人文学のためのテキストデータ構造化のチュートリアル

第8章

構造化テキストの活用手法

永崎研宣

version 1.0
2026.3.21 作成

本資料は、文部科学省委託事業「人文学・社会科学のＤＸ化に向けた研究開発推進事業」（JPMXP1624）において、学校法人慶應義塾が、大学共同利用機関法人人間文化研究機構から再委託を受けて作成したものです。本資料の利用にあたっては、出典を必ず記載するなど、「文部科学省ウェブサイト利用規約」を準用（ただし、商用利用は不可とする。）してください。

1．基本的な考え方

TEI/XML に準拠して作成されたテキストデータは、XML 向けの汎用ツールや、TEI 専用に設計されたソフトウェアを通じて、さまざまな形で利活用できる。紙媒体の研究成果は、国立国会図書館への納本制度に象徴されるように、保存と利用の継続性が制度的に担保されてきた。一方、デジタルデータの持続可能性は必ずしも自明ではなく、利用環境の変化によって容易に失われうる。

TEI は、この問題意識を背景として策定されてきた。すなわち、特定のソフトウェアや表示形式に依存しない、オープンで永続的な記述規則を定めることで、研究者が一度記述した解釈や判断を、長期にわたって再利用可能な形で保存することを目的としている。現在の TEI ガイドラインは、そのような設計思想のもとで継続的に改訂されてきた成果である。

TEI 協会の技術委員会は、民主的な手続きを経て選出され、定期的な会合に加えて メーリングリストや GitHub 上で公開の議論を行っている。改訂の履歴や議論の過程は誰でも参照可能であり、TEI が特定の研究者や組織に閉じた仕様ではないことがわかる。TEI/XML 文書は、スキーマに基づく自動検証を経て作成されることが多く、その結果、形式的な一貫性を保ったまま共有できる。

作成された TEI/XML ファイルは、そのまま Web サイトや Zenodo などの研究データリポジトリで公開される場合もあれば、TEI/XML 専用のリポジトリである TAPAS に登録される場合もある。また、XSLT や各種プログラムを用いて表示を整え、Web ページ、電子書籍（ePub）、印刷用 PDF、あるいはデータベースとして提供することも可能である。表示用のソフトウェアが陳腐化しても、元となる TEI/XML ファイルが残っていれば、新たな処理系を用意することで再利用できる。この点において、TEI/XML は紙媒体に近い持続可能性を、デジタル環境において実現しようとする試みである。

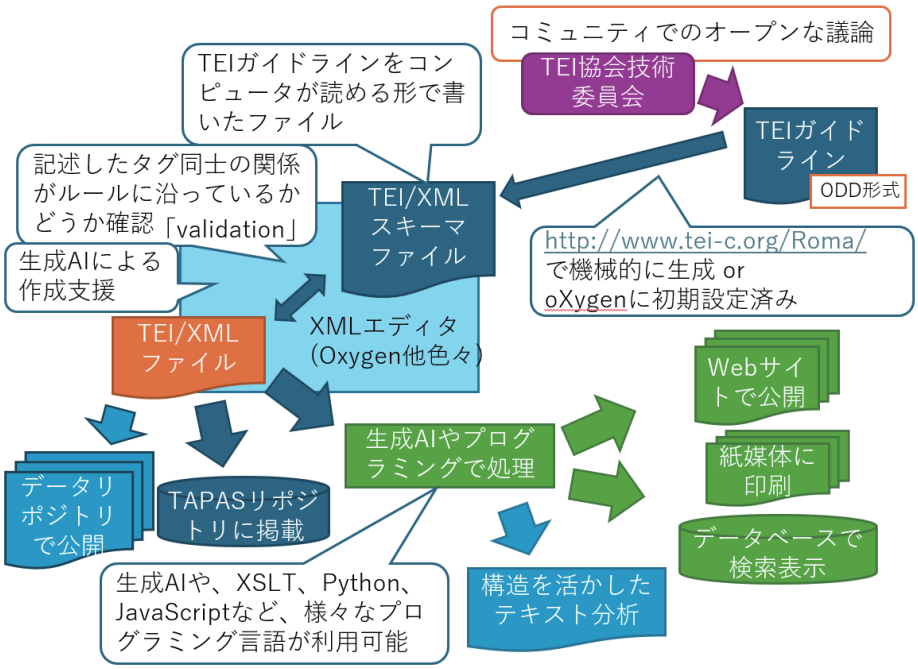
もっとも、TEI/XML ファイルは、そのまま人間が読んで理解しやすい形式ではない。記述が精緻になるほど、可読性は低下する。そのため、深い解釈を記述したデータほど、それに対応した変換・表示の仕組みを併せて設計する必要がある。

さらに近年では、生成 AI（大規模言語モデル）を研究支援に用いる試みが急速に広がっている。その際、TEI/XML ファイルは、単なるテキストの供給源ではなく、構造化された知識資源として重要な役割を果たしうる。TEI によって明示された段落構造、注記、人物・地名・日付といった意味的要素、さらには書誌情報や来歴情報は、生成 AI にとって文脈理解や情報抽出の手がかりとなる。とりわけ、検索拡張生成（Retrieval-Augmented Generation, RAG）のように、外部データを参照しながら応答を生成する枠組みにおいては、TEI/XML のような構造化テキストは、信頼性の高い参照データとして利用しやすい。

この観点から見ると、TEI/XML による記述は、従来の可視化や検索のためだけでなく、将来

の自動処理や知識生成を見据えた研究基盤の構築でもあると言える。人文学研究者が行った解釈や判断を、機械可読な形で保存しておくことは、生成 AI 時代における人文学的知の再利用と継承にも直結する課題である。

以上を踏まえ、本章では、TEI/XML ファイルをどのように検索・処理・変換し、研究に活用できるかについて、具体的な操作を通じて理解することを目的とする。



2．Oxygen 上での高度な検索

TEI/XML ファイルを有効に活用する方法の一つとして、Oxygen XML Editor が備える高度な検索機能がある。検索結果を一覧として取得し、表計算ソフト等で処理するだけでも、簡単な分析や整理は可能である。

Oxygen では、XML の階層構造や属性を考慮した検索が可能であり、さらに同一フォルダ内にある複数の XML ファイルをまとめて対象とすることもできる。

本節では、まず XML の構造を考慮しない基本的な検索を確認したうえで、構造を活かした検索、さらに XPath を用いた検索へと進む。

2-1．XML の構造を活かした検索

① 検索の準備

検索を行うためには、TEI/XML ファイルが必要である。ここでは、実践演習 3 で作成した soseki_letter_ex3.xml を用いる。実践演習 3 が未完了の場合には、完成版である soseki_letter_ex3_finished.xml を用いてもよい。

② 基本的な検索

Oxygen XML Editor では、Ctrl+F（macOS では Cmd+F）によって検索ダイアログを開くことができる。あるいは、メニューバーの「検索」から「検索／置換」を選択してもよい。

検索ダイアログでは、「検索」欄に文字列を入力し、「検索」または「すべて検索」を実行する。後者を用いると、画面が上下に分割され、下部に検索結果の一覧が表示される。

この方法で検索を行うと、本文中の文字列だけでなく、タグ名として用いられている文字列や属性値に含まれる文字列も、区別なく検索される。構造化された XML 文書に対してこのような検索を行うと、必要以上に多くの結果が得られてしまう。

