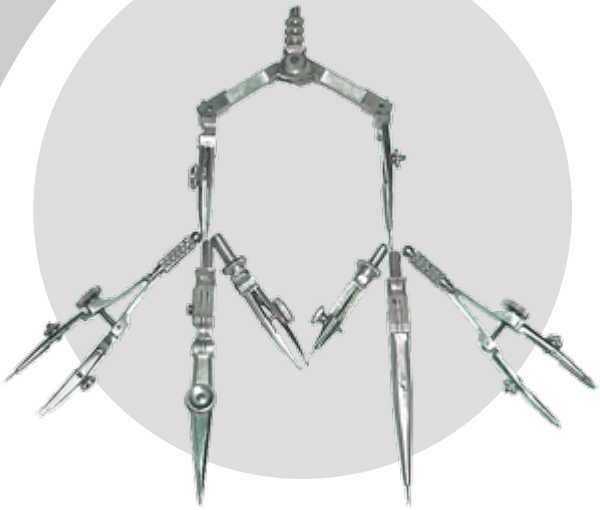
XML
設計の心得

ヒトが木構造を設計する意味とは？
実務者が伝える，実務者の心得。

PROJECT KySS 著



ダウンロード日時：（　　年　　月　　日・　　時　　分）
学校・勤務先：（　　）
氏名：（　　）
備考：

SDESIGN
セイザインデザイン

刊行にあたって

本書は，PROJECT KySS 著のオンライン書籍の試験公開第 2 弾として公開したものです。

第 1 弾は，2003 年 2 月にテスト公開した，サンプル・プログラム付き PDF ファイル「プログラムパーツ集，ASP.NET ネタ帳」です。

本書以降も，実験的な内容のもの，狭い専門分野に焦点をあてたために販売数は見込めないが読者にとっては必要なもの，また，ツールのバージョンアップや社会情勢などの絡みから公開を急ぐために，印刷媒体として作りこむ時間がないものにつきましては，印刷媒体との棲み分けをはかり，オンライン書籍として独自の公開を試みていく予定です。

注意事項

通常，印刷媒体として提供される書籍では，読者サポートも筆者業務のひとつです。そのため，筆者は，質問の可能性を考慮して，書籍執筆時のサンプル稼働環境を，書籍の流通期間中ずっと維持しています。しかしながら，現在では，OS，ブラウザ，開発実行環境などがめまぐるしくバージョンアップしています。

つきましては，筆者は，本書執筆時の環境を維持するものではなく，また，本書が求めに応じて提供されるものでもないことから，読者サポートは 行っておりません。

ダウンロードページに記載しております通り，皆さまの環境での解凍・閲覧・印刷に関するお問い合わせおよび，本書の内容に関するご質問には対応できかねます。

ご理解たまわりますよう，よろしくお願い申し上げます。

出版元：セイザインデザイン　代表　薬師寺 聖

著作権等について

・本書に掲載しておりますサンプル・データはダミーです。

・本著作権者ならびに出版権者は，本書の内容が正確であることに最善の努力を払いましたが，本書の内容を参考にして使用者がコンピュータおよびその周辺機器を操作することによって発生したハードウェアおよびソフトウェアの故障・損傷，設計業務の結果につきまして，著作権者ならびに出版権者はその一切の責任を負いかねます。あらかじめご了承ください。

・本書に掲載させていただきました Web サイトには，一般的に著作権が設定されています。本書は，各社ならびに各制作者の著作権を尊重することを表明いたします。本書に掲載させていただきました Web サイトは，随時更新されており，本書ダウンロード時から URI や内容が変更されていることがあります。あらかじめご承知おきください。

・Microsoft, Visual Studio, Visual Basic, .NET, LINQ, Office, Excel, Access, InfoPath, Silverlight は，米国 Microsoft Corportion の米国およびその他の国における登録商標または商標です。

・Windows の正式名称は，Microsoft Windows Oparationg System です。

・その他，本書に記載されている会社名，システム名，製品名は，一般に各社の商標または登録商標です。

・なお本文中には，TM，〇R マークは明記しておりません。

・本書に掲載されている登録商標または商標は，本書を制作する目的でのみ記載しており，その商標権を侵害する意図のあるものではありません。

本書の目的

XMLをテーマとする書籍や記事には，既に存在するXML文書のデータ処理方法について解説したものが多い。XMLの構造設計については，「受注した案件の内容を熟慮し，最適のXMLインスタンスを設計する」といった一言で済まされることすらある。
筆者自身，これまで多数の書籍や記事を執筆してきたが，その多くは「XML文書ありき」の内容である。

設計について語られることが多くないのは，それが簡単で単純な作業だからではない。むしろその逆で，非常に奥深いがために，語ることが難しいのである。業種や分野によって異なり，企画内容によっても大いに異なる。普遍的なメソッドがあるわけではなく，ケースバイケースで対応するしかない。かといってマニュアルのような書き方では，筆者が提供する卑小なセオリーに，読者を縛り付けることになりかねない。

「XML設計」という一言の背景にある，木構造の世界観を浮かび上がらせ，設計者の心得を伝える――それが本書の目的である。
人生にたとえて言うならば，本書は，「いかに生きるべきか（思想）」「社会のしくみ（仕様）」「人生設計（操作手順）」「生活の知恵（サンプル・プログラム）」をテーマとする本ではない。「存在を問うには，何を手掛かりに考えていけばよいのか」をナビゲートするための本である。

「XMLの本質」について書いてみては？というアイデアを提供したのは，音楽友達の旧知の編集者である。なるほどそれは書くべきだと思ったものの，遅々として筆が進まなかった。
これが技術書ではなく小説であれば，ストーリーの背景に，筆者のメッセージが見え隠れするものがある。繰り返し読めば，自分の心の成長に応じて，異なるメッセージの浮かび上がるものがある。技術書でもそういった書き方ができないだろうか？つまり，技術情報そのものの伝達を目的とするのではなく，仕様や実装について述べながらも，その背景にメッセージを潜り込ませるということはできないだろうか？などという無謀なことを考えてしまったからである。

このひどく難しい試みの前に筆者は無力であり，開発実務の合間をぬって，企画に1年，執筆に1年を費やし，2008年の秋に，ようやく初稿を書きあげた。W3Cの仕様が年々増え続け，バージョン間の差異を調べるだけでも，かなりの時間が必要だったという事情もある。初稿完成後すみやかに世に問うべきとは考えたが，さらに1年を要することとなってしまった。

この本は，読者の手をとって道を示すことはしない。大雑把な地図だけ渡して思考と応用を要求する。
なぜ，あえてそういう書き方をしたのか。読者の皆さんが，筆者の意図をくみ取っていただけたなら幸いである。

2009年初秋
PROJECT KySS

XML 設計の心得　目次

刊行にあたって
注意事項
著作権等について
本書の目的
プロローグ

第 1 章　木構造と XML

1.1 世界を表す木構造
1.1.1 情報の知覚と分類
事象の境界，情報の区別
静的な情報に「流れ」を観る
方向と順序のあるシーケンス
情報の分類
1.2 木構造による「流れ」の表現
1.2.1 いろいろなシーケンス
時間と空間による分類
社会的な共通概念による分類
1.2.2 木構造は柔軟に変化する
流れを表現できる木構造
複数のシーケンスを連結した木
シーケンスの方向を変えた木
ノードの順序を変えた木
1.2.3 木構造のセットによるデータの表現
木の部分は木
この世界を記述するモデル
1.3 変化する木構造
1.3.1 情報の構造と判断
Column「イメージを情報とする場合の木構造」
1.3.2 木による表現，表による表現
木構造データと表形式データ
柔軟な木構造
1.4 木構造と XML
1.4.1 データとメタデータ
1.4.2 XML によるメタデータの表現
Column「世界を記述する言語」

第 2 章　XML の基礎

2.1 XML 文書の構造
2.1.1 XML の概要
2.1.2 文書構造
2.1.3 整形式文書と妥当な XML 文書
2.1.4 整形式 XML 文書
整形式 XML 文書の構成
XML 宣言
要素
属性
コメント
処理命令
CDATA セクション
マーク付け
2.1.5 使用可能な文字および文字コード
要素名
特殊文字と空白文字
属性値
文字コード
2.1.6 妥当な XML 文書
2.1.7 XML 仕様書の記法
2.2 名前空間による識別
2.2.1 XML 名前空間
名前空間の意義
仕様書の入手先
2.2.2 名前空間の宣言と適用
宣言の方法
実際の宣言例
スコープ
2.2.3 デフォルト名前空間
2.3 ノード間の関係
2.3.1 異なる階層間の関係
親子関係
祖先子孫関係
2.3.2 同階層間の関係
兄弟関係
2.3.3 その他の縁戚関係
叔父甥，従兄関係

第 3 章　XML の処理

3.1 XML と XML 周辺仕様
3.1.1 基本の XML 周辺仕様
各仕様の役割
XML 周辺仕様の併用

3.2 XPath によるアドレス指定
3.2.1 XPath の概要
XPath の役割
XPath 仕様書と XQuery 仕様書の関係
3.2.2 ノードの扱いと順序
7 つのノード
文書順, 逆文書順
3.2.3 動的コンテキスト
XPath の用語と概念
パス式の記述例
3.2.4 基本構文
パス式
ステップ
軸
ノードテスト
述語
省略構文
記述例
演算子
3.3 XQuery によるクエリ
3.3.1 XQuery の概要
3.3.2 FLWOR 式
for 句, let 句, return 句
where 句
Order By 句
コンストラクタ
3.3.3 演算子と関数
仕様書の利用方法
代表的な関数と演算子
実務で利用する場合の注意事項
3.4 DOM によるデータ処理
3.4.1 DOM の概要
DOM の三大機能
構造モデル
3.4.2 仕様の現状
各仕様書の入手方法
3.4.3 DOM のメソッド
IDL 定義の見方
メソッド一覧
3.4.4 実際の記述例
XML ツリーの生成処理
要素ノードリストの取得
3.5 XSLT による変換
3.5.1 XSLT の概要
XSLT の機能
仕様の現状
3.5.2 基本構文
テンプレートルール
XSLT のルート要素と名前空間
XSLT の要素一覧
3.5.3 結果ツリーの生成
テンプレートルールの定義
テンプレートルールの適用
3.5.4 使用頻度の高い要素
条件分岐
ソート
空白と出力エスケープ
変数バインディング
3.5.5 XML 文書への XSLT スタイルシートの適用
XML 文書との関連付け
複数文書へのアクセス
DOM か XSLT か
3.6 LINQ to XML によるクエリ
3.6.1 LINQ to XML の概要
LINQ to XML の役割
LINQ クエリの基本構文
XML 文書の読み込み
基本のクエリ
LINQ to XML の軸
3.6.2 実際の記述例
XML ツリーの生成処理
要素ノードリストの取得
3.6.3 XPath 拡張機能の利用
Extensions クラスの概要
XPathSelectElements メソッドの使用例
XPathEvaluate メソッドの使用例
3.7 XML Schema による定義
3.7.1 XML Schema の概要
XML Schema の役割
XML Schema 文書作成および検証の必要度
XML Schema 仕様書の入手先
3.7.2 基本構文
ルート要素および要素
要素と属性の宣言と定義
3.7.3 データ型および出現方法の定義

制約条件の定義例
出現順序，回数，初期値の定義例

第 4 章　XML 設計の予備知識

4.1 社会の中の XML～誕生の背景～
4.1.1 ロジスティクス，CALS，SGML
4.1.2 HTML，XML，XHTML
4.2 社会の中の XML～普及と活用～
4.2.1 社会基盤を支える XML
4.2.2 ワンソース･マルチユース
木構造型のマルチユース
ネットワーク型のマルチユース
Topic XML の 10 の目標
4.3 社会の中の XML～役割と課題～
4.3.1 製品ライフサイクルとデータ･ライフサイクル
4.3.2 CALS の教訓
4.3.3 データ爆発の抑制
設計とメンテナンスによる抑制
メタデータによる廃棄の判断
4.4 設計者にもとめられるスキル
4.4.1 最重要スキルは予測力と判断力
サステナビリティの問題～蓄積期間と範囲の想定～
社会情勢の問題～時代の空気を読む～
哲学的問題～最小単位を定義する～
4.4.2 ヒトが設計することの意義
コンピュータとヒトの，予測･判断方法は異なる
全体像を俯瞰するヒトの能力を生かそう
4.4.3 XML 設計とは何か？
Coffee Break「未来のリサイクル業」

