデジタル表現にあった。当時 TEI と SGML に出会ったことは、人生を変える経験であり、その後長年にわたる活動の方向を定めた。その成果の一つが、大規模な仏教文献コーパスである。[2]

学術界での受け止め方はさまざまであった。そこで、より取り組みやすいテキスト符号化の方法を求め、Emacs の org-mode から派生した単純なデータ形式を用い、マークアップを可能な限り暗黙化する方針を採用した。[3] この方法を十数年続けた後、別のプロジェクトをきっかけに再び TEI に戻った。ただし、その時点では、研究者がソースを直接編集するという考えはすでに断念し、技術的な複雑さや山括弧を利用者から隠す共同作業プラットフォームの構築に集中していた。[4]

このプロジェクトを始めて 10 年ほど経った頃、漢籍リポジトリ と 漢學文典(TLS) を統合するために同じ枠組みに載せようとする試みから、テキストのあらゆる特徴を一つの容器に詰め込むことをやめ、Docker イメージにいくらか似た考え方を採用するという着想に至った。すなわち、テキストを個別の構成要素として組み合わせ可能な複数のレイヤーに分け、レシピ（「Dockerfile」を想像してほしい）に従ってコンポーザーがそれらを統合する。コンポーザーはレシピに記された処理を実行し、読者の希望に応じた一時的なテキストの姿を生成する。注釈、翻訳、デジタル・ファクシミリを含めるかどうかを選び、その時点で必要な形式、たとえば印刷用の PDF、ウェブ用の HTML、電子書籍リーダー用の ePub を出力するのである。

いつものことながら、実際に問題となるのは細部である。JSON は現代の多くの通信で事実上の共通言語となっているが、長期保存用のファイル形式としては必ずしも扱いやすくない。その用途には YAML の方が適している。数週間の試行錯誤の結果、目的に適合すると同時に、中国語テキストが数千年にわたって受け継いできた特徴を思わせるデータ形式が形になった。すなわち、巻物に書かれたテキストを束としてまとめ、最小限のメタデータだけを添えるという形である。

本稿では、漢籍という特定の用途に特化したテキスト形式を紹介したい。他の用途にはあまり向かないかもしれないが、模倣や応用を促す着想はいくつか含まれてい

1　京都大学人文科学研究所付属人文情報学創新センター

2　CBETA: https://cbetaonline.dila.edu.tw/en/ 。XML ソース: https://github.com/cbeta-org/xml-p5 。

3　この方式によるテキスト・リポジトリ: https://kanripo.org 。ソース: https://github.com/kanripo 。

4　Thesaurus Linguae Sericae（TLS）: https://hxwd.org 。ソース: https://github.com/tls-kr 。

るはずである。インターフェースの初歩的な実装はすでにオンラインで公開されており、テキストのソース形式も利用できる。[5]

漢籍の形態を概観する

中国語で *juan*（巻）と呼ばれる巻物は、長い間、テキスト制作の基本単位であった。巻物の長さはある程度調整できたため、巻は物理的な単位であるだけでなく、テキストの論理的な単位としても機能するようになり、特定箇所を作品名と巻数で示す慣習が成立した。

しかし、一つの巻の内部には、ごくわずかな段落表示、すなわち行末まで文字を埋めずに改行するような例を除けば、ほとんど構造がなかった。何百字、何千字もの文字列が、ほぼ途切れることなく連続していた。