そこで Oxygen では、「XML 検索オプション」という機能が用意されている。「検索／置換」ダイアログで「XML 検索オプションの有効化」にチェックを入れると、検索対象を限定する設定項目が表示される。ここで「属性値」を選択し、「検索」欄に「鏡」と入力して「すべて検索」を実行すると、属性値に「鏡」を含む箇所、すなわち本例では夏目鏡子への参照箇所のみが一覧表示される。

このほかにも、「次の中のみ検索」を用いることで、検索対象を要素内容やコメントなどに限定できる。構造を意識した検索を行うことで、TEI/XML ファイルの利点をより活用できる。

2-2．XPath 検索の考え方

XML 文書では、本文の文字列に対して要素（タグ）が付与されており、さらに要素には属性とその値が与えられる場合がある。このように、XML は単なる文字列の集合ではなく、要素・属性・値から成る構造化されたデータとして記述されている。

そのため、XML 文書に対しては、単に文字列を探すだけでなく、「どの要素の中にあるか」「どの属性に付与された値であるか」といった条件を用いて、検索対象を絞り込むことができる。さらに、XML 文書では要素同士が親子関係を持つ階層構造を成しており、この階層関係を利用することで、検索条件をより精密に指定することが可能となる。

このような、XML 文書の構造を前提とした検索や指定を行うための記法が XPath である。XPath は、XML 文書を木構造として捉え、要素を「経路（path）」として指定することで、文書内の特定の位置や条件にあるデータを選択するための仕組みである。要素名や属性名を単なる文

字列として扱うのではなく、文書構造の中での位置関係として記述する点に特徴がある。

XPath は長年にわたって拡張・発展してきており、条件分岐や関数など、部分的にはプログラミング言語に近い機能も備えている。しかし、ここではその全体像を扱うのではなく、検索の絞り込みという観点に限定し、XPath の基本的な考え方と有用性を体験的に理解することを目的とする。

2-3．XPath を用いて要素を検索する

それでは、実際に XPath を用いた検索を試そう。Oxygen XML Editor の「検索／置換」ダイアログにおいて、「XPath」欄に placeName と入力し、「正規表現」にチェックを入れる。さらに、「検索」欄には半角のドットとアスタリスクから成る .* を入力し、「すべて検索」を実行する。

ここで用いている .* は、正規表現において「0 文字以上の任意の文字列」を意味する記号である。XPath で要素名を指定し、正規表現として .* を組み合わせることで、「placeName 要素に囲まれたすべての文字列」を検索対象とすることになる。その結果、文書内に存在するすべての <placeName> 要素が抽出され、ここではおよそ 21 件がヒットする。

この検索結果は、文書中のすべての <placeName> を対象としたものである。<placeName> は本文を構成する <body> 要素の内部だけでなく、<correspDesc> や <back> といった本文以外の箇所にも記述されている。そのため、この結果には、書簡本文に登場する地名以外のものも含まれており、実際に本文中に現れる地名の数よりも多くなっている。

ここで、「書簡本文に登場する地名だけを数えたい」「本文中の用例だけを確認したい」といった目的が生じる。このような場合に、XPath 検索の利点が明確になる。

2-4．XPath によるパス指定の考え方

XPath では、要素の階層構造を「/」で区切ったパスとして記述することで、検索対象を限定できる。たとえば、XPath 欄に次のように記述して「すべて検索」を実行する。

```
/TEI/teiHeader/fileDesc/titleStmt
```

この指定は、文書のルート要素から順に TEI → teiHeader → fileDesc → titleStmt という階層をたどった先にある要素を対象とすることを意味する。このように、XPath は XML 文書の階層構造を、URL に似たパス表記によって辿る仕組みである。

一方で、文書内のどこかに存在する要素を対象としたい場合には、階層をすべて明示する必要はない。そのような場合には、次のように // を用いた省略表記を用いる。

```
//titleStmt
```

この指定は、「文書内の任意の位置にある titleStmt 要素」を意味する。// は、途中にどのような要素が挟まっていてもよいことを示す記法であり、構造が複雑な文書を扱う際に有用である。

2-5．正規表現についての補足

ここで用いている正規表現（regular expression）は、任意の文字列の集合を、記号を用いて簡潔に表現するための記法である。たとえば、. は任意の 1 文字を表し、* は直前の文字や記号が 0 回以上繰り返されることを意味する。

また、タグのような表記を対象としたい場合には、`<.+?>` のような記法を用いることがある。この表現は、「`<` で始まり、何らかの文字が 1 文字以上続き、最初に現れる `>` で終わる文字列」を意味する。ここで + は 1 回以上の繰り返しを、? は最短一致を指定する記号である。

正規表現には他にも多くの記号や用法が存在するが、詳細については専門書や解説記事を参照されたい。XPath と正規表現を組み合わせることで、XML 文書に対する検索の自由度は大きく高まる。

2-6．XPath でエレメントを絞り込む

ここまでの検索では、文書内に存在するすべての `<placeName>` 要素を対象としていた。しかし、書簡に登場する地名を分析したい場合、実際に関心があるのは書簡本文に現れる地名であり、書誌情報や付記部分に含まれる地名は必ずしも対象とならない。

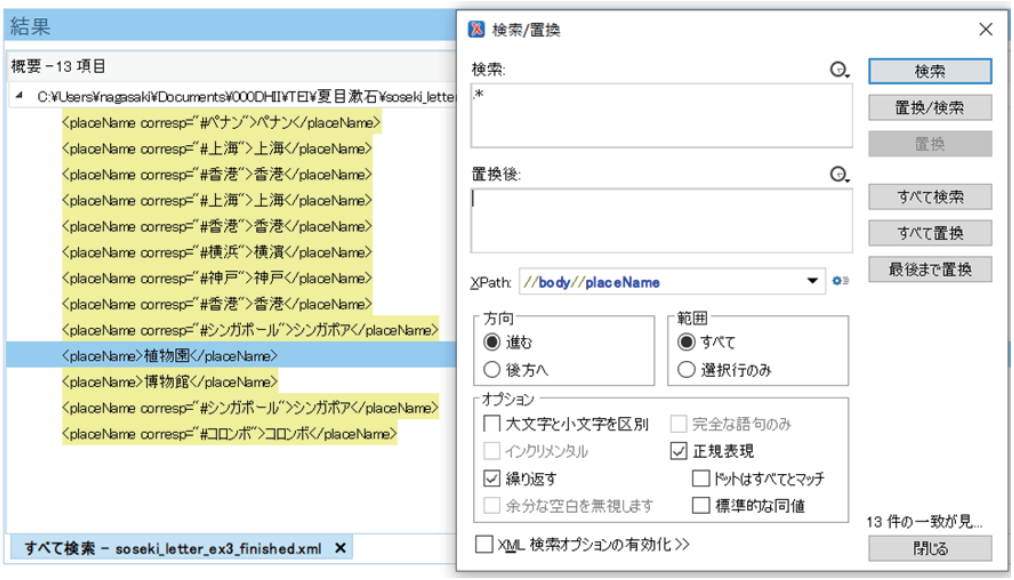
この XML 文書では、書簡の本文は `<body>` エレメントとして記述されている。したがって、本文中の地名だけを検索したい場合には、「`<body>` エレメントの内部に現れる `<placeName>`