第 5 章　XML 設計の基本

5.1 設計の準備～企画･仕様調査～
5.1.1 設計の基本工程
5.1.2 設計の準備～企画～
企画書作成
処理アプリケーションの動作要件
ミドルウェア導入の判断
作成･処理のための仕様の決定
5.1.3 設計の準備～調査～
設計用データの準備
既存スキーマの調査
設計のルール作り
XML 文書の作成方法の決定
企画内容の再確認
5.2 木構造の設計
5.2.1 ボトムアップ型設計
ボトムアップ型設計の進め方
ボトムアップ型設計の利点
要素と属性の選択基準
要素の内容の分割
5.2.2 ヒトによる理解とコンピュータによる計算のバランス
5.2.3 技術以外の視点による微調整
5.3 文書内における整合性の確認
5.3.1 ノードの編集･追加･削除と整合性
物理的な動きとデータの不整合
モノの所在と，データの不整合
モノの状態と，データの不整合
ヒトやカネの動きと，データの不整合
5.3.2 不整合を抑止するルール作り
要素･属性･テキストの処理
ノードの編集と整合性
ノードの追加と整合性
ノードの削除と整合性
5.3.3 ノード間の関係と処理実行のタイミング
5.4 文書ファイル群の構成
5.4.1 複数 XML 文書ファイルの設計
ファイル構成とファイル間の関係
文書ファイルの構造
5.4.2 処理対象ファイルの指定
管理ファイルを併用しない場合
管理ファイルを併用する場合
5.5 ユーザインタフェースの設定と適用
5.5.1 設定ファイルによる表示や動作の制御
5.5.2 設定ファイルによるナビゲーションの制御
木構造のナビゲーション
レガシーデータの一部を利用した表引き
5.5.3 設定ファイルによるデザインの制御
変換テーブルの XML 化
手動によるデータ変更を表示に反映
RDB のレコードあるいはフィールドの表示の制御
1 ユーザ 1 設定のカスタマイズ
5.6 複数ファイル間における整合性の確認
5.6.1 複数ファイルの利点と問題点
管理ファイルとデータファイルの処理

ファイルの変更と整合性
ファイルの新規生成･追加と整合性
ファイルの削除と整合性
CRUD のタイミング
5.6.2 単一ファイルと複数ファイルの使い分け
5.7 タグ名の確定
5.7.1 タグ名の重複可能性の確認
5.7.2 要素内容および属性値のルール作り
5.8 設計後の作業
5.8.1 仕様書の作成
5.8.2 実データの準備とタグ付け

第 6 章　XML 設計の実践

6.1 ネイティブ XML 文書の検索処理
6.1.1 検索処理における XML の利点
6.1.2 データ作成への表計算ソフトの利用
6.1.3 XML 化に適した Excel レガシーデータの作成
表のレイアウトとフィールド名
データの有無と要素の最低出現回数の表現
深い階層の子孫要素の表現
出現回数未定要素の表現
6.2 不定形多階層構造文書の絞り込み検索
6.2.1 表データからの XML 文書の作成
データの内容から判断するファイル構成と構造
変換方法の検討
6.2.2 プログラムと XSLT スタイルシートによる変換
同じ意味を持つ要素の名前の統一
1 枚のワークシートの複数ファイルへの分割
重複を避けた構造化
6.2.3 絞り込み検索の実装
6.3 ネイティブ XML 文書のフルテキスト検索
6.3.1 ストップワード方式による検索対象文書の作成
Excel レガシーデータとデータファイルの作成
データファイルのコンパクト化
ストップワードによるコンパクト化の例
6.3.2 検索対象文書生成アプリケーションの実装
プログラムと XSLT スタイルシートの併用
管理ファイルと複数データファイルの結合
大文字小文字，記号，インデントの扱い
6.3.3 ストップワードのカスタマイズ
6.4 検索条件指定によるフレーズ検索
6.4.1 要約記述の義務付けによるサイズダウン
条件指定検索の目的
複数の要約と管理ファイルの結合
6.4.2 ユーザの選択に対応するプログラム
二者択一の選択肢に対応するデータ作成
XSLT による，変換処理に利用する属性の追加
疑似インデックスによるプログラムの簡素化
6.4.3 練習から汎用的な実践へ
6.5 設計後の工程に影響する問題
6.5.1 データ作成にかかわる問題
6.5.2 アプリケーションの設計と実装にかかわる問題
処理に必要な値の指定
処理に必要な値の取得
スキーマ変更の影響の抑制
フルテキスト検索の文書構造に対する誤解
6.5.3 更新･運用･作業管理における問題
頻繁な更新による応急処置
最大フィールド数の増加
開発後の中途階層追加
想定外データの追加
データ管理における意識
コードのレビューとテスト
報連相と文書化で防ぐデータの不整合
完璧でないからこそ，完璧を目指す
6.6 アルゴリズム的手法を超えて
6.6.1 本書を超えて，手順なき実践の世界へ
入門者は手順に従い，経験者は俯瞰する
適切な判断に基づく木を植えよう

エピローグ
筆者紹介
奥付

プロローグ

XML1.0 勧告仕様の発表から 10 年が過ぎた。バージョンアップを重ねても，その目標とゴールは変わらない。仕様書の「目標とゴール」は，次の 2 つの設計目標から始まる。

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.

この 2 つは，そのまま XML 設計の心得であるといえよう。

XML 文書は，「いつか，どこかで，あるいは将来開発されるであろうアプリケーションによって」利用される可能性がある。
そのため，XML を学び実践するにあたっては，世界を俯瞰する姿勢が必要となる。長期的で広い視野を持たなければ，獲得した技術を役立てるどころか，むしろ再利用困難なデータ爆発に手を貸すことになりかねない。そればかりか，ヒトの平穏な暮らしを揺るがすリスクを，社会の中に持ち込む恐れがある。
———それが，XML 設計の難しさでもあり，やりがいでもある。

XML 設計を始めるのならば，時間と空間の認識方法を改めて，イメージを膨らませる必要がある。

まずは，自分のパソコンからネットワークでつながった先にある無数のコンピュータを思い浮かべよう。現在からつながった未来のデータセンターで駆動しているサーバに思いを馳せよう。遠い未来のどこかの国のエンジニアたちが，これから作成するXML文書を，アーカイブとして活用している姿に，今この場所から声援を送ってみよう。まだインターネットを知らないこれから生まれてくる子供たちが，アプリケーションを開発している光景を心待ちにしておこう。

XML 文書は永きにわたって蓄積され，活用される。
XML は，何十年何百年と年輪を重ね，風雨に耐えつつ，立ち続ける木に似ている。木は，光りを浴びるときにはいつも，その根元に，シンプルでエレガントな陰を映すだろう。

我々もシンプルでエレガントな構造を目指したいものである。

第 1 章　木構造と XML

本章では，木構造で表現可能な情報について理解を深める。

美しいアイディアは醜いアイディアに比べて，正しいアイディアであるチャンスははるかに大きい。
（A beautiful idea has a much greater chance of being a correct idea than an ugly one.）
ロジャー・ペンローズ「皇帝の新しい心」，林一訳／みすず書房

1.1 世界を表す木構造

1.1.1 情報の知覚と分類

事象の境界，情報の区別

いま，ある一人の IT エンジニアが，客先へと急いでいる。本日 9 時に A 社，10 時に B 社，11 時に C 社にアポをとっており，新しいシステムの説明をすることになっているのだ。

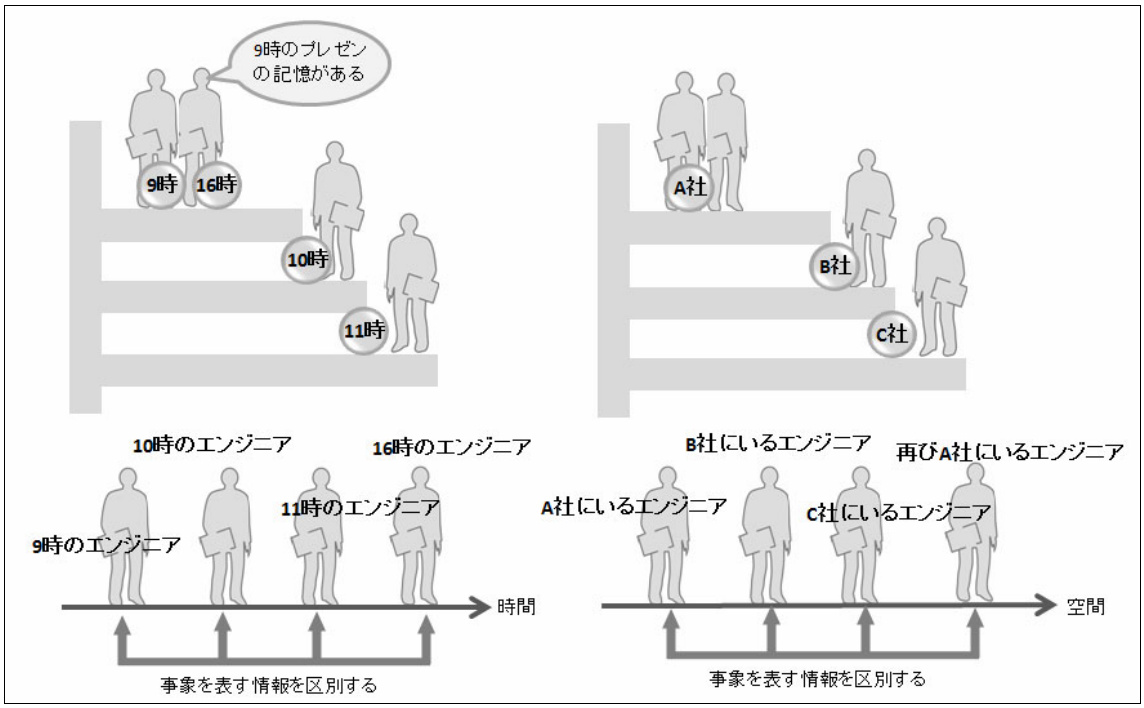
彼は思う，「ああ，時間が足りない！自分が 3 人いたら一度で済むのに！」と。だが，悲しいかな，この IT エンジニアは，同じ時間，たとえば 9 時に，A 社と B 社と C 社を訪問することはできない。完全に同じ時間に，このエンジニアの存在が重なり合うことはない。

このエンジニア氏は，B 社と C 社の訪問を終えたところで，A 社での説明不足に気付いて，嘆く。「9 時の時点に戻れたらいいのに！」と。そこで，再訪問のため，16 時にアポをとる。だが悲しいかな，再訪問した彼は，同じ A 社の会議室にいながら，初回訪問時の過去の彼ではない。9 時の彼と 16 時の彼は，完全に同一ではない———彼の身体を構成する分子の入れ替わりについて考えるまでもなく———16 時の彼は，9 時のプレゼンでの説明不足に気付いてしまった彼，既にその経験を蓄積した脳を持ち合わせている彼である。

このエンジニアは，同じ時間に異なる空間に存在することができず，異なる時間に同じ空間に存在することもできない。

このエンジニア自身は，いや，我々も，この世界の，重なり合わない事象の間に境界線を引き，事象を表す情報を区別している（●図 1-1）。

●図 1-1　我々は，重なり合わない現象の間に境界線を引き，現象を表す情報を区別している



まるで我々にとって，存在とは，「我々の感知可能な」時空における，重なり合うことのない原子の「一意な」座標（ポジション）のセット，あるいは，一意であるかのように知覚されうる原子のポジションのセット，そういったセットを形作る設計情報であるかのようである※1。

※1：XML 設計の本であるにもかかわらず，こういった話から始めたのには理由がある。第 4 章で説明している。

静的な情報に「流れ」を観る

重なり合うことのないこのエンジニアの所在の，時間と空間の情報を，それぞれ図で表すと，●図 1-2 のようになる。

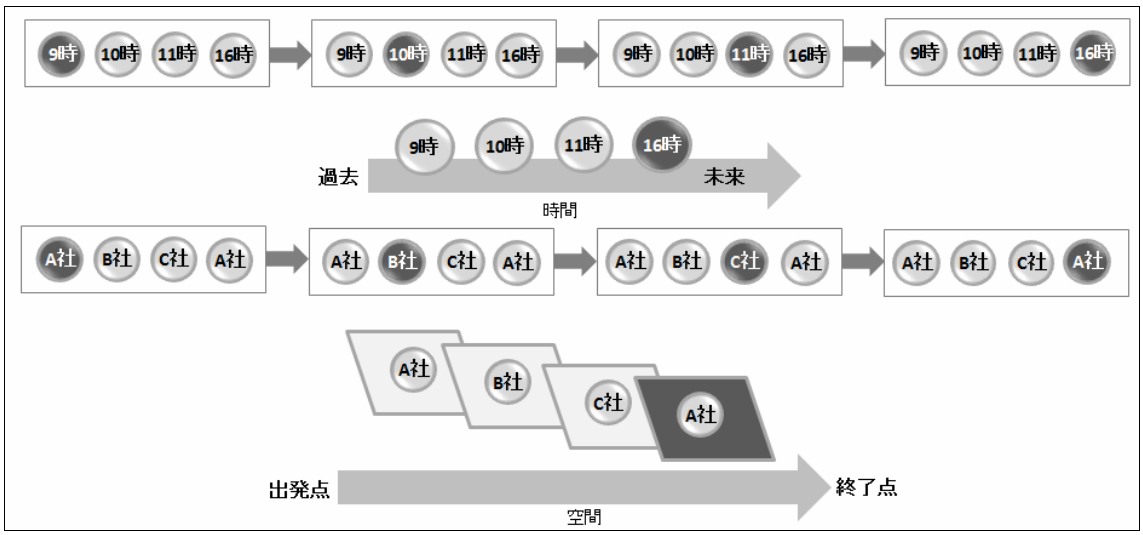
このエンジニアにとって未来の出来事であれ，過去の出来事であれ，どちらも，この図の上では静的（static）である。そこには時間や空間の流れなどない。

●図 1-2　あるITエンジニアの訪問時間と訪問先の情報。時間や空間の流れはなく静的である。

ところが，このエンジニアが，―――いや，彼の上司や労務管理の担当者や，あるいは我々でもいいのだが―――，この静的な●図 1-2 の事象を観るやいなや，「流れ」が生まれる。ヒトはその流れに「変化」を知覚する。