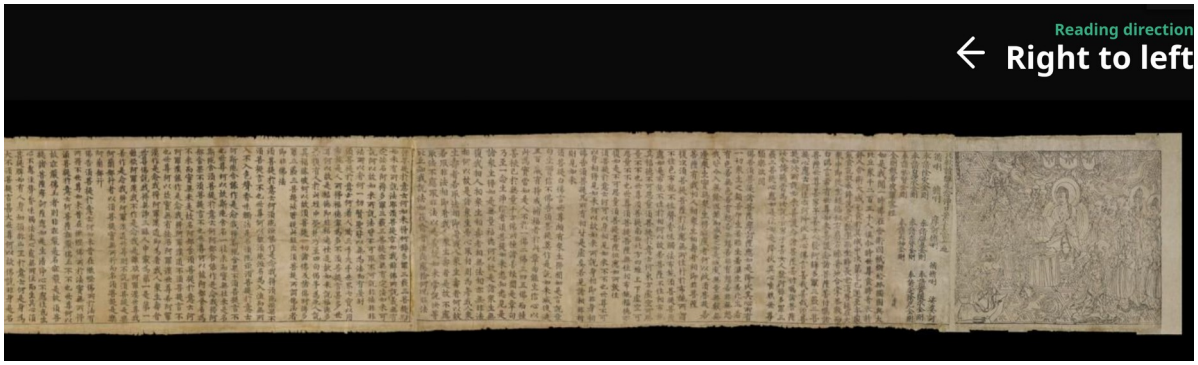

図 1: Or. 8210[6]

もちろん、より複雑な内部構造を持つテキストも存在する。たとえば、本文の間に注釈が織り込まれたものがある。この場合、注釈は通常の一文字が占める正方形の四分の一ほどの小さな字で書かれることがあり、その結果、一つのテキスト内に少なくとも二つの異なる論理的な流れが形成される。

5 https://bunkankun.org 。関連資料とコード: https://github.com/bunkankun 。テキスト: https://github.com/bkkbooks 。