要素」に検索対象を限定すればよい。このように、要素の階層関係を利用して検索範囲を制限できる点が、XPath の大きな特徴である。

そこで、XPath の欄を次のように書き換え、検索欄にはこれまでと同様に .* を入力して「すべて検索」を実行する。

```
//body//placeName
```

この XPath 式は、「文書内の任意の位置にある body 要素の配下に現れる placeName 要素」を意味する。ここでは // が二度用いられているが、それぞれに役割がある。最初の //body は、「文書中のどこかに存在する body 要素」を指定している。一方、後半の //placeName は、「その body 要素の内部で、途中にどのような要素が挟まっていてもよいので placeName 要素を探す」ことを意味する。



このような書き方は、<body> の直下に <placeName> が置かれているとは限らず、 や <p> など、さまざまな要素が間に入る可能性がある場合に有効である。文書構造の細部をすべて把握していなくても、安全に検索対象を指定できる点に、// を用いた XPath の利便性がある。

この検索を実行すると、本文中に現れる地名のみが抽出され、ここではおよそ 13 件がヒットするはずである。これにより、先ほどの検索結果から、本文以外に記述された地名が除外されたことが確認できる。

検索結果の一覧では、任意の行をクリックすると、上部の編集画面において該当箇所が自動的にスクロールされ、ハイライト表示される。これにより、単に件数を数えるだけでなく、実際にどのような文脈でその地名が用いられているかを、容易に確認できる。このように、検索結果と

本文表示を往復しながら確認できる点も、Oxygen を用いた検索の利点である。

2-7．XPath で属性を絞り込む

XPath では、要素そのものだけでなく、要素に付与された属性を対象として検索することもできる。TEI/XML 文書では、人物名や地名などが要素として記述されるだけでなく、識別子や参照先が属性として与えられることが多い。このような属性情報を用いることで、より構造的な検索が可能となる。

たとえば、本書簡では人物参照として <rs> 要素が用いられ、その人物識別子が @corresp 属性として記述されている。この人物 ID を一覧として抽出したい場合には、XPath を次のように記述する。

```
//body//rs/@corresp
```

この XPath 式では、//body//rs によって本文中の <rs> 要素を指定し、その後の @corresp によって、その要素が持つ corresp 属性を検索対象としている。すなわち、「本文中で人物参照として用いられている識別子」を直接取り出す指定の仕方である。

このファイルでは <rs> 要素の数がそれほど多くないため、検索結果としての効果は限定的に

見えるかもしれない。しかし、人物参照が多数含まれる文書や、複数文書を横断して検索する場合には、このような属性検索が大きな威力を発揮する。XPath を用いることで、「どの人物が、どれくらいの頻度で参照されているか」といった分析の前段階となるデータを、容易に取得できるようになる。

2-8．XPath への理解を深めるには

XPath は、XML の関連技術として広く用いられており、Oxygen XML Editor に限らず、多くの処理環境やプログラミング言語で利用されている。次節で扱う XSLT においても、XPath は、変換対象を指定するための基盤として不可欠である。

さらに、XPath を拡張した問い合わせ言語として XQuery も存在し、XML データベースなどではより高度な検索や集計が行われている。XPath の基本的な考え方を理解しておくことは、単なる検索操作にとどまらず、XML 文書を研究データとして扱うための重要な基礎となる。より詳しい用法については、専門書や解説資料を参照されたい。

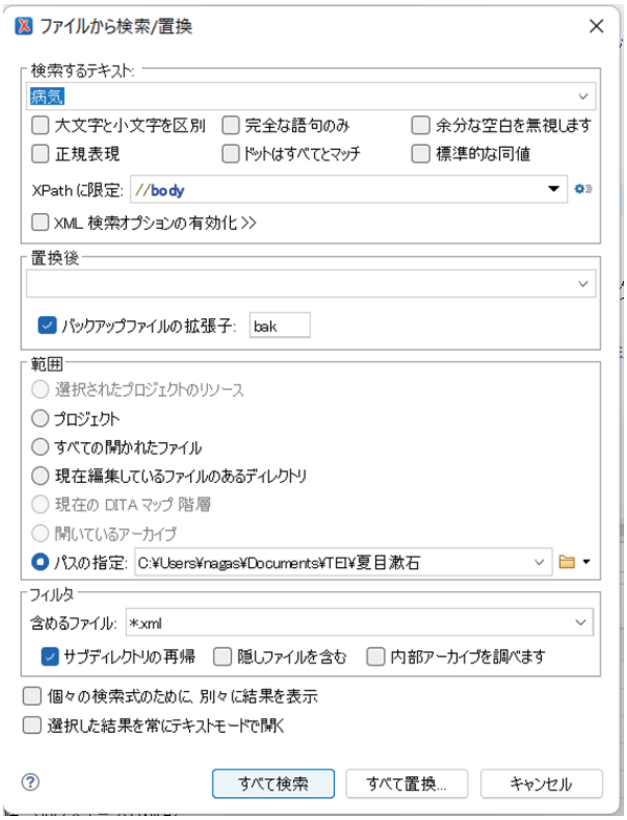
2-9．複数ファイルの同時検索

TEI/XML によるテキスト構造化の効果は、単一の文書を扱う段階よりも、同じ規則で記述された複数の文書をまとめて扱う段階で、より明確になる。Oxygen XML Editor には、同一フォルダ内にある複数の XML ファイルや、ファイル名をワイルドカードで指定したファイル群を対象として、一括で検索や置換を行う機能が備わっている。

この複数ファイル検索では、これまで見てきた XML 構造を考慮した検索や XPath 検索も、そのまま利用できる。したがって、同一の TEI スキーマやタグ付け規則に基づいて作成された文書が増えてくるほど、この機能の有効性は高まる。個々の文書を一つずつ開いて確認するのではなく、文書群全体を一つのデータ集合として扱える点に、この機能の意義がある。

複数ファイルを同時に検索するには、メニューバーの「検索」から「ファイルから検索／置換」を選択する。すると、複数ファイル検索用のダイアログが表示される。「検索するテキスト」の設定は、これまで使用してきた検索ダイアログと基本的に同じであり、文字列検索、XML 検索オプション、XPath 検索などを同様に指定できる。

検索対象となるファイル群は、「範囲」の設定によって指定する。ここで「パスの指定」を選択し、検索したいファイルが保存されているフォルダを指定すると、そのフォルダ内のファイルがまとめて検索対象となる。さらに、検索対象を限定したい場合には、「フィルタ」の設定を用いる。「含めるファイル」の欄にワイルドカードを用いてファイル名を指定することで、特定の種類のファイルだけを検索対象とすることができる。たとえば、拡張子が .xml のファイルのみを対象としたい場合には、「*.xml」と入力すればよい。



検索を実行すると、結果はファイルごとに整理され、各ファイル内でのヒット数が一覧表示される。これにより、どの文書にどの程度該当箇所が含まれているかを、俯瞰的に把握できる。また、検索結果ウインドウ上で右クリックを行うことで、検索結果をファイルとして保存することも可能である。この機能を用いれば、検索結果を外部で整理・集計したり、分析の基礎データとして利用したりすることもできる。

このように、複数ファイル検索は、TEI/XML 文書を「個別のテキスト」ではなく、「構造化された文書群」として扱うための重要な手段である。検索対象が増えるほど、構造化された記述の利点が実感される点を、ここで確認しておきたい。