それはキーフレーム・アニメーションに似ている。個々のフレーム（コマ）は静止画であるけれども，複数のフレームが順番に表示されると，そこには変化が立ち現れるのである（●図 1-3）。

●図 1-3　静的な情報も，観る者がいると，流れができる。



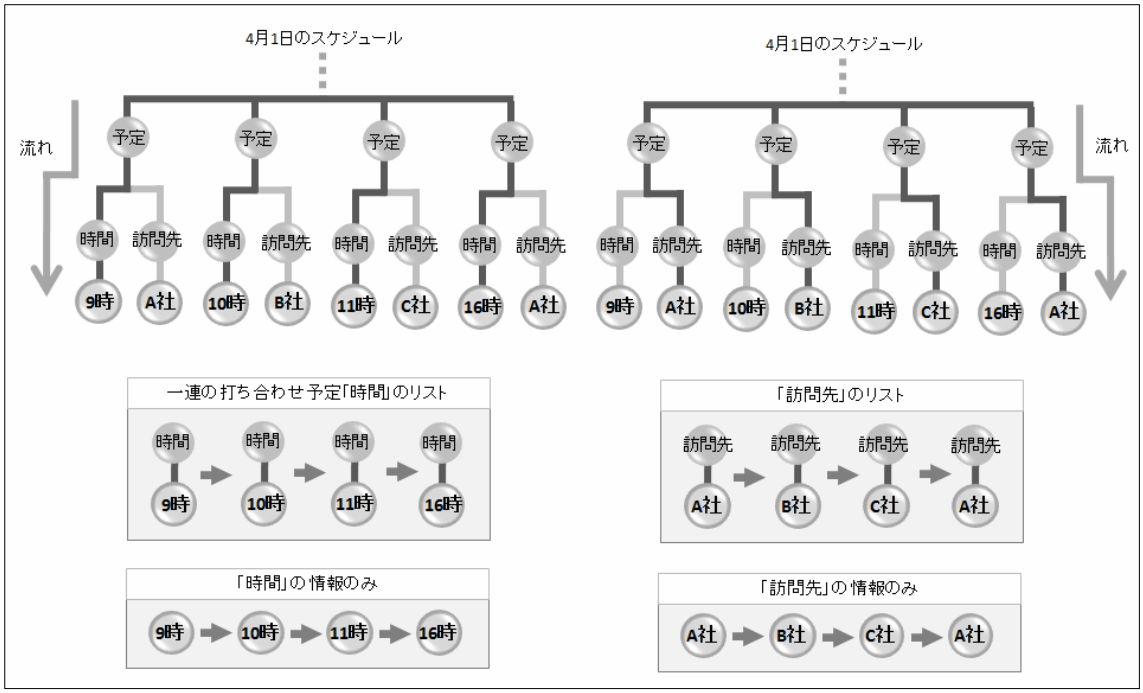
方向と順序のあるシーケンス

仮に，このエンジニアの打ち合わせ日を，4 月 1 日としよう。彼の 4 月 1 日のスケジュールには，4 つの「予定」があり，各々の「予定」について「訪問先」と「訪問時刻」が決まっている。

この予定を表わす情報を，●で表わしたノード（※2）を用いて表現すると，●図 1-4 のようになる。この図の事象を観ると，そこにはいくつもの流れが生まれる。

情報を観るヒトの意識のフォーカスは変化し――もちろん脳内には物理的な変化が生じ―――，意識のフォーカスの軌跡は流れとなる。

●図 1-4 「流れ」のある情報の部分にも「流れ」がある



●図 1-4 では，まず，4 月 1 日のスケジュールに「予定」があり，その予定には「時間」がある，という流れが見られる。あるいは，「予定」があって，その予定には「訪問先」がある，という流れも見られる。

流れる情報の部分もまた流れを形作る。一連の打ち合わせ予定「時間」のリストには，過去から未来へ向かう流れがある。「訪問先」のリストには，このエンジニアの所在を連ねた流れがある。

情報のみを連ねても，流れがある。「予定」の「訪問先」を順番に観ると，「A 社→B 社→C 社→A 社」という流れがある。「予定」の「時間」を順番に観ると，「9 時→10 時→11 時→16 時」という流れがある。

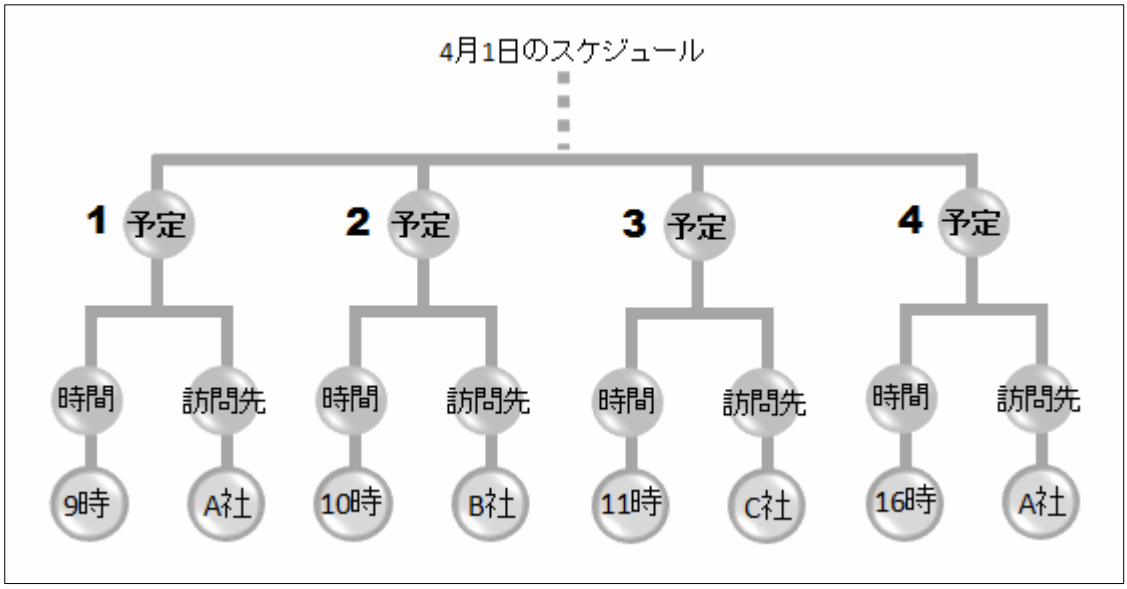
流れには，方向と順序がある。我々が流れていると考えるものは，シーケンス（一連のノードのセット）」である。

たとえば，4 つの「予定」ノードには，1 番目の予定，2 番目の予定，3 番目の予定，4 番目の予定，という方向に進む順序がある（●図 1-5）。

各「予定」ノードの，1，2，3，4 というポジションは静的だが，我々が 1，2，3，4 の順番に「予定」ノードをたどると，意識のフォーカスは「1→2→3→4」と移動し，文脈上のポジションは「1」「2」「3」「4」と順次移動し，「1→2→3→4」という流れが生まれる。「予定」ノードのセットは，シーケンスである（**※3**）。

図を観る側がヒトではなく，コンピュータであったとしても，走査対象とするノードは，動的（dynamic）に変わる。我々が意識したり，コンピュータが計算し始めるやいなや，ただそこにある事象は流れ始め，シーケンスが認められるようになる。

●図 1-5　流れには，方向と順序を持つシーケンスがある



※2：ノードとは直訳すれば「節」であり，グラフにおける頂点のようなものをイメージしてほしい。

※3：本章の解説に用いる言葉（シーケンス，ポジション，フォーカス等）は，実は XML および周辺仕様の用語でもあるが，本章では一般用語として，フィーリングで読み飛ばして差し支えない。XML および周辺仕様については，第 2 章，第 3 章で解説している。

情報の分類

前項のように，我々は，我々の知覚する情報の所在を表す時空間―――情報がいつ存在していると知覚されるか，情報がどこに存在していると知覚されるか――によって，この世界の事象を表す情報を，分けて並べることができる。分けて並べた 1 セットの情報は，さらに大きな情報の部分である。

●図 1-5 でいえば，我々は，時間および訪問先の情報を，区別することができる。そして，時間と訪問先を「予定」として分類することができる。「予定」という情報のセットは，ある一人の IT エンジニアの「4 月 1 日のスケジュール」の部分である。

我々はもう 1 つ，この世界の事象を表す情報を，我々の社会的な概念のプライオリティ（優先順位）によっても分けることができる。ここでいうプライオリティとは，我々が社会生活を送るうえで必要な，分類の大小，五十音やアルファベットや数字の並び順，社会的価値の高低といったものである。

ただし，同じ人類であっても，その多様性から，誰もが同じように情報を分類するとは限らない。時間と空間による分類については，文化圏の違いよりも，個人的な差異の方が大きいかもしれない。ヒトの時間や空間に対する感覚はじつに多様であるので，それぞれ分類方法の異なる可能性がある。「過去から未来へと一方向に時間が流れている」という感覚が，すべてのヒトにとって共通であるとは限らない。

一方，社会的な概念による分類は，それぞれの所属する社会によって，大きく異なる可能性がある。「外→内」「高→低」あるいはその逆のシーケンスひとつとっても，何を高とし，何を低とするかは，文化的背景や歴史的背景などにより異なり，同じ文化圏であっても社会事情によって異なる。

1.2 木構造による「流れ」の表現

1.2.1 いろいろなシーケンス

時間と空間による分類

ここまで，ある IT エンジニアの作業予定を例として説明してきたが，ほかにも，我々の暮らすこの世界の情報には，いろいろなシーケンスの認められる，情報の流れがある。いくつか身近な例をあげてみよう。

たとえば，我々が「ソフトウェアをダウンロードしてインストールして再起動した」ならば，その操作手順には，流れがある。

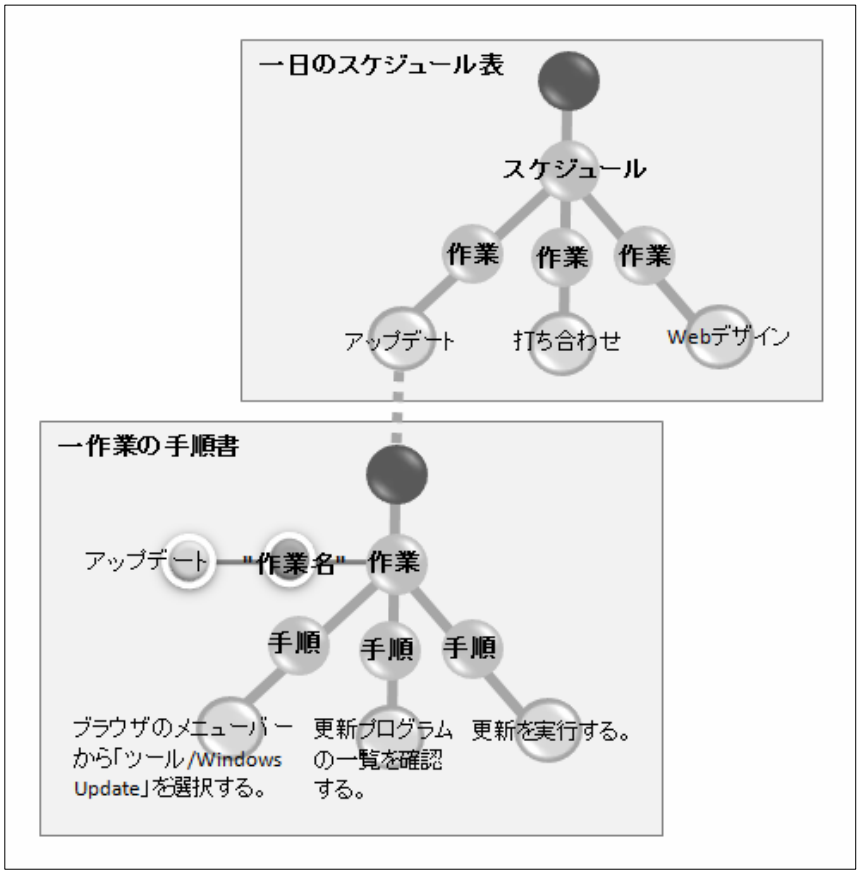
また，既に実行された過去の情報でなくとも，これから「ソフトウェアをダウンロードしてインストールして再起動する予定である」場合であっても，操作手順そのものに，「ソフトウェアのダウンロード→インストール→再起動」という流れがある。

さらに，先にも述べたように，流れは，ヒトの行動だけにしか認められないわけではない。文書であっても，記載された情報に流れがある。操作手順書には，「ソフトウェアのダウンロード→インストール→再起動」という流れがある。ここでいう文書とは，操作手順書に限らない。工程表，営業日報，出張予定表，仕様書の変更履歴，作業指示書などにも，流れがある。