6 https://idp.bl.uk/collection/51FDAEAFB4A24E2E9981692A98130BC8/

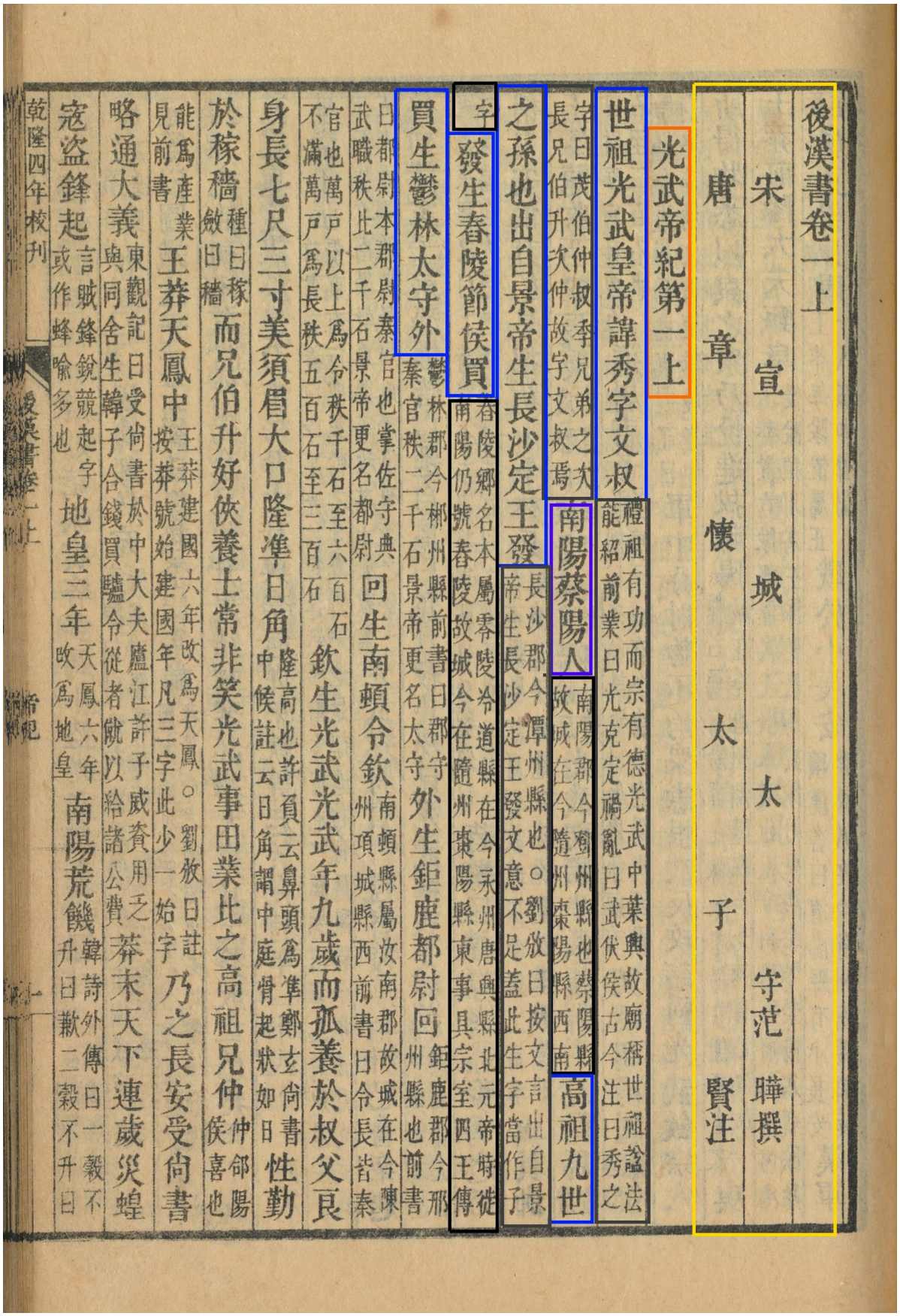
後漢書卷一上
宋 宣 城 太 守 范 曄 撰
唐 章 懷 太 子 賢 注
光武帝紀第一上
世祖光武皇帝諱秀字文叔 禮祖有功而宗有德光武中葉興故廟稱世祖蓋法
字發生春陵節侯買 春陵鄉名本屬零陵今道縣在今永州唐興縣北元帝時徙
買生鬱林太守外 鬱林郡今郴州縣前書曰郡守
之孫也出自景帝生長沙定王發 長沙郡今潭州縣也○劉攽曰按文言出自景
南陽蔡陽人 南陽郡今鄧州縣也蔡陽縣
高祖九世 故城在今隨州棗陽縣西南
能紹前業曰光克定禍亂曰武伏侯古今注曰秀之
南陽荒饑韓詩外傳曰一穀不
升曰歉二穀不升曰