2-10．XSLT による処理

XML 文書を処理・整形するための代表的な手法の一つに、XSLT（XSL Transformations）がある。XSLT は、XML の関連技術として定義された XSL（Extensible Stylesheet Language）の一部であり、XML 文書の構造を読み取り、別の XML 文書や HTML、あるいはテキスト形式へと変換するための言語である。単に表示を整えるための仕組みではなく、XML 文書の構造そのも

のを変換対象とする点に特徴がある。

これまでに見てきた検索や XPath は、既存の XML 文書から特定の情報を見つけ出すための手段であった。それに対して XSLT は、XML 文書全体を入力として受け取り、新たな出力文書を生成するための処理系である。すなわち、XSLT を用いることで、構造化された XML データを、閲覧・分析・再利用に適した別の形式へと体系的に再構成することが可能となる。

XML 文書の処理は、Python や Java、Ruby といった汎用的なプログラミング言語を用いて行うこともできる。実際、これらの言語には XML を読み込むためのライブラリが整備されており、複雑な処理や高度な分析を行うには有力な選択肢となる。とりわけ Python は、XML 処理用ライブラリに加えて、自然言語処理や機械学習、深層学習のためのライブラリが充実しており、近年の人工知能技術と接続しやすいという点で大きな利点を持つ。そのため、TEI/XML データを分析や自動処理、生成 AI と連携させた応用へと展開することを考える場合、実践的には Python を用いるのが基本的に推奨される選択肢である。

一方で、このような汎用プログラミング言語による処理では、XML 文書の構造をプログラム内部で逐次的に解釈し、処理の流れを手続き的に記述する必要がある。その結果、処理内容がプログラムのロジックに強く依存し、XML の構造そのものがどのように変換・利用されているのかが、外部からは見えにくくなる場合も少なくない。

これに対して XSLT は、XML を XML として扱うことを前提に設計された宣言的な言語である。どの要素を対象とし、どのような構造に変換するかを、テンプレートとして明示的に記述するため、処理の対象と変換結果の対応関係が比較的読み取りやすい。すなわち、XSLT のスタイルシートそのものが、「この XML 文書をどのように理解し、どのような構造として再構成するか」を記述した文書になっている。

また、XSLT は XML、XPath と密接に結びついた標準技術であり、特定の実装やプログラミング言語に依存しない。そのため、処理環境が変わっても再利用しやすく、長期的な運用や共有に適している点も重要である。この点は、研究成果の持続可能性を重視する人文学研究において、特に大きな意味を持つ。

XSLT は確立された文法と処理モデルを持つ言語であり、本来は基礎から体系的に学ぶべき対象である。そのため、欧米の Digital Humanities 教育においては、XML 処理の標準的手法として基礎科目の中で教えられることが多い。しかし、本節の目的は、XSLT を網羅的に習得することではなく、構造化された XML 文書が、どのような原理で処理・変換されうるのかを具体的に理解することにある。

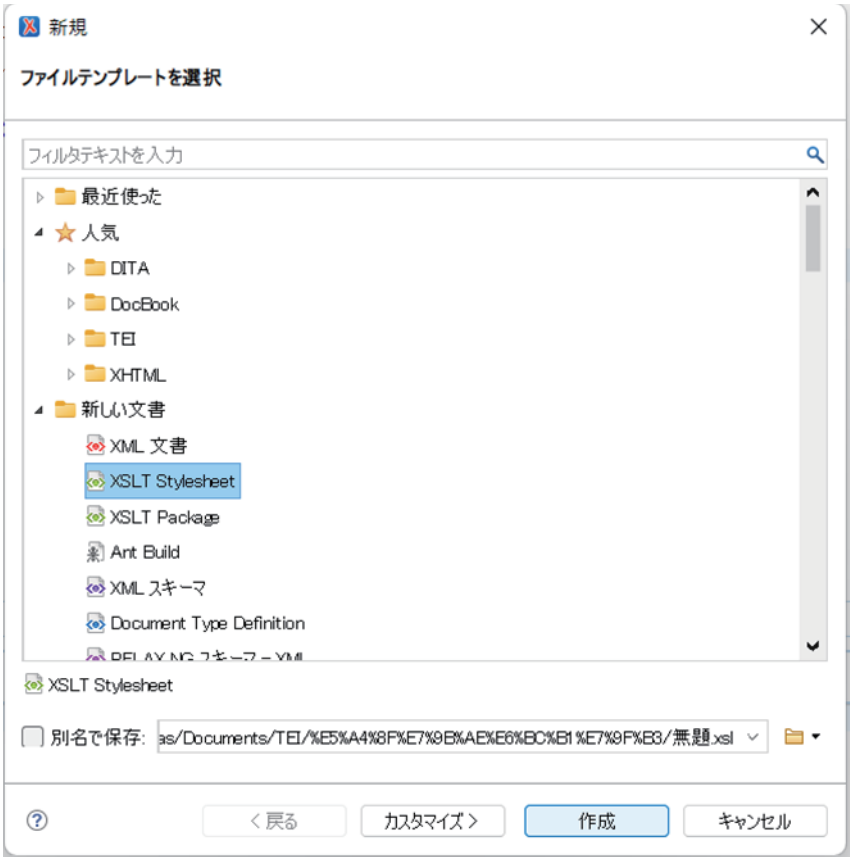
Oxygen XML Editor には、XSLT を実行するための機能が標準で備わっている。これにより、特別な実行環境を構築することなく、XML 文書と XSLT スタイルシートを組み合わせた変換処理を容易に試すことができる。そこで本節では、漱石書簡の TEI/XML 文書を例として、XSLT を用いた基本的な整形・変換を実際に行いながら、その考え方と可能性の一端を確認する。

2-11．XSLT による変換の準備と実行

① XSLT 文書を新規作成する

まず、XSLT スタイルシートを新規に作成する。Oxygen XML Editor で「新規作成」を選択すると、作成する文書の種類を選ぶダイアログが表示される。この中から「XSLT Stylesheet」を選び、「作成」をクリックする。

ここで作成されるファイルは、拡張子が .xsl の XML 文書であり、XSLT もまた XML に準拠した記述形式であることが分かる。XSLT は、プログラムでありながら XML 文書として記述される点に特徴があり、XML を XML で処理するという発想がここに現れている。



② TEI の名前空間を記述して保存する

新規作成された XSLT ファイルには、最小限のテンプレートがあらかじめ記述されている。この段階で、XSLT ファイル自体が XML 文書として正しく構成されていることを確認しておくとよい。

次に、TEI/XML 文書を処理対象とするため、TEI の名前空間に関する情報を XSLT 側に明示

的に記述する。具体的には、図に示したとおり、5 行目に TEI の名前空間宣言を追加し、続いて 6 ～ 9 行目に必要な設定を記述する。これらのタグは、Oxygen の入力支援機能を利用することで、比較的容易に入力できる。

ここで記述した TEI の名前空間宣言は、「この XSLT 文書の中では、TEI に属する要素を tei: という接頭辞付きで扱う」という約束を定めるものである。TEI/XML 文書では名前空間が用いられているため、この宣言を行わないと、XSLT 側から TEI の要素を正しく参照できない。