こういった流れを，先の●図 1-5 のようなノードを使った図で表現してみよう。

一例として，一日のスケジュール表と，一作業の手順書を，表してみることにする（●図 1-6）。

●図 1-6 スケジュール表と，操作手順を表した図の一例

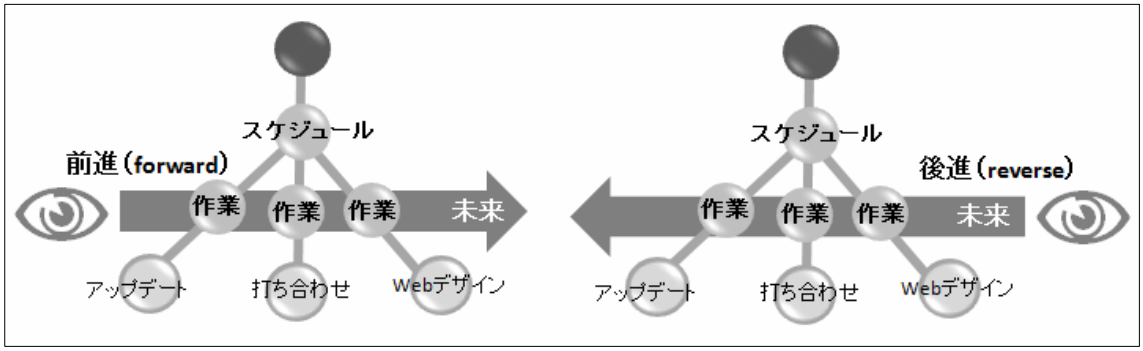


この図には，1日のスケジュールとして，アップデート･打ち合わせ･Webデザイン，という3つの作業がある。このうち，アップデートに関しては，「Windows Update を選択する → 更新プログラムの一覧を確認する → 更新を実行する」という手順で行うものとする。この図では「作業」ノードに対して，"作業名"が「アップデート」であるという属性（性質，プロパティ）が付いている。

また，●図1-6には，過去→未来という方向の流れしかないわけではない。我々生きている人間にとって時間は不可逆だが，情報の流れに認められるシーケンスの方向は，視点次第である。

視点によって，シーケンスの方向と順序は異なってくる。過去→未来という前進（forward）方向の流れもあれば，逆に，「未来→過去」という後進（reverse）方向への流れもある（●図1-7）。

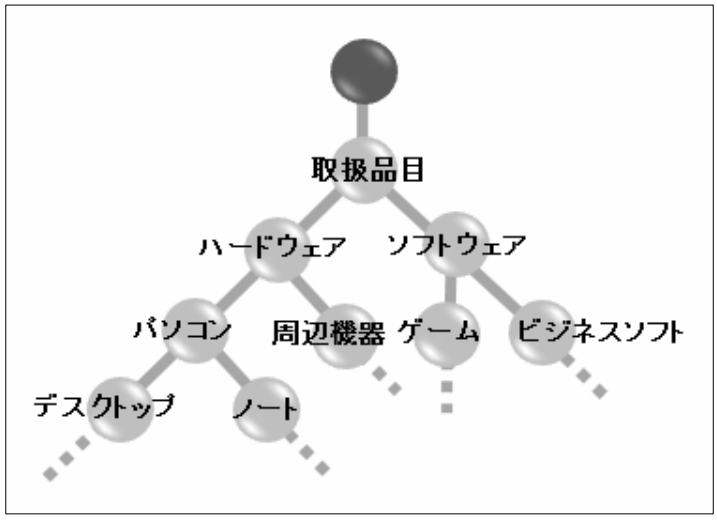
●図1-7 視点によって，流れの方向は変わる。



社会的な共通概念による分類

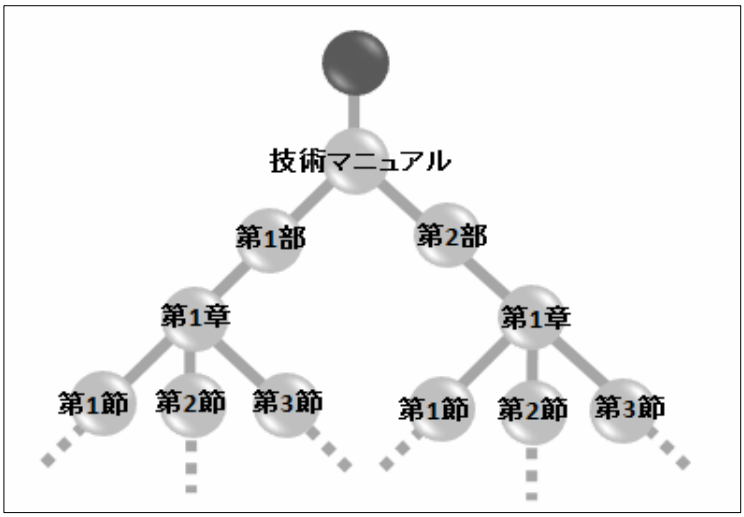
社会的な共通概念による分類と言えば，図書分類であるとか，動植物の分類であるとか，商品分類などを思い浮かべるかもしれない。たとえば，絞り込み検索のアプリケーションでは，商品情報が，任意のカテゴリに分かれており，その中に複数の商品があり，各商品の中に型番や色やサイズや単価がある，といったものがある。（●図1-8）。

●図 1-8 絞り込み検索には，「外から内」への流れがある。



また，マニュアル類にも，「第1章」「第1節」「第1項」のように，「外から内」への流れが認められる（●**図1-9**）。これは，第1章が第1節より大きい，第1節は第1項より大きい，といったプライオリティによる分類である。さらに，「第1章」「第2章」といった流れもある。マニュアルに限らず，報告書や論文や帳票，書籍や記事でも，同様である。

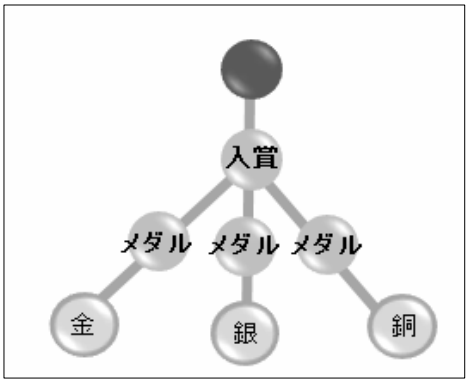
●図 1-9 マニュアルには，「外→内」「前→後」の流れがある



より分かりやすい例としては，社会的な地位があげられる（●図 1-10）。

社長，専務，部長，課長であるとか，1 位，2 位，3 位といったものである。アルファベット順や 50 音順は，我々の社会的な概念に基づく流れである。日本語を理解する者には，たとえば，「あ」は「い」より先，といった共通認識にもとづくプライオリティがある。

●図 1-10 社会的な「高→低」へのプライオリティによる流れがある。



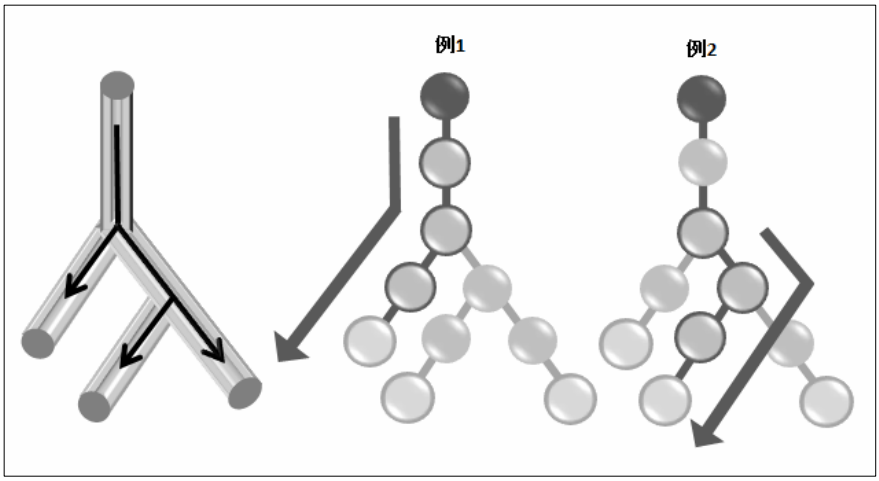
1.2.2 木構造は柔軟に変化する

流れを表現できる木構造

既にお気付きのことと思うが，●図 1-6 以降の構造は，説明するまでもなく「木構造」である。

木を眺めれば，根っこから幹が伸び，枝葉が生い茂っている。地下から水を吸い上げ，成長していく，その流れを模した形で，情報を表現することができる（●図 1-11）。当然のことながら，木構造で表現すべき情報とは，流れのある情報であって，関連性のないバラバラの情報の寄せ集めではない。

●図 1-11 木構造では流れを表現することができる。



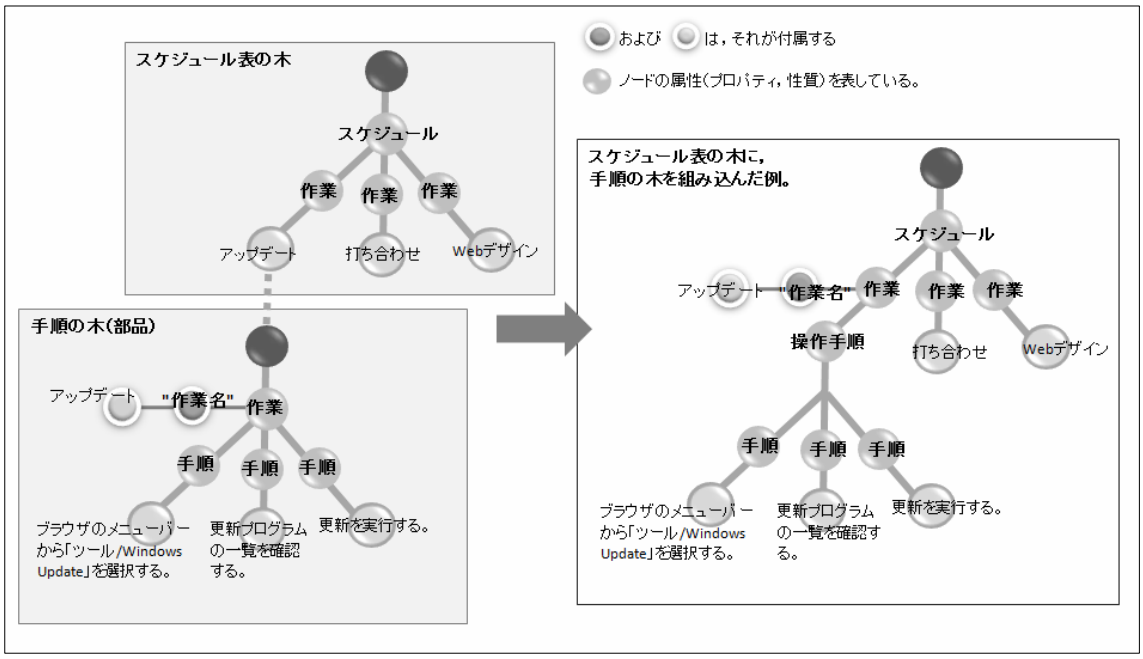
複数のシーケンスを連結した木

木構造では，いろいろな流れを表現することができる。

流れを表す木は，実在の樹木ではなく，木の形をした情報の集まりであるから，人が手を加えても生命が失われることはない。我々は，木の痛みを恐れることなく，木の全体や部分を増やしたり，接ぎ木したり，盆栽のように枝ぶりを変えたり，枝葉を伐採することができる。つまり，木のノードを，複製したり，連結したり，構造を変更したり，削除することができる。

たとえば，前掲の●図1-6の一日のスケジュール表と，一作業の手順書は，別の時間の流れを持つ木であるが，どちらも「シーケンス」があることに変わりはない。そこで，スケジュール表に操作手順を連結してみよう。この結果は，●図 1-12 のようになる。

●図 1-12　●図 1-6 のスケジュール表と，操作手順は別の流れを持つが，組み込むこともできる。



もし，スケジュール表が細分化したならば，枝葉を追加すればよい。ここではスケジュールの木の下位に手順の木を連結しているが，複数のスケジュールの木を，1 カ月単位，年単位という上位の別の木に連結することもできる。

こういった接ぎ木は，実際には，人が手作業で行うというよりも，プログラムで自動的に行う。その方法については，第 3 章以降で述べる。

シーケンスの方向を変えた木

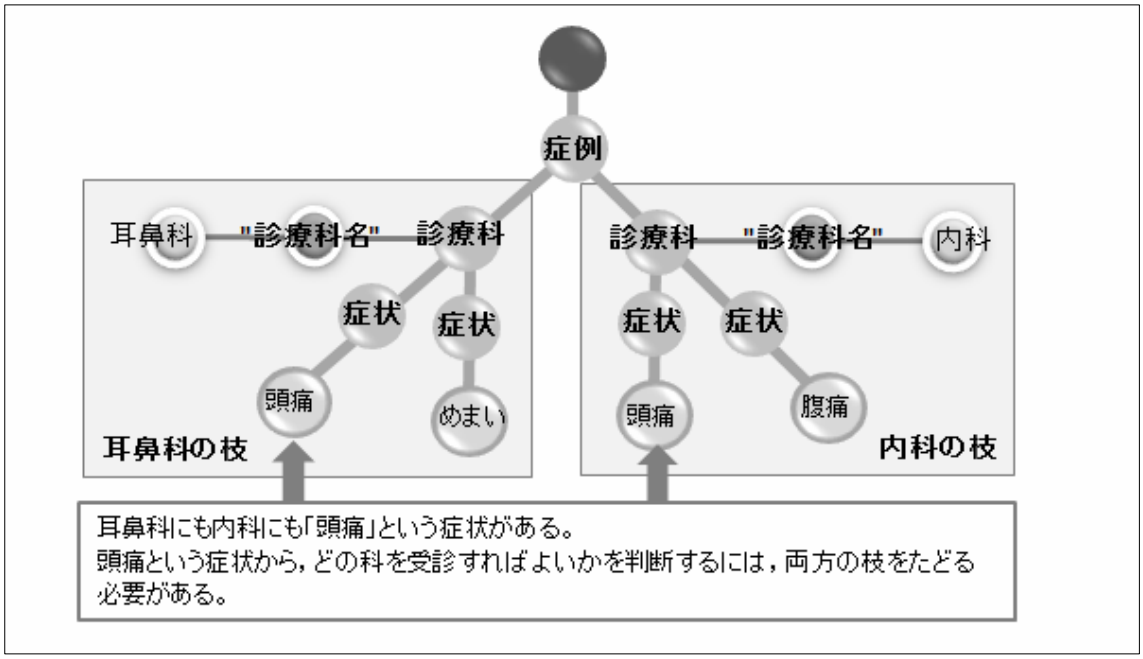
一度構築した木を，異なる枝ぶりに変えてしまうこともできる。情報そのものは同じであっても，木を観ているヒトの側の事情に合わせて，別の方向を持つ適した構造に変えてしまうのである。