図 3 は、同じ箇所が現在 Kanripo でどのように表示されているかを示す[7]。

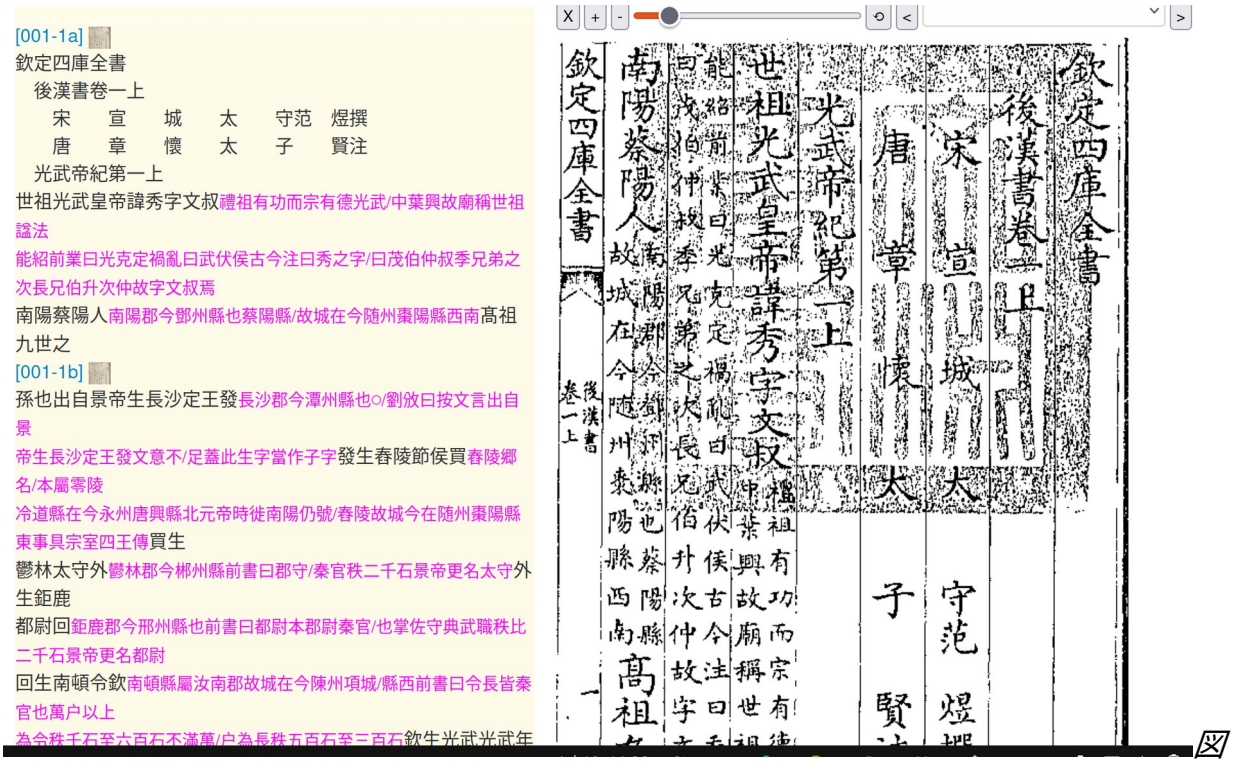

図 3: 漢リポ における同一箇所の表示

デジタル表現とシステム設計

TEI ではどうか

TEI/XML が中国語テキストを十分に表現できることは言うまでもなく、すでに 30 年以上にわたって利用されている。最大規模のもの、少なくとも一般公開されている最大のコレクションは、CBETA 電子佛典集成（CBETA Collection of Buddhist Scriptures）であろう。[8] もちろん、ほかにも大規模なコレクションが存在する可能性はある。別のプロジェクトでは、こうしたテキストを共同研究するためのプラットフォーム開発に長年取り組み、それが実際に可能であることを示してきた。

しかし年月を経るうちに、表現そのものよりも、この構造が提供する機能や制約を含むシステム全体の設計に問題を感じるようになった。

共同作業プラットフォームの目標の一つは、テキスト研究のワークフロー全体を支援することであった。そのワークフローは、まずデジタル・ファクシミリを参照しながら、異本や平行箇所を比較し、テキストを構成する正確な文字を確定することから始まる。次に、前付け、本文、後付けといった部分、さらに確認できる範囲で

7 https://www.kanripo.org/text/KR2a0009/001#1a
8 TEI P5 XML ソース: https://github.com/cbeta-org/xml-p5 。

内部構造を同定する[9]。 その後、節、段落、句読点を追加する。この段階で、プロジェクトの主要作業である注釈と翻訳に取りかかれる状態になる。

これは当該プロジェクトにおける TEI の利用方法に起因する問題だった可能性もある。しかし実際には、テキストを共同作業プラットフォームの外部で準備し、十分に整った段階で初めてアップロードするのような作業の流れを取る必要があった。その結果、本来望ましい循環型のワークフローではなく、ウォーターフォール型のワークフローになってしまった[10]。

では、そもそもテキストとは何か

今年予定されていた主要課題の一つは、大量の追加テキストを既存プラットフォームへ統合し、収録テキスト数を約 10 倍に拡大することであった。最大の問題は、今回の追加分は未構造化の生データであり、既存システムへ簡単には追加できなかったことである。

何度かの研究会での議論を通じて、私たちは「テキストとは何か」「それをどのように扱うべきか」という基本的な前提になるそのものを問い直すことになった。第一千年紀の巻物に見られる生のテキスト表現を出発点にすれば、後世に加えられたものはすべて、テキストの本質的要素ではなく、解釈的な付加物と見ることができる。

そこで私たちは、テキストを固定された出版物としてではなく、制作者、さらには読者の都合に応じて、さまざまな部分から構成される複合的な人工物として捉えるようになった。

それをコンピューターにどう伝えるか

この考えから、TEI に依存しない解決策を探すことになった。マークアップによる記述は必要だが、インラインではなくスタンドオフ方式であることが望ましい。ある実験ではテキスト表現に JSON を用いたが、編集しにくいうえ、git ベースの環境では管理もしにくかった。そこで YAML の方が適していると考え、現在の実装とデータ形式が形作られた。

現時点ではまだ実験的な段階にあるが、今日の生成 AI エージェント型コーディングを利用すれば、短期間で実用的な実装に仕上げ、コーパス全体で検証できるだろう。ただし、まだ完全に欠けている構成要素もある。

9 理想的には、各工程は必要に応じて循環的に適用されるべきである。すべての工程を完全に終えてから次へ進むのではなく、その時点で必要な部分だけを処理できる方がよい。