6 行目では、出力形式として HTML を指定し、文字コードを UTF-8、HTML5 に準拠した形で出力することを指示している。7 行目では、テンプレートの対象として XML 文書のルート（/）を指定し、8 行目で TEI 名前空間を持つ <TEI> 要素の内容を処理対象として取り出している。9 行目は、このテンプレートを閉じる終了タグである。

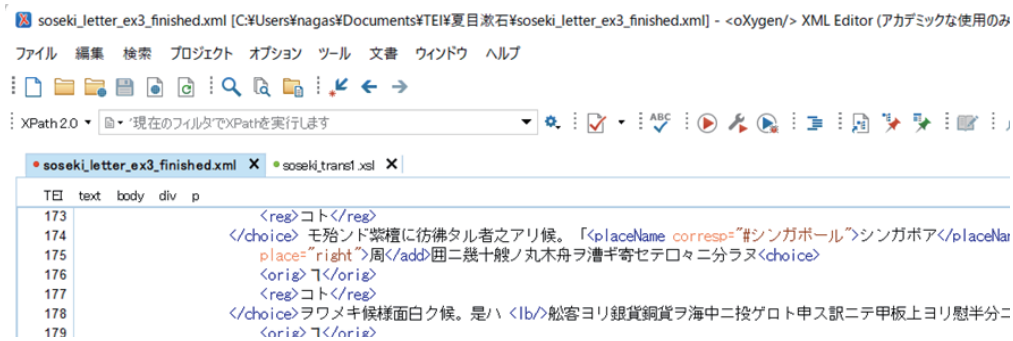
これらの設定を行ったら、ファイルを保存する。ファイル名は、たとえば soseki_trans.xsl のように、処理内容が分かる名称を付けるとよい。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3      xmlns:xs="http://www.w3.org/2001/XMLSchema"
4      exclude-result-prefixes="xs"
5      version="2.0" xmlns:tei="http://www.tei-c.org/ns/1.0">
6      <xsl:output method="html" encoding="UTF-8" version="5"/>
7      <xsl:template match="/">
8          <xsl:apply-templates select="tei:TEI"/>
9      </xsl:template>
10 </xsl:stylesheet>
```

③ XML 文書に XSL ファイルを関連付ける

XSLT による変換を効率的に行うためには、XML 文書と XSLT ファイルを関連付けておくと便利である。これにより、変換結果を確認しながら、XSLT の修正を繰り返すことができる。

先ほど XSLT ファイルを保存した後、夏目漱石書簡の TEI/XML 文書を開き、「変換シナリオの設定」ボタンをクリックする。



④「変換シナリオの設定」

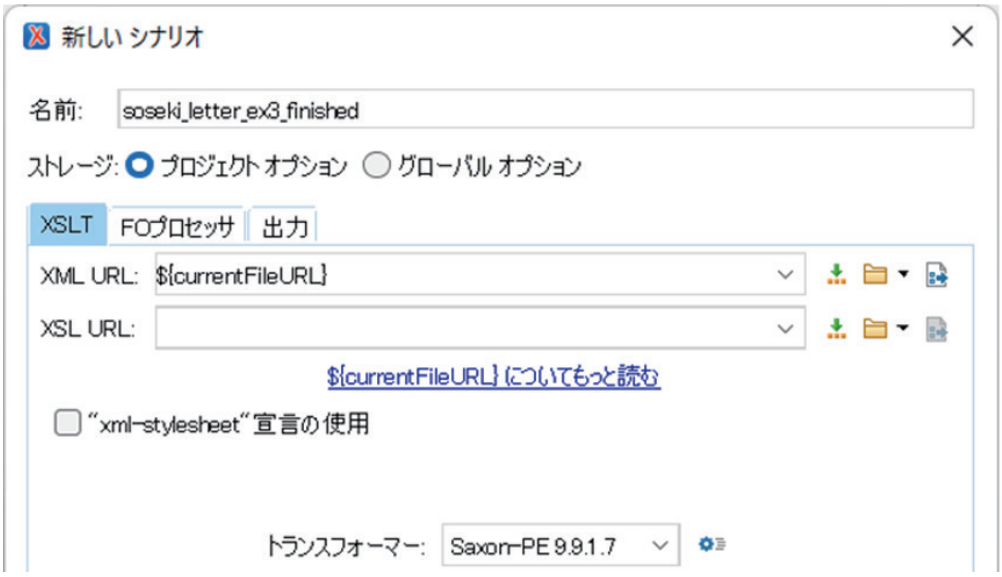
「変換シナリオの設定」ダイアログが開いたら、「新規」ボタンをクリックし、表示される選択肢の中から「XML Transformation with XSLT」を選択する。

ここで設定する「変換シナリオ」とは、「どの XML 文書を、どの XSLT ファイルを用いて、どのような形式で出力するか」をまとめて定義したものである。これを一度設定しておくことで、以後は同じ条件で何度でも変換を実行できる。

⑤「新しいシナリオ」の設定（XSLT）

「新しいシナリオ」の設定画面では、XSLT を用いた変換の詳細を指定する。XML 文書から設定を開始しているため、変換対象となる XML 文書は「XML URL」の欄にすでに指定されている。

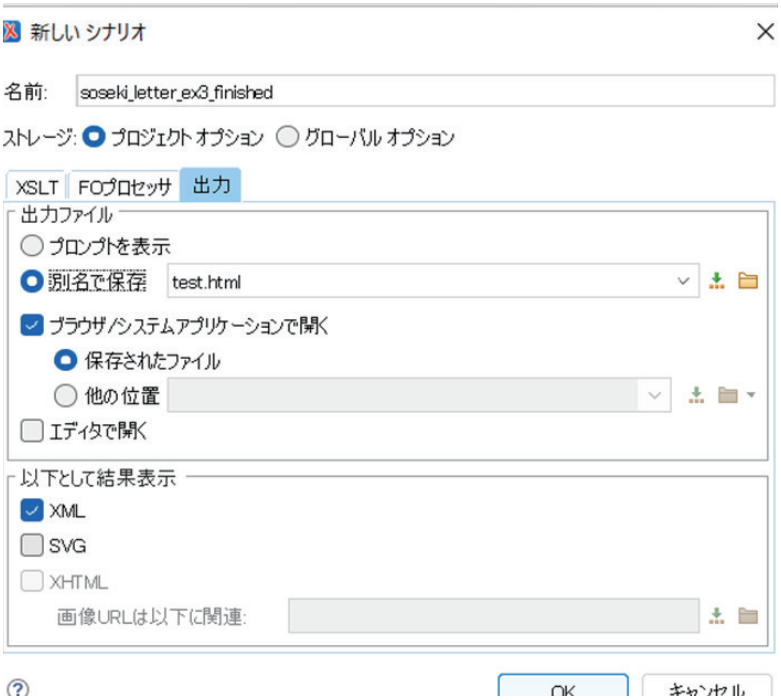
ここでは、「XSL URL」の欄に、先ほど作成・保存した XSLT ファイルを指定する。その後、「出力」タブをクリックして、出力形式に関する設定を行う。



⑥「新しいシナリオ」の設定（出力）

出力設定の画面では、まず「別名で保存」の欄に、出力される HTML ファイルの名前を入力する。次に、「ブラウザ／システムアプリケーションで開く」にチェックを入れる。これにより、変換処理の実行後、自動的に Web ブラウザが起動し、結果を確認できるようになる。

設定が完了したら、「OK」をクリックして変換シナリオを保存する。

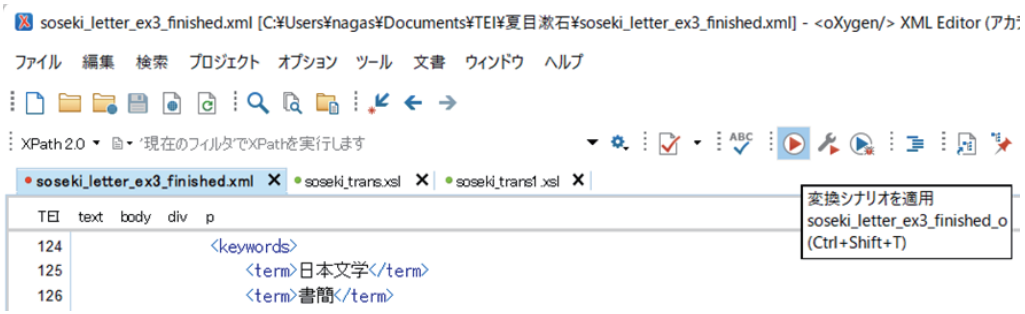


⑦変換シナリオを適用する

設定を終えたら、変換対象となる漱石書簡の XML 文書の画面に戻り、「変換シナリオを適用」ボタンをクリックする。

すると Web ブラウザが起動し、TEI/XML 文書からテキスト部分のみが抽出された HTML が表示されるはずである。この表示が確認できれば、XML 文書と XSLT ファイルの関連付けは正しく行われている。

以後、XSLT ファイルや XML 文書を修正した場合でも、この「変換シナリオを適用」を実行することで、修正内容を反映した変換結果をすぐに確認できる。



⑧ 本文 <body> の内容を表示させる

次に、XSLT ファイル（soseki_trans.xsl）に戻り、図に示した 10 ～ 18 行目の内容を追記して保存する。その後、再び漱石書簡の XML 文書に戻り、「変換シナリオを適用」を実行する。