なにしろ，木構造には，シーケンスの方向は一つであり，枝をたどる経路が 1 種類に限られるという特性がある。これはパソコン内のファイルへのパスが限られるのと同じである。この特性は利点でもあるが，そのままでは情報を利用しにくいこともある。木の幹の向こう側の葉影を見ることができないように。

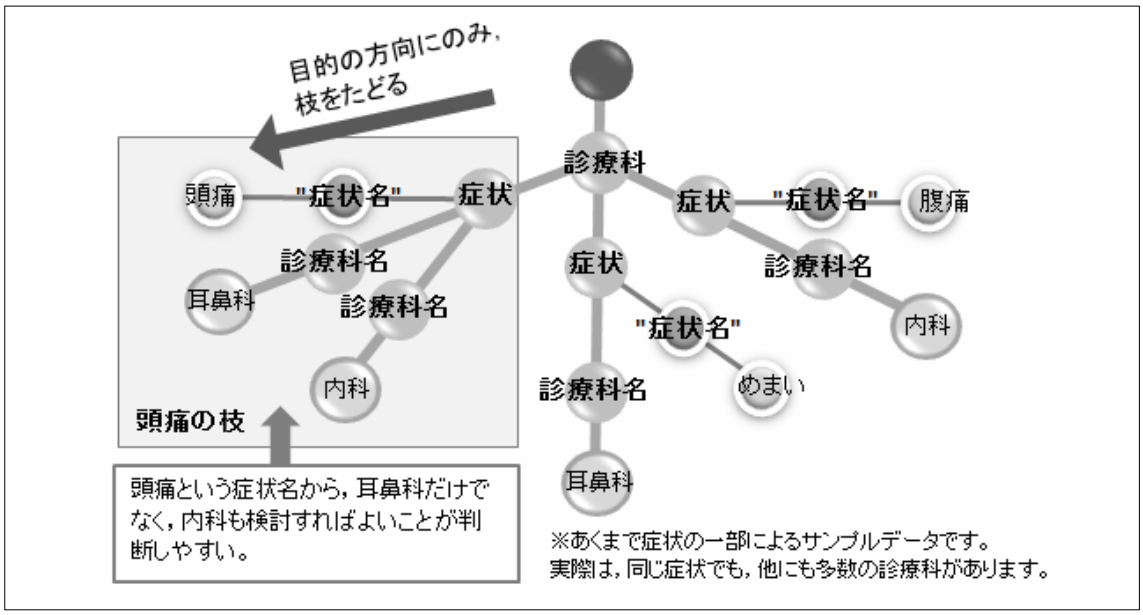
具体的に説明しよう。●図 1-13 は，病気の症例データを表した木である。「診療科」の属性である"診療科名"から，たどるべき枝を選択した時点で，この木を見ているユーザの意識のフォーカスは，「診療科」の中にある「症状」に移る。もし，アレルギー性鼻炎のある人が，「頭痛」を「耳鼻科」の症状だという考えのもとに枝をたどると，内科の枝は意識のフォーカスから外れてしまう。この木を利用する目的が「症状から診療科を決定する」ことならば，この構造は，木の特長を生かしているとはいえない。同じ症状がある診療科の数だけ，何度も枝をたどり直す必要が生じてしまうからである。

逆に，●図 1-14 のように，症状によって分類した木では，診療科から症状を探すことは困難である。もし目的が「各診療科で見られる症状を知る」ことであれば，この構造は，木の特長を生かしているとはいえない。

●図 1-13　診療科別分類で，症状を表現した木の例。



●図 1-14　症状別分類で，診療科名を表現した木の例。



このように，木構造にある「シーケンス」を生かすためには，まず，木の目的を明確にする必要がある。もし，複数の目的があって，木の情報を利用するユーザの，意識の流れが木の促す方向と異なるならば，別の経路を持つ複数の木が必要になる。

もし，これが，同じ医療関係データでも，症状データではなく，医療カルテであったなら，と想像してみてほしい。病院別のカルテの木しかなければ，患者は，異なる病院を受診する度に，既往症や副作用などを逐一申告しなければならなくなる。一方，病院別のカルテの木以外に，患者別の木もあれば（且つ，その木が共有されていれば。現実的には実施困難であろうが），出張先で体調を崩したとしても，どの病院に飛び込もうと，病院側は登録済みの既往症やアレルギーのデータを検索することができる。木は一方向へのシーケンスしか表現できないだけに，その構造はユーザ・インタフェースの問題に直結するのである。

このような場合は，どちらか一方のオリジナルの木から，構造の異なる木を生成して，目的別に木を使い分けるという方法が考えられる（構造変換の具体的な方法については，第 3 章以降で述べる）。

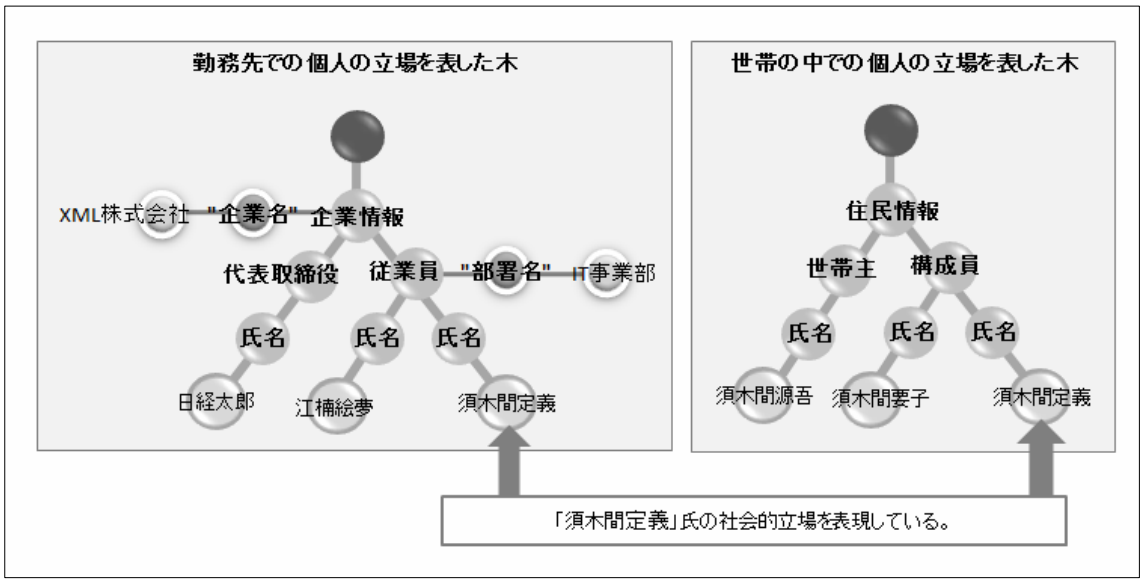
ノードの順序を変えた木

2 つ以上のオブジェクトがあるとき，オブジェクト間には相対的なプライオリティが発生する。ここでいうオブジェクトとは，木やノード群や単一のノードである。木を見る視点を持つ者の，各オブジェクトに対するプライオリティが異なれば，木構造も異なる。

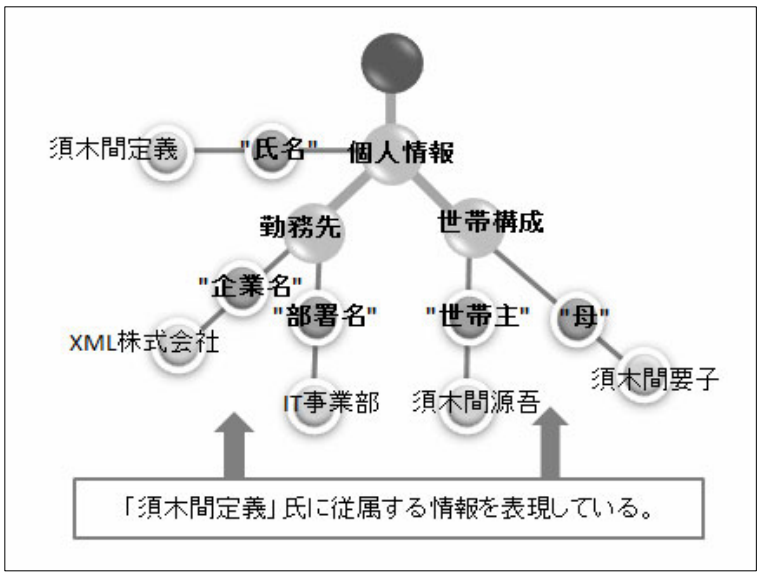
ここでは，社会の中の個人を表現した木を例に説明しよう。同じ一人の人物を表現しても，勤務先や，家族など，帰属先別の木ができる。●図 1-15 では，いずれの木も，社会的役割のプライオリティが個人のそれより高い。

一方，●図 1-16 は，個人の社会的立場を表現した木である。同じ人物を対象としても，個人のプライオリティが帰属先よりも高ければ，帰属先のデータが従属する構造となる。

●図 1-15　社会的役割のプライオリティが個人より高い場合の，2 本の木。



●図 1-16 個人のプライオリティが帰属先より高い場合の木。

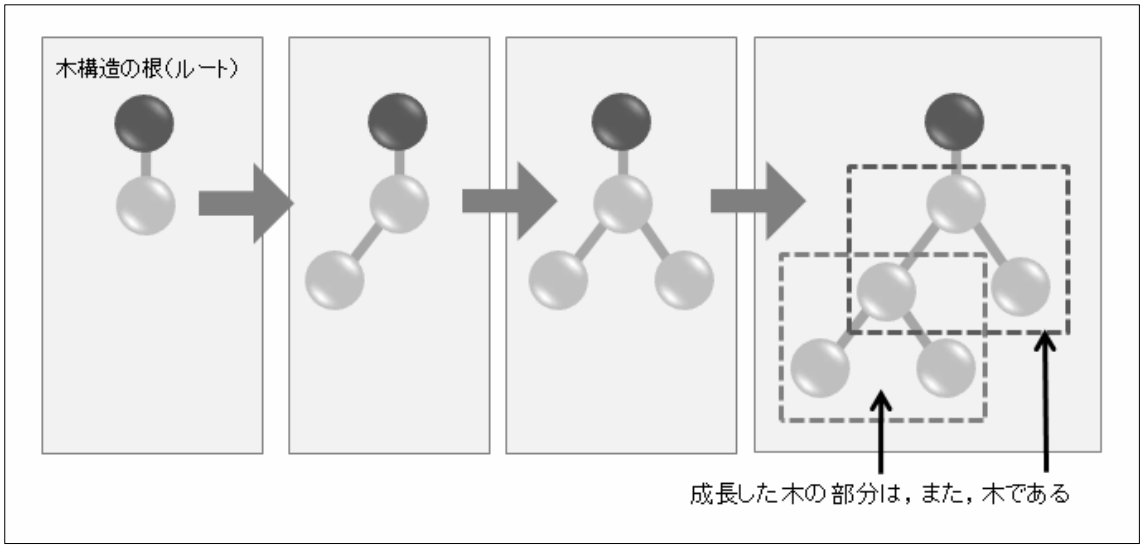


1.2.3 木構造のセットによるデータの表現

木の部分は木

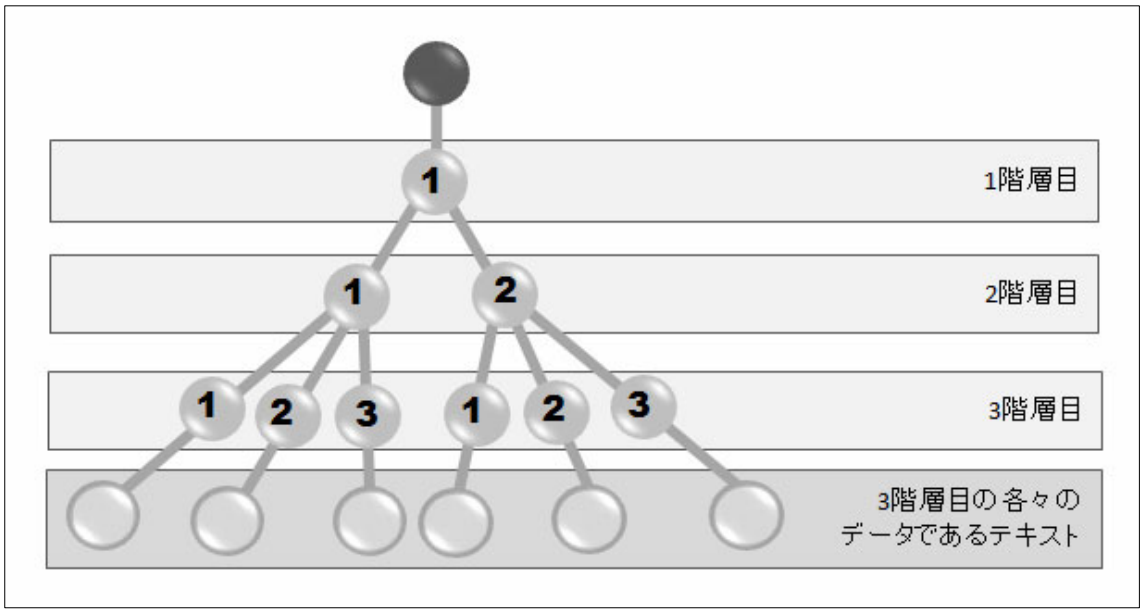
木の基本構造は，根（ルート）から生えた 1 個のノードである。ノードが追加されると，木は成長していく。成長した木の部分は，また，木である。つまり，木構造のセットによって木全体の構造を表現することができる（●図 1-17）。1 本の木の部分を成す一連のノード群にもまた，シーケンスがある※4。