10 ある時点では、テキスト確定に必要な本文批判作業の多くについてユーザーインターフェースを実装したが、使いにくく壊れやすかったため、実際にはほとんど利用されなかった。

分散レイヤーとしてのテキスト

便宜上、テキストは巻（juan）単位に分割し、それぞれを独立した YAML ファイルに格納する。これらのファイルを「テキスト・バンドル」にまとめ、マニフェスト・ファイルでバンドル内の運用と管理用メタデータを宣言する。少なくとも一つの表層版を持ち、必要に応じて数に制限のない追加版を、`editions` サブディレクトリの下に同様の構成で配置できる。

最小限のテキスト・コレクションは、図 4 に示すようなファイル群から構成される。

- 全ファイルの目録を含むマニフェスト・ファイル `KR6c0128.manifest.yaml`
- 実際のテキストを含む `KR6c0128_001.yaml` のようなテキスト・ファイル
- 追加情報を持つ「資産」。たとえば、メタ情報をテキストのオフセットへ結び付けるマーカーを含む `assets/KR6c0128_001.markers.yaml`。付録にはこれらのファイル例を示している。

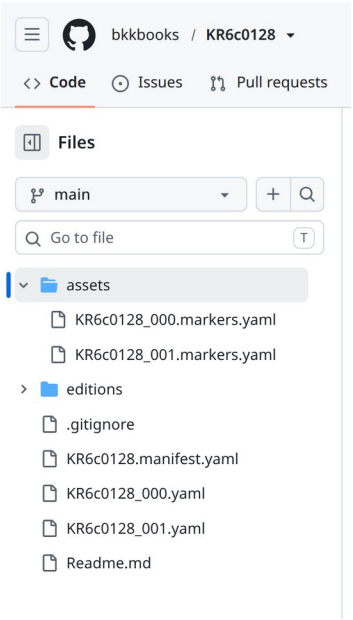

図4: 最小構成のテキスト・バンドル

保存用形式

すべてのテキスト・フィールドには、場所や型にかかわらず、その内容を検証するためのハッシュ・フィールドが付随する。

一つの巻ファイルは `front`、`body`、`back` の三つのセクション を持つ。非空でなければならないのは `body` だけであり、ほかは任意で、空であれば省略できる。追加のメタデータ・フィールドも利用可能である。`body` と `back` のテキスト要素は、必要に応じてさらに分割できる。

典型的な `front` には、その巻がより大きなコレクションのどこに位置するかを示す冒頭行、作品名、巻の通し番号、本文に関係する人物名と役割を示す責任表示が含まれる。`back` には巻末行が含まれる。序文、跋文、奥書などのパラテキストをどこに置くかは固定しない。形式を採用するプロジェクトの判断により、`body` に含めても、`front` または `back` に分離してもよい。

`body` には一つの `text` 要素があり、その巻全体の正規文字列を保持する。このストリームには空白、句読点、改行などを含めない。それらは巻の論理的なマーカー集合へ抽出される。

マーカーは最低限、`type` と `offset` を持つ。必要に応じて `id`、`content`、およびマーカー型に応じた追加の構造化情報を含める。マーカー型の集合は開放されており、小さな中核語彙だけを別途定義する。

マーカーは物理的には二つの場所に格納できる。バケットは、巻単体を確認するときに有用な構造マーカーを `markers` リストとしてインラインに持てる。一方、大部分のマーカーは、バンドルの `assets/` ディレクトリにある巻ごとのマーカー資産へ格納し、マニフェストからピン留めできる。

利用側は、インライン・マーカーとマーカー資産内のマーカーを一つの論理的な集合として扱い、オフセット順に並べる。これにより、同じテキスト位置指定モデルを保ちながら、巻ファイルを簡潔にできる。

一つのテキストは複数の版で表現できる。各版は「master」版から参照できるほか、レシピから直接指定することもできる。

正規化

マーカーは生のオフセットによって対応するテキスト・ファイルへ結び付く。この仕組みは潜在的に壊れやすいため、テキスト・フィールドの完全性を確保するために特別な注意が必要である。

中国語テキストでは、正規文字集合を定義する必要がある。文字は Unicode から選ぶが、一部を除外し、別の文字を追加する。この基準となる文字集合は、他の参照資産とともにファイル形式の一部として提供される。