```
7  <xsl:template match="/">
8      <xsl:apply-templates select="tei:TEI"/>
9  </xsl:template>
10 <xsl:template match="tei:teiHeader"> </xsl:template>
11 <xsl:template match="tei:text">
12     <html>
13         <head/>
14         <body style="padding:50px">
15             <xsl:apply-templates select="tei:body"/>
16         </body>
17     </html>
18 </xsl:template>
```

この設定では、7 ～ 9 行目で <TEI> 要素全体を処理対象とした後、10 行目で <teiHeader> を指定しつつ出力を行わず、11 行目から <text> の処理を開始している。そして、その下位にある <body> 要素のみを 15 行目で出力している。

12 ～ 14 行目および 16 ～ 17 行目には HTML のタグが記述されており、これらはそのまま出力されて HTML 文書として機能する。14 行目の <body> タグには @style 属性が付与されており、ここでは表示領域の周囲に 50px の余白を設ける指定を行っている。

このように、XSLT を用いることで、TEI/XML 文書の中から必要な構造だけを選び出し、表示や利用に適した形へと再構成できることが確認できる。

⑨ 本文の段落を表示する

前段までの設定では、本文の内容は出力されるものの、段落の区別が行われておらず、連続し

た文字列として表示されている。そのため、可読性が著しく低く、実際の書簡本文として読むことが困難である。ここでは、TEI/XML 文書において段落として記述されている構造を利用し、段落ごとに区切って表示する処理を追加する。

今回用いている XML 文書では、本文中の <body> の内部に があり、その中に複数の <p> 要素が記述されている。このような「同種の要素が繰り返し現れる構造」を処理するために、XSLT では <xsl:for-each> 要素を用いる。select 属性に XPath を指定することで、その XPath に一致する要素が存在する限り、同じ処理を繰り返し適用することができる。

```
19  <xsl:template match="tei:div">
20      <xsl:for-each select="tei:p">
21          <p>
22              <xsl:apply-templates select="node()"/>
23          </p>
24      </xsl:for-each>
25  </xsl:template>
```

そこで、先ほど作成したテンプレートの後に、図に示したスクリプトを追記する。この処理では、まず 要素を処理対象とし、その内部に含まれる複数の <p> 要素を順に取り出して処理している。各 <p> 要素については、HTML の <p> タグで囲んで出力することで、Web ブラウザ上でも段落として表示されるようになる。

このスクリプトでは、新たなテンプレートが定義されている。19 行目では 要素を処理することを指定し、20 行目ではその内部の <p> 要素を繰り返し処理することを宣言している。21 行目と 23 行目では、表示用の HTML <p> タグを開始・終了として出力し、22 行目で元の <p> 要素の内容を処理・表示している。これにより、TEI/XML 文書における段落構造が、HTML 上でも段落として反映される。

⑩ <choice> における選択を行う

次に、TEI/XML 文書における校訂情報の扱いを考える。現時点の XSLT では、<choice> 要素に含まれる <sic> と <corr>、あるいは <orig> と <reg> が、そのまま並列に出力されてしまっている。この状態では、表示としても検索対象としても扱いにくい。

<choice> は、複数の表記のうち、どれを採用するかという判断を明示的に記述するための要素である。表示や検索の目的に応じて、どの表記を採用するかを決める必要がある。ここでは、まず「原文に忠実な表記」を採用するという方針をとり、<orig> または <sic> が存在する場合には、それを表示する処理を追加する。

図に示したテンプレートを、先ほどのスクリプトに続けて追記し、保存したうえで再度変換を実行する。このテンプレートでは、<choice> 要素の中に <orig> または <sic> が存在すれば、

それを選択して表示するよう指示している。

```
26    <xsl:template match="tei:choice">
27        <xsl:apply-templates select="tei:orig"/>
28        <xsl:apply-templates select="tei:sic"/>
29    </xsl:template>
```

その後、同じテンプレート中の tei:orig および tei:sic を、それぞれ tei:reg および tei:corr に書き換えて保存し、再度変換を実行してみる。すると、今度は正規化された表記や訂正文が表示されるはずである。この違いを確認することで、XSLT が「記述された構造の中から、どの解釈を採用するか」を明示的に選択する仕組みであることが理解できる。

⑪ <subst> の表示を行う

<subst> 要素は、抹消と追記の関係、すなわちミセケチと追記を表現するために用いられる。構造としては <choice> に近く、削除された文字列（<del>）と、新たに書き加えられた文字列（<add>）を組み合わせて記述する。

ここでも、表示の方針を明示的に決める必要がある。本演習では、削除された文字列には取り消し線を引き、追記された文字列は強調表示するという方法をとる。具体的には、<del> に含まれる文字列を HTML の <s> タグで囲み、<add> に含まれる文字列を太字で表示する。

```
30    <xsl:template match="tei:subst">
31        <s>
32            <xsl:apply-templates select="tei:del"/>
33        </s>
34        <b>
35            <xsl:apply-templates select="tei:add"/>
36        </b>
37    </xsl:template>
```

この処理を記述したテンプレートを追記して保存し、変換を実行すると、抹消と追記の関係が視覚的に分かる形で表示される。このように、TEI/XML で記述された書字行為の痕跡を、XSLT によって適切に再構成できることが確認できる。

⑫ <add> 要素の表示を調整する

漱石書簡の XML 文書には、<subst> に含まれない <add> 要素も存在する。これらは単純な追記であり、文中に直接挿入された場合もあれば、欄外や行右に書き加えられたものもある。

ここでは、すべての <add> 要素を太字で表示することに加え、@place 属性に right を持つも

のについては、表示位置を少しずらす処理を行う。これにより、行右への追記であることを視覚的に表現できる。