●図 1-17 根（ルート）から生えたノードから木は成長していく



木は，順序のあるシーケンスであるから，どのノードにも木構造の中での順序がある（第 3 章で解説，図 3-4 参照）。さらに，同一階層内での順序もある（●図 1-18）。この木の中の順序，および階層内順序によって，任意のノードを特定することができ，各ノードを一意なものとして扱いうる。木構造では，必ずしも各ノードに対して一意な主キーを付加せずとも，任意のノードを選択することができるのである。

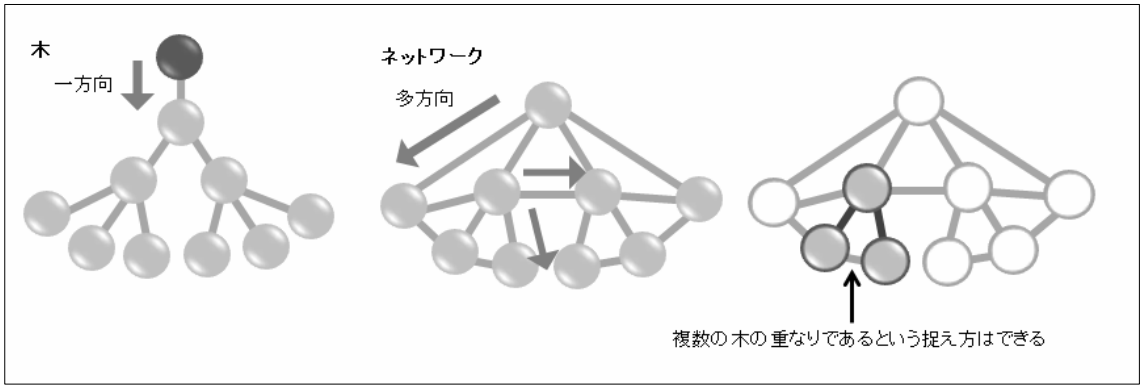
●図 1-18　木では，どのノードも順序指定により一意なキーとして扱える



ただし，木はシーケンスを表現することはできるが，その方向は，先にも述べたように「一方向」である。単純な木のままでは「複数の方向を持つ，順序が絶対的ではないシーケンス」を，直接表現することはできない。たとえば，ネットワークを表現することはできない。

ただし，ネットワークを，複数の木の重なりであると考えるならば（●図 1-19），木で表現可能ということも（多少の無理はあるにせよ）言えなくはない。

●図 1-19　木とネットワークの違いの一例。木のシーケンスは一方向である



※4：本書で述べている「木」とは，あくまで本書のテーマである XML を表現するためのモデルである。●図 1-17 の根（ルート）から生えた 1 個のノードに接続する枝はあってもなくてもよく，あったとしても次数は問わない。

この世界を記述するモデル

先に述べたように，我々が意識したり，コンピュータが計算し始めるやいなや，単なる事象の所在は流れ始め，シーケンスが認められるようになる。我々が意識するところ流れはあるのだから，この世界の事象には流れがあり，それは木構造で表現可能なのである。木構造は，この時間の流れと空間の拡がりのある世界を記述可能な，すばらしく良くできたモデルである。

木のイメージを心にとめて，この世界を眺めれば，実際，そこにも，あそこにも木があることに気付くだ

ろう。ディレクトリ・ツリーはなじみ深い木であるが，家系図，フローチャート，回路図，言語構造，計算法，身近なところでは料理の手順にさえ，生活の中にも学問の中にも，木構造はある。

およそ，この世界の情報で，木によって表現できないものはないとさえ思われる。表現できないとすれば，この世界ではない世界の概念だけといっても過言ではないかもしれない（本章末別掲 Column も参照）。ただし，その概念の一部が，数式というインタフェースで表現されうるのであれば，その数式を木構造で表現することは可能であろう。

1.3 変化する木構造

1.3.1 情報の構造と判断

先に述べたように，同じ情報を表すとしても，木構造は，必ずしも1種類とは限らない。目的や，視点などによって，適切な構造が異なる。そういった目的や視点を定め，何が最適の構造かを判断することは，設計者の務めである。

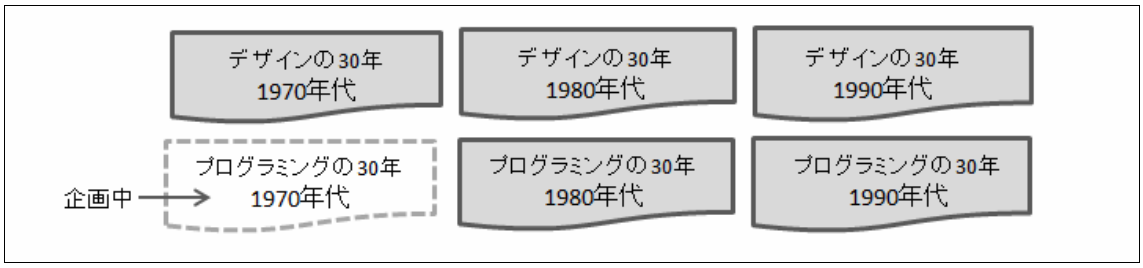
実際にありそうな，簡単な例で見ていこう。

●図 1-20 は，5 本のオンライン記事の情報である。「デザインの 30 年」と題する記事は，1970 年代・1980 年代・1990 年代の 3 本に分かれている。また，「プログラミングの 30 年」と題する記事は，1980 年代・1990 年代の 2 本に分かれており，1970 年代については企画中であって，まだ執筆されていない。

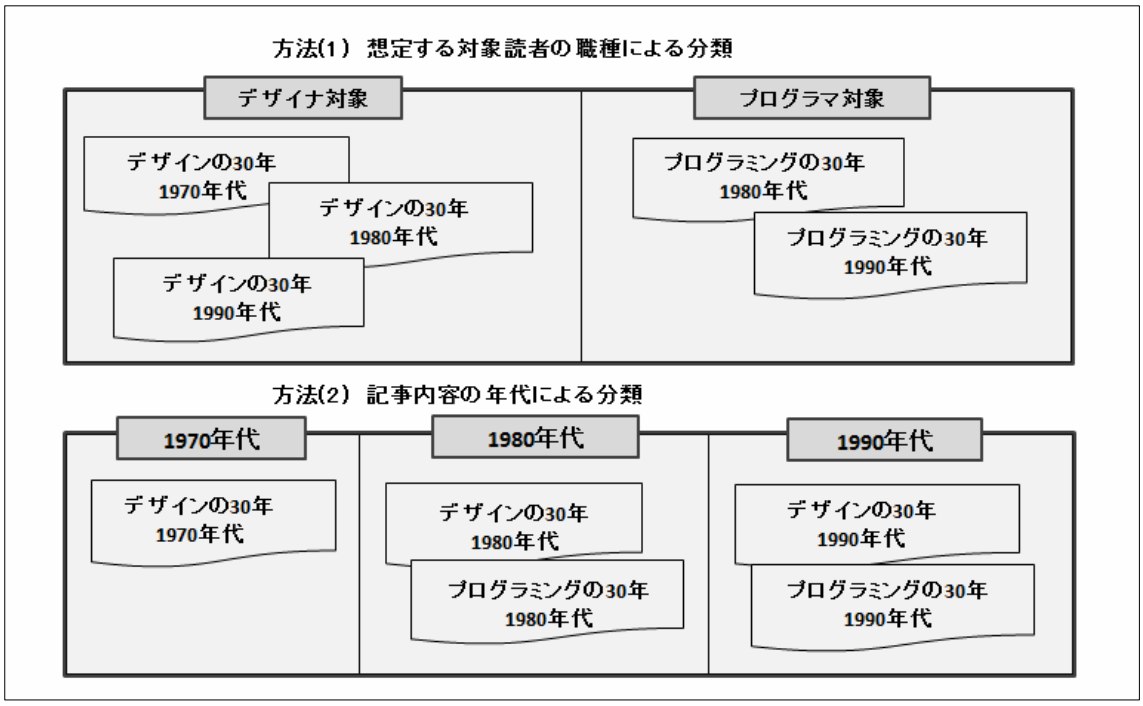
執筆済みのこれら 5 本の記事を Web サイト上で公開するために，分類と掲載順序を検討して，構成を決めていくものとする。一般的には，●図 1-21 のような，2 種類の分類方法が考えられる。

このように，同じ情報であっても，考えられうる構造は，一つではないのである（別掲 Column も参照）。どの構造を採用するかは，情報の活用方法によって異なってくる。本書では，職種による構造（方法 1）にしたがって，以降の解説を進めていく。

●図 1-20 5 本のオンライン記事



●図 1-21　●図 1-20 の 5 本の記事を分類する基本的な 2 つの方法



Column「イメージを情報とする場合の木構造」

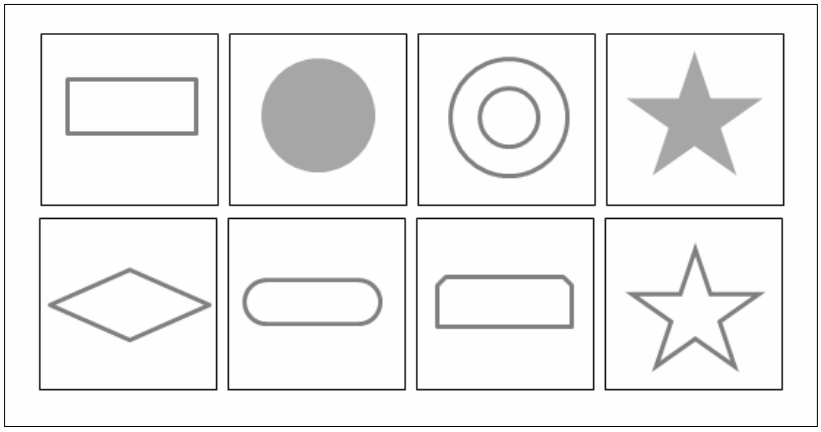
意味の明確な言葉による情報ではなく，シンボルのように，ヒトによって解釈の異なるイメージを，木構造で表現しなければならない場合もある。

たとえば，●図 1-22 のような，図形を描いた 8 枚のカードがあるとする。各々の図形の性質にしたがって木構造を考えるなら，皆さんはどうするだろうか？

ソフトウェア･エンジニアであれば，まず「フローチャートの図記号」と「それ以外の図形（●◎☆）」の 2 つに分けるかもしれない。かたやデザイナであれば，「塗りつぶしなし／塗りつぶしあり」あるいは「アタリケイ／ケイあり」といった方法で分けるかもしれない。

このような，実に単純な図形であっても，ヒトによってシンボルの持つ「意味」が異なれば，判断は異なる。言葉と違って，イメージの扱いは難しい。

●図 1-22　対象の持つ「意味」が異なれば，ヒトによって木構造の判断は異なる



1.3.2 木による表現，表による表現

木構造データと表形式データ

●図 1-21 のような情報は，木構造でしか表現できないわけではない※5。

「職種による構造」を単純な表形式で表現すると，●図 1-23 のようになる。この例ではフィールド数もデータ件数も少ないため，同一テーブルに登録している。リレーションシップは張っていないが，フィールドとレコードで表現する，いわゆる RDB（リレーショナルデータベース）の構造である。一意なキーとして，各記事に「ページ ID」を振っている。