原資料は決定的な手順によって正規形へ変換される。ソースを変更するすべての工程はマーカーとして記録されるため、正規テキストとマーカーから原資料を完全に復元できる。

手順は次の順序で適用する。

1. **Source（入力文字）** — 入力は UTF-8 で符号化された Unicode として扱う。
2. **Entity expansion（エンティティ展開）** — 漢リポ資料の `&KRxxxx;` 形式の実態参照や TEI の `<g/>` 要素など、現行 Unicode に存在しない文字を指すエンティティ参照を、バンドルが宣言するエンティティ符号化方式に従い、

Supplementary Private Use Area（SPUA-A、U+F0000–U+FFFFD）のコードポイントへ展開する。展開は決定的であり、マーカーは生成しない。PUA コードポイントと宣言済みの符号化方式の組み合わせだけで元のエンティティ参照を復元できるからである。この方法で生成された PUA コードポイントは、正規テキスト・ストリームの正式な構成員であり、他の文字と同様にオフセット、テキスト・ハッシュ、後続のマーカー処理に参加する。

3. **Unicode normalization（Unicode 正規化）** — NFC を適用し、基底文字を合成済み形式にする。
4. **Extraction of layout features（レイアウト特徴の抽出）** — 空白、句読点、インデント文字、改ページ・改行、欄の区切り、その他のレイアウト・構造・パラテキスト情報を担う文字を `text` 要素から除去し、残ったテキスト・ストリームへのオフセットを持つマーカーとして出力する。
5. **Substitution to canonical characters（正規文字への置換）** — 抽出後のストリームにある文字のうち、正規文字集合に属さないものを正規等価文字へ置換する。各置換について、オフセット、置換された文字または文字列、置換理由を記録するマーカーを生成する。
6. **Hash（ハッシュ）** — 得られたテキスト・ストリームを UTF-8 で符号化し、`text` 要素に付随する `hash` フィールドへの入力とする。

オフセットは、置換後のテキスト・ストリームに含まれる Unicode コードポイント数で数える。異体字セレクタや類似の結合文字は独立したオフセット対象とはせず、直前の基底文字に結び付けたままにする。

宣言された集合の外にある文字をソースに含めること自体は禁止しない。正規化時に置換し、その置換を substitution マーカーとして記録する。除外の一般的な理由には、推奨形式が明示されている互換文字、非推奨コードポイント、第三者の符号化方式による場当たり的な私用領域割り当て、意図的にまだ対応していない Unicode ブロックなどがある。

マーカー

マーカーは、正規化の過程でテキスト・ストリームから取り除かれた情報、またはストリーム外部からテキストを注釈する情報を、正規テキスト上の特定位置へ再び結び付ける仕組みである。

バケットの論理的マーカー集合は、存在する場合にはバケット内のインライン `markers` リストと、マニフェストで宣言された巻のマーカー資産にある対応リストを合わせて構築する。したがって、インライン・マーカーは保存上の便宜にすぎず、別の意味クラスではない。両方にマーカーがある場合、利用側はオフセット順に統合し、同じオフセットを持つマーカーについてはソース内の順序を維持する。

マーカーには二つの形がある。ポイント・マーカーは単一位置に作用し、`offset` だけを持つ。レンジ・マーカーはテキスト・ストリームの一部分に作用し、追加の `length` フィールドを持ち、区間 `[offset, offset + length)` に適用される。どち

らの形を取るかはマーカー型によって固定される。

すべてのマーカーは次の必須フィールドを持つ。

- `type`: マーカーの種類
- `offset`: 正規テキスト・ストリーム内でマーカーが始まるコードポイント位置
- `length`: マーカーが覆うコードポイント数。レンジ型では存在し、ポイント型では存在しない

任意フィールドはマーカー型ごとに定義され、`id`、`content`、`note`、`responds-to` などを含められる。`responds-to` は別のマーカーの `id` を参照し、そのマーカーへの注釈または拡張であることを示す。さらに型固有の構造化フィールドも利用できる。

`id` を持つマーカーは他の場所から参照できる。`id` は、修飾されない限り巻内でローカルである。`id` の一意性は、一つの物理ファイル内だけでなく、組み立てられた論理的マーカー集合全体に対して検査する。