```
38    <xsl:template match="tei:add">
39        <xsl:choose>
40            <xsl:when test="@place = 'right'">
41                <span style="font-weight:bold;vertical-align: super;">
42                    <xsl:apply-templates/>
43                </span>
44            </xsl:when>
45            <xsl:otherwise>
46                <span style="font-weight:bold;">
47                    <xsl:apply-templates/>
48                </span>
49            </xsl:otherwise>
50        </xsl:choose>
51    </xsl:template>
```

この処理では、条件分岐を行うために <xsl:choose> 要素を用いる。39 行目で条件分岐を開始し、40 行目の <xsl:when> において、<add> 要素の @place 属性が right である場合の処理を指定する。この場合、HTML の タグにスタイル属性を付与し、太字かつ上付き表示として出力している。42 行目で実際の文字列を出力し、45 行目では @place 属性が right でない場合、すなわち通常の追記として扱う場合の処理を記述している。

このように、XSLT では属性値に基づいて処理を分岐させることで、TEI/XML に記述された物理的・書字的情報を、表示上の差異として反映させることができる。

⑬ <closer> の表示を調整する

現時点では、<closer> 要素に対する特別な処理を行っていないため、書簡の結語や署名部分は本文と同じ体裁で表示されている。しかし、書簡という文書ジャンルを考えると、結語（<salute>）や署名は、本文とは異なる配置や視覚的強調を持つのが自然である。

```
52    <xsl:template match="tei:closer">
53        <div style="text-indent:3em">
54            <xsl:apply-templates select="tei:salute"/>
55        </div>
56        <div style="text-align:right">
57            <xsl:apply-templates select="tei:signed"/>
58        </div>
59    </xsl:template>
```

そこでここでは、<closer> 内の構造を利用し、結語部分はやや行を下げて表示し、署名部分は右寄せで表示する処理を追加する。これにより、TEI/XML 文書に記述された書簡特有の構造が、表示上でも分かりやすく反映される。

具体的には、<closer> やその下位要素を処理するテンプレートを追加し、HTML の 要素に @style 属性を付与して CSS による表示指定を行う。ここで重要なのは、TEI/XML 側では文書の論理構造や役割分担を記述し、実際の見た目の調整は CSS に委ねている点である。このように、構造の記述と表示の指定を分離することが、TEI と XSLT を用いた処理の基本的な考え方である。

⑭ <lb> を改行として表示する

漱石書簡の TEI/XML 文書では、行の切れ目を示すために <lb> 要素が用いられている。<lb> は、段落や文の意味構造とは異なり、原資料における行分割という物理的構造を記述するための要素である。

Web ブラウザ上で原資料の見た目に近い形を再現するためには、この <lb> を HTML の改行要素である
 に対応付けるのが一つの方法である。そこで、<lb> 要素を検出した際に
 を出力するテンプレートを追加する。

```
60    <xsl:template match="tei:lb">
61        <br/>
62    </xsl:template>
```

この処理は、これまでに行ってきた要素変換と同様の書き方で記述できる。TEI/XML における物理的構造が、表示上の改行として反映されることで、原資料の行配置を意識した閲覧が可能となる。

⑮ 縦書き表示への対応

日本語資料では、縦書きか横書きかという書字方向の違いが重要な意味を持つ場合がある。XSLT 側で単純に縦書き用の CSS を指定することも可能であるが、その場合、縦書きであるかどうかの判断が処理側に固定されてしまう。

より望ましい方法は、縦書きか横書きかという情報を XML 文書側に明示的に記述し、XSLT はその記述を参照して表示方法を切り替える、という役割分担をとることである。この考え方は、表示の判断をデータに委ね、処理系はそれを解釈する、という TEI 的な発想に基づいている。

```
11    <xsl:template match="tei:text">
12        <html>
13            <head/>
14            <xsl:element name="body">
15                <xsl:attribute name="style">
16                    <xsl:value-of select="tei:body/@style" />
17                    ;padding:50px
18                </xsl:attribute>
19                <xsl:apply-templates select="tei:body"/>
20            </xsl:element>
21        </html>
22    </xsl:template>
```

ここでは、その一例として、<body> 要素の生成方法を変更し、<xsl:element> を用いて HTML の <body> 要素を動的に作成する。15 ～ 18 行目では、変換対象となる XML 文書に記述されたスタイル情報を参照し、それに応じて縦書きまたは横書きの CSS を適用している。また、17 行目では、可読性を高めるための追加的なスタイル指定を行っている。

このように、XSLT を用いることで、XML 文書に含まれる情報をもとに表示形式を柔軟に切り替えることが可能となる。

⑯ 余計な空白を削除する ― 適切な検索のために

これまでの処理によって生成された HTML では、<choice> 要素の境界などに半角スペースが挿入されてしまう場合がある。このような空白は、見た目上は問題にならなくても、文字列検索を行う際には支障となる。たとえば、<choice> 内の語とその前後の文字列を連続した形で検索できない、といった問題が生じる。

```
68    <xsl:template match="text()">
69        <xsl:value-of select="normalize-space(.)"/>
70    </xsl:template>
```

そこで、当面の対処として、出力された本文中の文字列から不要な半角スペースを削除する処理を追加する。図に示したスクリプトを追記して保存し、再度変換を実行すると、余分な空白が除去された状態で HTML が出力される。

ここで重要なのは、この処理が「表示のため」というよりも、「検索や再利用を前提とした整形」である点である。XSLT による変換は、単に見た目を整えるためのものではなく、後続の検索・分析・再利用を見据えた前処理としても機能する。この点を意識することで、XSLT 処理の位置

づけがより明確になる。

⑰ 人名リストを取り出してみる

ここまでの演習では、主として TEI/XML 文書全体を対象とし、閲覧や可読性を高めるための整形処理を行ってきた。しかし、XSLT の有用性は、表示のための変換にとどまらない。タグ付けされた構造を利用して、文書中から特定の情報だけを抽出する場合にも、XSLT は有効な手段となる。

ここでは一例として、本文中に現れる人名と、その人物に対応付けられた識別子（@corresp 属性）を抽出し、カンマ区切りのリストとして出力する処理を行う。このような処理は、人物索引の作成や、後続の分析・可視化のための前処理として位置づけることができる。