この●図 1-23 のように，RDB では，各フィールド間の関係も，各レコード間の関係もフラットである。

一方，木構造で表現すると，●図 1-24 のようになり，シーケンスがある。

●図 1-23　●図 1-3 の「職種による分類」を表形式で表現した例

公開記事データベース

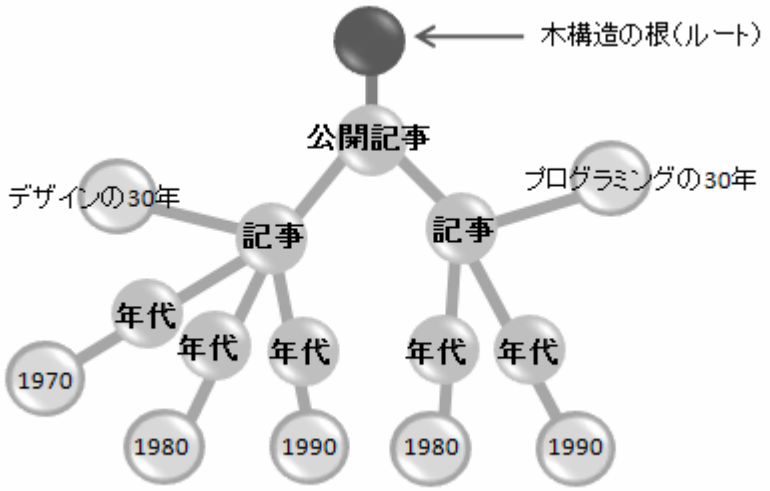
ページID	記事	年代
D1970	デザインの30年	1970
D1980	デザインの30年	1980
D1990	デザインの30年	1990
P1980	プログラミングの30年	1980
P1990	プログラミングの30年	1990

デザイナ対象のオンライン記事

プログラマ対象のオンライン記事

●図 1-24　●図 1-23 の「職種による分類」を木構造で表現した例

●表 1-1 を忠実に置き換えるのであれば，各「年代」から「ページ ID」の枝を伸ばすことは可能であるが，木構造では，各「年代」の記事自体が一意であるため，ここでは省略している。



※5：●図 1-23～●図 1-28 では，「記事」の中に，「デザインの 30 年」というデータと，「年代」がある構造になっているが，これはあくまで説明用の図を複雑化しないためであり，実務では，さらに細分化したほうがよい（第 3 章，リスト 3-1 等を参照）。

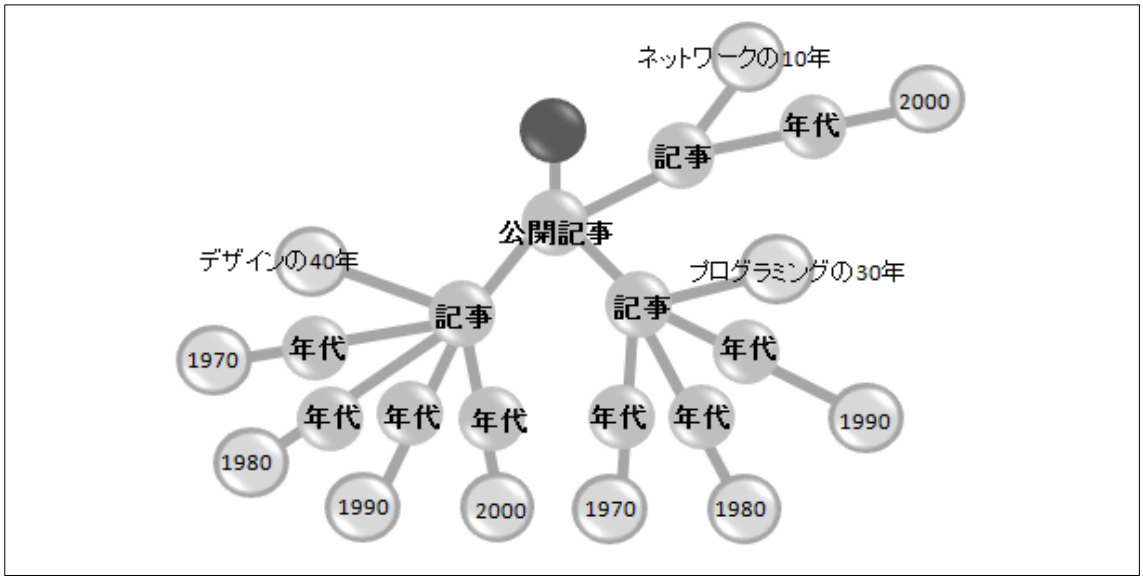
柔軟な木構造

先に述べたように，木のノードや枝は，複製したり，連結したり，構造を変更したり，削除することができる。木構造は変化を吸収しやすいモデルである。ここでいう変化とは，記事が追加されたとか，別サーバを立てたといったヒトの外で起こる変化や，ヒトの内で起こる意識のフォーカスの変化などである。

もし，別の記事が追加されたならば，●図 1-25 のように，ノードごと接ぎ木すればよい。

2010 年まで公開を続け，2000 年代の記事が追加されたら，それに合わせてタイトルも「デザインの 40 年」に変更すればよい。逆に，さかのぼって 1970 年代のプログラミングについての記事が追加されたならば，これもまた枝を追加すればよい。さらに，「ネットワークの 10 年」といった別の記事を追加する必要が生じた場合も，接ぎ木で対応すればよい。

●図 1-25　●図 1-24 の木に，枝を接ぎ木してタイトルのデータを変更した例



さらに，●図 1-23 中の「デザインの 30 年　1970 年代」の記事内容が膨らみ，字数の関係から章や節に分割する必要性が生じた場合を想定してみよう。

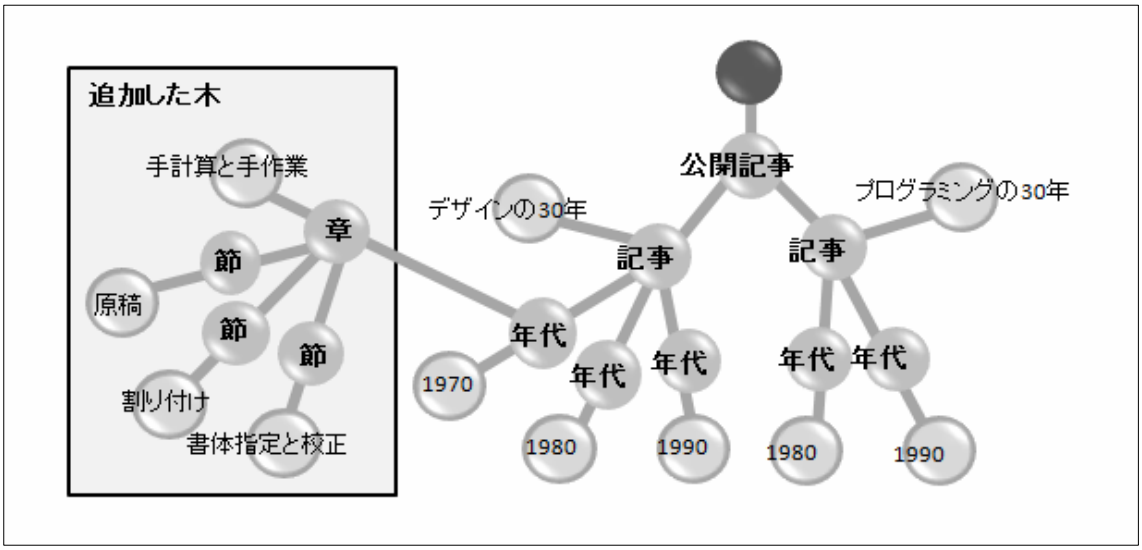
表形式ならば，●図 1-26 のように，空のセルが生じるが，木構造では，●図 1-27 のように，末尾に接ぎ木するだけで済む。

●図 1-26　●図 1-23 にフィールドを追加した例

回顧録テーブル

ページID	記事	年代	章	節
D1970P1	デザインの30年	1970	手計算と手作業	原稿
D1970P2	デザインの30年	1970	手計算と手作業	割り付け
D1970P3	デザインの30年	1970	手計算と手作業	書体指定と校正
D1980P1	デザインの30年	1980		
D1990P1	デザインの30年	1990	データがない	
P1980P1	プログラミングの30年	1980		
P1990P1	プログラミングの30年	1990		

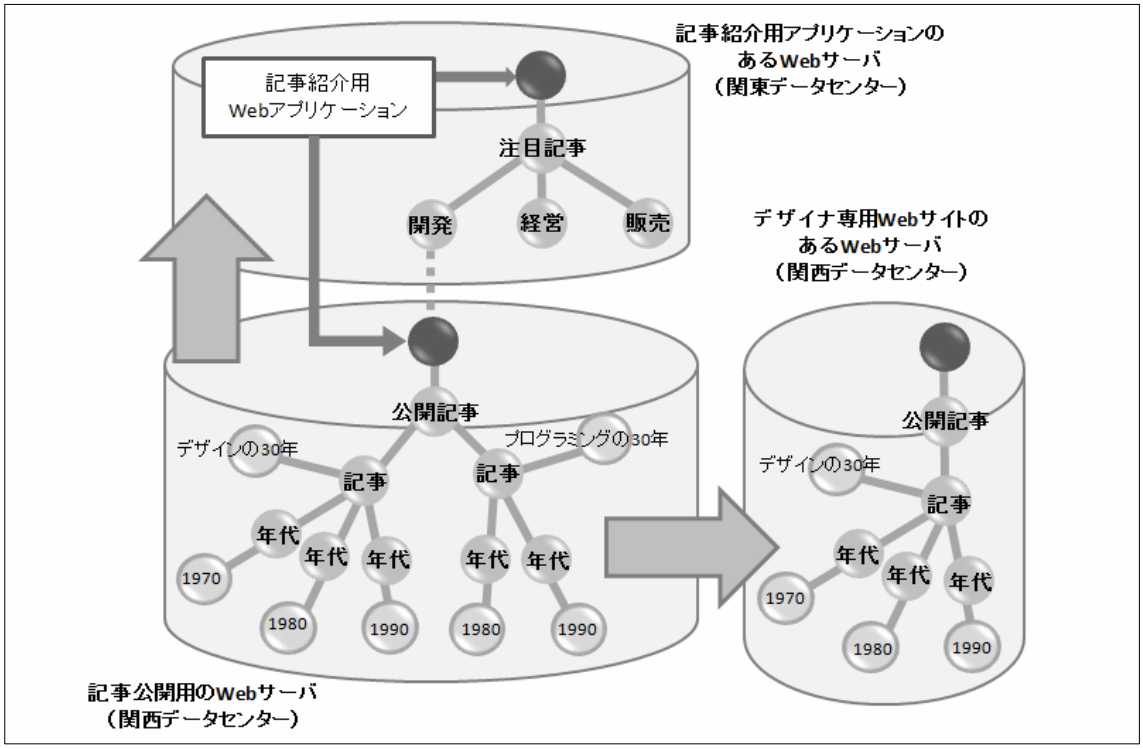
●図 1-27　●図 1-26 を木構造で表現した例



木は，枝葉を伸ばして繁ったり，枯れ枝を落としたり，落ち葉を散らす。データが増えて生い茂ることもあれば，根を残して切り倒されることもある。●図 1-28 のように，●図 1-24 の木が存在する Web サーバとは地理的に離れた，異なる Web サーバにある木と連携して，生い茂ることもある。逆に，デザイナ専用 Web サイトに掲載するために，デザイナ対象記事以外の枝を剪定することもできる。

●図 1-28 の例は，1990 年代といった 10 年分が 1 個のノードとなっているが，さらに記事の内容が充実してきた場合，2000 年代前半／2000 年代後半，あるいは 2001 年，2002 年，2003 年…というように細分化して，複数の Web サーバに分散させることも可能である。

●図 1-28　木は，利用される時間と空間に応じる形で，柔軟に変化する



このように，木構造は，変化を受容する柔軟なモデルであり，情報の BREAD（Browse, Read, Edit, Add, Delete）※6 も容易である。この例のような数件のデータではなく，大量の既存データが既に登録されている場合は，構造を 1 から変更することは難しい。そこで，柔軟性を持つ木構造を採用することがある。

これまで掲載した各図から分かるように，我々は，木の各ノードをたどることにより，シーケンスの表わす意味を理解することができる。木構造には，コンピュータが計算できる「構造」だけでなく，ヒトがコンテキストとして理解できるシーケンスがある。ヒトによる理解を，コンピュータによる計算と同等か，それ以上に重視して，情報を表現することができる。従来 RDB で表現されていた情報の一部を，木構造に置き換えるケースが増えているのは，木が，ヒトの理解しうるシーケンスを表現でき，変化に柔軟であるからに他ならない。

※6：ソフトウェアによる処理では，CRUD（Create，Read，Update，Delete）が一般的に用いられるが，木構造データを Web アプリケーションでブラウザ上に表示することも多々あるから，ここでは BREAD を用いている。

1.4 木構造と XML

1.4.1 データとメタデータ

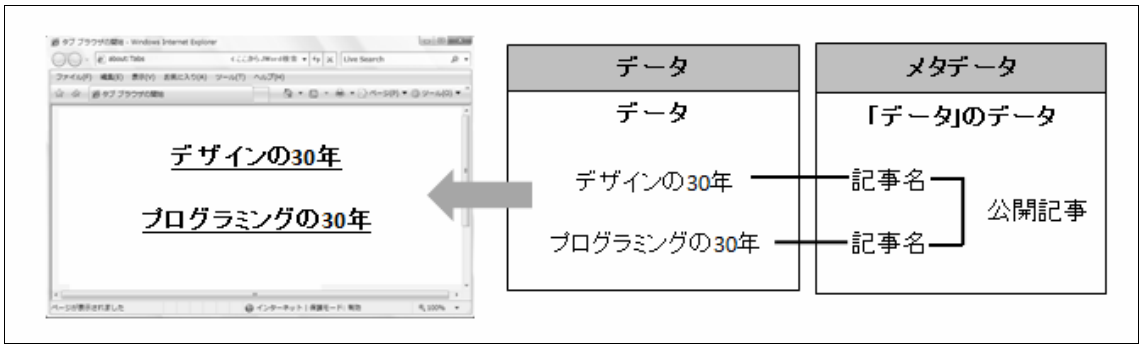
これまで長々と木構造について述べてきた。では，木構造と本書のテーマである XML との間には，どのような関係があるのか？単純な 2 件のデータを例にとって説明しよう。