マーカー型の集合は開放されている。レイアウト、構造、声部、置換のために、小さな中核語彙を定義する。`page break`、`line break`、`indent`、`punctuation`、`paragraph break`、`register divider`、`head`、`comment`、`substitution` はポイント型であり、`voice` はレンジ型である。各プロジェクトは独自の型を拡張できるが、中核語彙にない型には、その型を導入したプロジェクトの名前空間を付けるべきである。

XML からのインポートでは、存在自体は保存すべきだが、その意味が中核インポーターで明示的にモデル化されていないソース要素に対し、汎用ポイント・マーカーを生成できる。このマーカーは `type: xml-element` を持ち、必須フィールドとして `offset`、`name`、`role` を含む。`name` は、ソース・メタデータの他の箇所と同じ接頭辞形式で元要素名を記録し、`role` は通常 `open` または `close` である。必要に応じて `id` やソース属性を保持する `attrs` マッピングなどを追加できる。

ページ区切り、行区切り、`tls:head`、`tls:seg` は、それぞれ専用マーカーで表現し、`xml-element` マーカーとしては出力しない。

既定では、構造上およびナビゲーション上重要なマーカーをインラインに残す。具体的には `tls:head`、`tls:div-start`、`tls:div-end`、`cbeta:juan-start`、`cbeta:juan-end`、`cbeta:mulu`、および目次から直接参照されるマーカーである。レイアウト・マーカー、句読点、改ページ・改行、voice マーカー、異文マーカー、substitution マーカー、コメント、その他の大容量プロジェクト・マーカーは通常、マーカー資産に置く。この既定値はパッケージング方針であって意味上の制約ではないため、すべてのマーカーをインラインに持つ旧バンドルも引き続き有効である。

声部（Voices）

voice マーカーは、正規テキスト・ストリームのある範囲が、名前を持つ特定のテキスト・レイヤーに属することを記録する。これは、根本文とその注釈のように、互いに織り込まれた複数の流れを分離する仕組みである。利用側は、任意の一つのレイヤーを抽出したり、複数のレイヤーを並列表示したり、出版時の交錯した配置を再現したりできる。

voice マーカーはレンジ型であり、通常の必須フィールドに加えて次を持つ。

- `type`: リテラル値 `voice`
- `length`: コードポイント単位の範囲長
- `name`: この範囲が属するテキスト・レイヤー名

`name` の語彙は開放されているが、プロジェクト間の可読性を保つため、次の小さな標準集合を定義する。

- `root` — 注釈の対象となる主要テキスト
- `commentary` — 根本文に対する注釈レイヤー
- `subcommentary` — 注釈に対する再注釈レイヤー
- `gloss` — 短い行間注。典型的には行間に小さな文字で配置される
- `note` — 注釈、語釈、異読、引用など、具体的役割が未確定の括弧付き範囲。利用可能な手がかりが `(…=)=` のような句読記号による囲みだけで、導出処理がそれ以上の意味を断定できない場合に用いる

プロジェクトは、他のマーカー語彙と同様に、`kr:apparatus`、`tls:swl` などの接頭辞を付けて拡張する。

voice マーカーは、`id`、`note`、`responds-to` などの任意フィールドを持てる。`responds-to` は、その声部が応答する別の voice マーカーの `id` である。subcommentary は通常 commentary に応答し、commentary は root に応答する。この関係は多対一であり、一つの root 範囲に対し複数の commentary 範囲が対応し、それぞれが同じ root 範囲を `responds-to` の対象として指定できる。

声部は巻の単一テキスト・ストリームと共存し、声部ごとに別の `text` 要素を持つわけではない。巻の正規テキストには、すべての声部の内容がソース順に収められ、voice マーカーがそれを名前付きの範囲へ切り分ける。

「root のみ」を抽出する利用側は `name=root` で絞り込み、オフセット順に各範囲を読む。交錯した原形を再現する利用側は、すべての voice マーカーをオフセット順にたどる。どちらの表示も同じ正規テキストと同じハッシュから導出される。

ストリーム外の内容を持つ軽量なポイント・マーカー `comment` と `head` は、短い編集者注や散発的な見出しなど、まばらな行間素材に引き続き利用できる。挿入テキストを正規化・ハッシュ化・マーカー付与可能な第一級の範囲として扱うほどではない場合に適している。内部レイヤーが十分に密で、それ自体を正式に処理する必要がある場合には voice が適切である。