```
4     xmlns:tei="http://www.tei-c.org/ns/1.0">
5     <xsl:output method="text" encoding="UTF-8"/>
6     <xsl:template match="/">
7         <xsl:apply-templates select="tei:TEI"/></xsl:template>
8     <xsl:template match="tei:teiHeader"></xsl:template>
9     <xsl:template match="tei:body">
10            <xsl:for-each select=".//tei:persName">
11  <xsl:apply-templates select="."/>,<xsl:apply-templates select="@corresp"/><xsl:text>&#xA;</xsl:text>
12            </xsl:for-each>
13    </xsl:template>
14    <xsl:template match="tei:back"></xsl:template>
15    <xsl:template match="text()">
16        <xsl:value-of select="normalize-space(.)"/>
17    </xsl:template>
18  </xsl:stylesheet>
```

今回は、新たに XSLT ファイルを作成し、図に示したスクリプトを記述して適用する。ここでの処理は、これまでの XSLT と比べて出力形式が異なっている点に注意が必要である。5 行目の method 属性では、HTML ではなくテキスト形式で出力することを指定している。これにより、Web ブラウザでの表示ではなく、機械処理やデータ分析に適したプレーンテキストが生成される。

また、今回は書簡本文のみを対象とするため、8 行目および 14 行目において <body> 要素以外からは出力が行われないように制御している。11 行目では、人名要素とその @corresp 属性の値を取り出し、それらをカンマで区切って出力している。さらに、改行を明示的に挿入するために <xsl:text>
</xsl:text> を用いている。

15 〜 17 行目では、これまでと同様に、不要な空白を削除する処理を行っている。このような整形は、後続の検索や集計、プログラムによる読み込みを想定した場合に重要となる。

同じ考え方を用いれば、<rs> 要素全体や <placeName> 要素を対象としたリストの作成、あるいは日付情報や出来事の抽出なども容易に行える。XSLT による抽出処理は、TEI/XML 文書に埋め込まれた意味構造を、明示的なデータとして取り出すための第一歩である。

⑱ さらなる挑戦

Oxygen XML Editor の「変換シナリオの設定」では、複数の XSLT ファイルを登録し、同一の XML 文書に対して複数の変換を同時に実行することができる。この機能を利用すれば、同じ TEI/XML 文書から、目的の異なる複数の出力を並行して生成することが可能である。

そこで、ここまでの演習を踏まえた応用課題として、「書かれたままのテキスト」と「検索や分析を行いやすいように正規化されたテキスト」を、それぞれ別の出力として作成することを考えてみてほしい。このとき重要なのは、「書かれたまま」とは何を指すのか、「検索しやすく正規化する」とはどのような処理を意味するのかを、あらかじめ明確に定義することである。

たとえば、校訂前の表記を保持するのか、正規化された表記を採用するのか、改行や空白をどこまで残すのか、といった判断は、すべて研究目的に依存する。XSLT は、そのような判断を処理として明示的に記述するための道具である。この課題を通じて、構造化テキストの利活用が、単なる技術操作ではなく、研究上の選択の集合であることを確認してほしい。

2-12．XSLT のまとめ

本節では、XSLT の理論的背景に深入りすることはせず、実際に動作する例を通じて、XML 文書がどのように処理・変換されうるのかを一通り確認してきた。これにより、XSLT を用いることで何が可能であり、どのような場面で有効であるのかについて、概観的な理解が得られたはずである。

ここで試した例は、いずれも基本的なものであり、個々のスクリプトを調整するだけでも、さまざまな用途に応用できる。一方で、XSLT は体系化された言語であり、より複雑な処理や効率的な記述を行うためには、専門のガイドブックや解説資料を参照しながら、段階的に理解を深めていく必要がある。

重要なのは、XSLT を「すべてを XSLT で行うための技術」と捉えるのではなく、TEI/XML によって記述された構造を活かし、表示・抽出・前処理といった役割を担う技術として位置づけることである。この位置づけを理解することで、XSLT と Python 等の汎用プログラミング言語、さらには生成 AI を用いた分析や処理との役割分担も、より明確になる。

3．多様な処理や再利用を前提とした研究基盤として

3-1．既存のツールによる活用

TEI ガイドラインは、人文学の多様な分野や研究観点を反映して策定されてきた。そのため、TEI ガイドラインに準拠して作成されたデータを対象とするツールも、用途や目的に応じて多様なものが存在する。研究者自身が、自らの研究目的に合わせて表示・処理ツールを開発すること

も少なくないが、近年では、それらのツールのソースコードがオープンソースとして公開される例も増えている。その結果、既存のツールやコードを基に、自身のデータに適した形へと調整・拡張することが可能になっている。

TEI に準拠してデータを記述する意義の一つは、このような既存ツールを比較的容易に再利用できる点にある。個別のプロジェクトごとに専用形式を定義した場合、そのプロジェクト専用のツールを新たに開発する必要が生じる。一方、TEI ガイドラインに従って記述されたデータであれば、既存のツールに読み込ませるだけで、一定程度の表示や分析が可能となる場合がある。

既存の TEI 対応ツールについては、すでに見てきたように、TEI 古典籍ビューワが縦書き文献に関しては様々な構造に対応できるようになっているので試用してみていただきたい。

また、今回扱っている漱石書簡の TEI/XML 文書を利用しやすい具体例としては、書簡の送受信関係を可視化するツールが公開されているので、これを簡単に試してみよう。

TEI 協会東アジア／日本語分科会が公開している GitHub 上の青空文庫 TEI プロジェクトのページにアクセスすると、ページ下部に「視覚化ツール」というコーナーがある。その中の「`<correspDesc>` を用いて書簡の送受信を可視化する」を選択すると、「書簡の送受信の可視化」というページが開く。このページ最下部にある「TEI/XML ファイルを選択して表示する」というフォームから、漱石書簡の XML 文書を指定すると、地図や年表などの形で書簡の送受信関係が表示されるようになっている。このツールでは、さらに、複数の書簡ファイルでも読み込んで表示できるようになっている。

この例が示しているのは、TEI/XML 文書に記述された構造的情報が、個別に追加の処理を行わなくても、既存のツールによって再解釈・再利用されうるという点である。TEI ガイドラインに準拠した記述は、単なる保存形式ではなく、分析や可視化を支える共通基盤として機能する。

3-2．保存と共有

このようにして作成した TEI/XML 文書は、ファイルとして保存し、GitHub や GitLab といったコード共有プラットフォーム、あるいは Zenodo などの研究データ共有サービスを通じて公開・共有することができる。また、TEI 協会が運営に関わっている TEI ファイル専用の共有プラットフォームである TAPAS に登録し、公開することも可能である。

TEI/XML 文書が他のファイルを参照している場合には、参照先のファイルも含めて一式として公開することが望ましい。研究データの公開と共有は、近年ますます重要性を増しており、その際には、データの内容や作成経緯、利用条件などをメタデータとして明示する必要がある。TEI/XML 文書に含まれる `<teiHeader>` は、このようなメタデータ記述のための枠組みとしても有用であり、公開・共有の段階で大きな役割を果たす。

3-3．おわりに

本章では、TEI ガイドラインに準拠したデータの利活用について、検索、変換、抽出、可視化、

共有といった一連の流れを、実習を通じて概観してきた。TEI/XML で作成された文書は、単なる保存用データではなく、多様な処理や再利用を前提とした研究基盤である。

XSLT による変換に限らず、Python、Java、Ruby、R、PHP など、多くのプログラミング言語は XML を扱うための機能を備えており、TEI/XML データを起点としてさまざまな応用が可能である。特に Python は、自然言語処理や機械学習、生成 AI といった技術と結びつけやすく、今後の人文学研究において重要な役割を果たすことが期待される [1)]。

注

1　TEI 協会東アジア／日本語分科会の GitHub サイトでは、こうした利活用の具体例やツールが継続的に紹介されている。ここで学んだ内容を出発点として、TEI/XML による構造化データを、各自の研究関心や方法論に応じて発展させていってほしい。