例

短い注釈文が、根本文の行とそれに対する注釈を交互に並べるとする。巻の正規テキストを一つのストリームとして表示すると、次のようになる。

```
色不異空謂色相虚幻不離空性空不異色謂空性遍周不離色相
```

巻は四つの voice マーカーを持つ。

```
markers:
 - {type: voice, offset: 0, length: 4, name: root, id: r1}
 - {type: voice, offset: 4, length: 9, name: commentary, responds-to: r1}
 - {type: voice, offset: 13, length: 4, name: root, id: r2}
 - {type: voice, offset: 17, length: 9, name: commentary, responds-to: r2}
```

`name=root` を抽出する利用側は、マーカー一覧をオフセット順にたどり、範囲 `[0,4)` と `[13,17)` を連結する。その結果は 色不異空空不異色 となる。

交錯したソースを再現する利用側は、すべての voice マーカーをオフセット順にたどり、各範囲を順に出力する。これにより元のストリームが得られる。どちらの表示も、同一の正規テキストと同一のハッシュから導出され、基盤となる巻を変更する必要はない。

オーバーレイ・バンドル

オーバーレイ・バンドルとは、別のバンドルのテキスト・ストリームを変更せずに、追加のマーカーを提供するバンドルである。そのマニフェストは対象を宣言する。対象は、参照先バンドルの正規識別子とハッシュによって指定される。オーバーレイ側の巻ファイルは、対象バンドルの対応する巻内のオフセットを指定するマーカー集合を持つ。

対象バンドル自体は影響を受けない。ハッシュは変わらず、既存のピンも有効なままである。オーバーレイは通常のバンドル方式に従い、独立して参照、ハッシュ化、版管理できる。

この仕組みの動機は明快である。有用なマーカーは、バンドルの公開後にも作成できる。また、対象バンドルの所有者でない第三者が、原著者の想定しなかった目的のために作成してもよい。たとえば、翻訳のための分節、古典テキストへの現代的な句読点の付与、機械可読な形では示されていなかったレイアウトを分離する voice マーカー、独立配布される学術注釈などである。利用側は、レシピを通じて対象と任意のオーバーレイを組み合わせる。

このような分散レイヤーによって、システムは多様な要求や利用場面に適応できる。たとえば研究論文は、引用箇所をオーバーレイとして保持し、書誌情報を扱うのと似た方法で、レシピを使ってコーパスから原文と対応する翻訳を取得できる。

読者はそのオーバーレイを利用して、引用文を原文脈の中で確認し、コメントを付けたり、自分の翻訳を追加したりできる。

おわりに

ここで紹介したテキスト形式は誕生してまだ数週間であり、実験と実装のごく初期段階にある。少数のテキストによる初期試験を経て、現在は三つの異なるソースからのテキストをより本格的に変換した結果を GitHub の `bkkbooks` アカウントで公開しており、その数は約 12,500 タイトルに達している。

インポート、エクスポート、ウェブサーバー、フロントエンド UI など、さまざまなメンテナンス作業のためのコードはすべて `bunkankun/bkk` で利用できる。

次の段階では、デジタルテキストへ適切な分節を追加する予定であり、これもマーカーとして表現する。ソース内で独立した見出し行から始まるテキスト区分は、既存マーカーを分析すれば容易に認識できる。

現代的な意味での段落は原資料に存在せず、したがって表示もされていない。しかし予備実験によれば、適切なプロンプトと実行枠組みを与えれば、大規模言語モデルをこの作業に利用できる。句読点についても同様である。

そして、この点で本データ形式の強みが明確になる。異なるモデルからの応答を、基礎テキストへ一切影響を与えることなく、容易に重ね合わせ、比較し、さらに処理できるからである。

現時点では、これは研究ワークフローの最初の数段階、すなわち十分に理解されたテキストを確立する作業を担う大きな機会に見える。その後の経路としては、漢籍リポジトリまたは TEI ベースの TLS 形式へ進む選択肢を残せる。

これら三つのプロジェクトが今後どのように発展するかは、時間が明らかにするだろう。