いま，●図 1-29 のように，Web ブラウザ上に，「デザインの 30 年」「プログラミングの 30 年」という文字列が表示されている。ブラウザ画面上の文字列だけを見れば，それが書籍のタイトルなのか，ドキュメンタリー作品の DVD のタイトルなのか，社内報のコラムなのかは分からない。そこで，「デザインの 30 年」「プログラミングの 30 年」は「記事名」であり，それらはいずれも「公開記事」である，ということを示すことができたら，どうだろうか？ヒトはデータの意味を理解し，コンピュータもまた，それらのデータを「公開記事の中の，記事名」として処理することができるようになるに違いない。

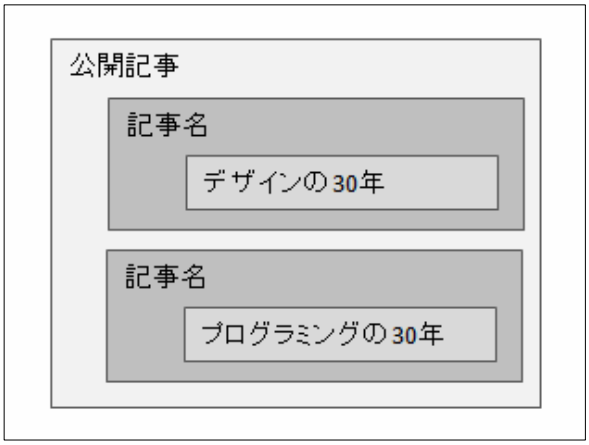
この「記事名」および「公開記事」のような，データのためのデータを「メタデータ」という。
「メタ」という言葉は木構造特有のものではなく，一般用語であり，（（A）の A）を，「メタ A」という。たとえば，既存の哲学について哲学することを「メタ哲学」，学習の仕方を学習することを「メタ学習」などという。データにメタデータを付加すると，データは単なるテキストデータではなく，メタデータで表現される意味を持つようになる。RDB でいえば，メタデータの名前は，フィールド名に相当する。

●図 1-29 の，「公開記事」の中に「記事名」が 2 個あり，それぞれのデータが「デザインの 30 年」「プログラミングの 30 年」であるという構造を図で表すと，●図 1-30 のような入れ子になる。

●図 1-29　データとメタデータ。



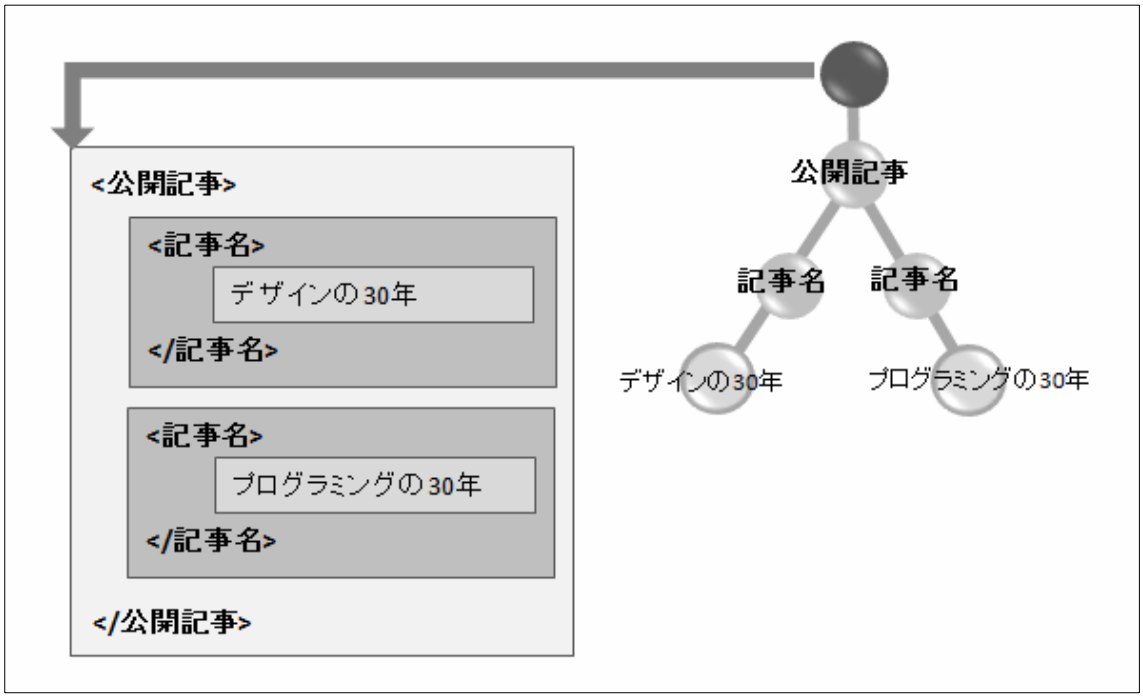
●図 1-30　●図 1-29 の階層構造は入れ子になっている。



この中のデータとメタデータを区別するために，メタデータの始まりを<メタデータの名前>，メタデータの終わりを</メタデータの名前>のように，<（大なり記号）と>（小なり記号）で挟むと，●図 1-31 のようになる。

これは，<公開記事>をたどると<記事名>があり，さらに<記事名>のデータがあるという「シーケンス」であるから，もちろん木で表現することができる。木は，テキストデータと，データの意味を表現するメタデータを，ノードとする構造になる。

●図 1-31　●図 1-30 のメタデータを<と>で挟んでデータと区別した例と，その木構造。



1.4.2 XML によるメタデータの表現

●図 1-31 は，メタデータおよびデータの構造を的確に表現している。しかしながら，この図で表現された構造を，特定のアプリケーション固有の形式で記述したのでは，データの長期蓄積や幅広い活用を見込むことはできない。将来にわたって，読み込みも編集も保存もできる可能性が高いのは，テキスト形式である。少なくともヒトが、思考やコミュニケーションの道具として言語を使い続ける限り、テキスト形式はなくならない。いかにハードウェアや OS や開発フレームワークが進化したとしても、テキスト形式のファイルとして蓄積している限り，ヒトがデータの内容を理解することは可能であると考えられる。もちろん，テキスト形式で保存しておけば，コンピュータによる計算も可能である。

そこで，この図をテキストとして表現すると，リスト 1-1 のようになる。つまり，前掲●図 1-31 の木構造をテキストで記述すると，リスト 1-1（次ページ）のようになるのである。

リスト 1-1 のような方法を使えば，これまで説明に用いてきた木構造の図も，すべて同じように，テキストで記述することができるようになる。この世界の情報は，木モデルで表現するだけではなく，テキストで記述することができるのである。

リスト 1-1　●図 1-31 をテキストに置き換えた。

```
<公開記事>
    <記事名>デザインの 30 年</記事名>
    <記事名>プログラミングの 30 年</記事名>
</公開記事>
```

そのような，木構造の文書のあり方と，解析用プログラムの動作について規定した標準化仕様が，本書のテーマである「XML（Extensible Markup Language，拡張可能なマーク付け言語）」である。XML は，木モデルを，テキストで表現しうる現在最良の仕様である。当然のことながら，XML は，木構造の特長と利点を持つ仕様である。

<と>はデータに付けた付箋のようなもので，タグと呼ぶ。タグの名前（タグ名）はメタデータの名前であり，データの意味を表している。このタグを付加する作業（タグ付け）のための規則は，XML の仕様書で定められている。ただし，XML の仕様書では，分類方法やタグ名まで定められているわけではない。ヒトの判断はさまざまであるから，リスト 1-1 のようなタグ付けではなく，リスト 1-2 のようなタグ付けも考えられる。

リスト 1-2　リスト 1-1 の構造はそのままで異なるタグ名を付けた例。

```
<記事リスト>
    <記事タイトル>デザインの 30 年</記事タイトル>
    <記事タイトル>プログラミングの 30 年</記事タイトル>
</記事リスト>
```

ﾘｽﾄ 1-1 とﾘｽﾄ 1-2 を比較すると，ﾋﾄであれば，タグ名は異なれど，タグ付けの「意図」は同じであると理解することができる。しかしながら，コンピュータに，ﾋﾄと同じ理解をもとめることは，現段階では難しい。ﾘｽﾄ 1-1 とﾘｽﾄ 1-2 のいずれかをﾋﾄに示して，「どのような記事があるか」と問えば，十中八九同じ答えが返ってくる。だが，コンピュータには，たとえば，「文書の中から，<記事名>という名前のタグの付いたデータを抽出せよ」「文書の中から，<記事ﾀｲﾄﾙ>という名前のタグの付いたデータを抽出せよ」といった，曖昧さを排した命令を与えなければ，正しい結果を得ることはできない。タグ名をコンピュータによる計算のﾄﾘｶとして用いるには，ﾒﾀデータの意味が同じであるならば，同じタグ名を付けるべきである。そこで，ﾘｽﾄ 1-1 やﾘｽﾄ 1-2 だけでなく，他に，適切なタグ付けがないかを検討し，いずれか一種類のタグを決定する作業が必要となる。

このような，情報を分類して木の階層構造を決定し，タグ名やデータのあり方を決める一連の作業が，「XML 設計」である。良い設計にもとづいて作成された XML 文書は，ﾋﾄにとって理解しやすく，コンピュータにとっても処理しやすいものとなる。再利用困難な文書ではなく，蓄積して価値のある文書となる。

前述のように，この世界の多くの情報は木構造で表現できる。そして，新しい情報が今この時も大量に生成され，サーバや記録ﾒﾃﾞｨｱ等に蓄積されつつある。良い XML 設計は急務となっている。

とはいえ，対象データの意味を理解しなければ，設計作業はできても，良い設計はできない。別掲コラム「ｲﾒｰｼﾞを情報とする場合の木構造」で触れたように，対象の持つ意味を判断するための前提条件となる知識や経験が異なれば，設計した構造も異なる結果になる。

業務系の XML 設計者には会計についての知識が必要であるし，医療カルテ用 XML の設計者には医療の現場が見えていなければならない。とはいっても，XML 設計者が多数の業界の知識と経験を得ることは困難であるから，2〜3 の得意な業種の知識と経験を得た上で，設計を行うことが望ましい。逆にいえば，何らかの専門職に就いている人が，XML 設計をできるようになることが，本来は望ましい。XML しか知らないエンジニアが医療用の XML を設計するよりも，医師が XML を学んで設計する方が，社会に役立つ構造を考えられるであろう。実際，各業界団体では，その業界の当事者によって，XML ベースの仕様が続々と策定されつつある。

本章に掲載したﾘｽﾄを見ると，XML 設計は簡単だという印象を持つかもしれない。ところが，XML という仕様自体はシンプルなのだが，良い設計となると非常に奥が深いのである。

では，良い設計を行うための心得は？それを語るには，XML の世界の言葉が必要になる。まずは次章で，XML の基礎知識に触れておく。既に知識のある読者は，第 2 章を読み飛ばして差し支えない。

Column「世界を記述する言語」

この世界の現象は重なり合わないので，現象同士の間には何らかのシーケンスがある。我々が知覚する現象なら何でも，木構造で表現することができるように思われる。木構造は万能のすばらしいモデルのように思われさえする。

では，木構造で表現できないことなど，あるのだろうか？

木構造では，我々が情報のセットとして認識し難いものを表現することは難しい。たとえば，A 子さんの書いた記事の「Windows7 の特長」と，B 子さんのブログの「朝顔の生育記録種」は，A 子さんと B 子さんに何の接点もなければ，木構造で表現することは難しいだろう。「この世界の現象」という情報のセットには違いないし，「○年○月○日」のヒトの行動という意味では，時間経過というシーケンスがある。表現することはできるが，表現することに意味がない，といったところであろうか。

もうひとつ，時間や空間のない世界の「コト」は，表現できない。時間の流れや空間の拡がりを観ることができない世界では，シーケンスの捉えようがない。

想像してみてほしい。時間の流れも，空間の広がりもない世界である。それは，「世界」という言葉で語るしかないから，そう書いているだけで，我々の言葉で語りうる「世界（宇宙）」ではない。一瞬が永遠であり永遠が一瞬であり，一点が偏在であり拡がりが一点であるような，（世界としか言いようがないのだけれども，かといって），世界ではない状態である。

一瞬と永遠が同じだなんて，それは矛盾している，と思われるかもしれない。いや，我々の住むこの世界の言葉で表現するから矛盾しているのであって，我々の言葉は，万能ではない。言葉では，語りえないことがある。言葉が有効に機能するのは，最大限見積もっても，我々が，働き，遊び，飲み食らう，この世界に対してである。

そのような時間の流れも，空間の拡がりもない世界などあるはずがない，と思われるかもしれない。この世界には始まりがあり，始まりがあるということは，その前があったはずだ，という人もいる。この世界の外に何かがあるなら，そのまた外には何かがあるはずだ，と。いや，始まり，終わり，前や後，内や外，先にあった，外にある，というのは，時空という概念があってこそ成立するのであり，その時空がない状態を，我々の時空ありきの感覚で使っている言葉で表現することなどできない。

木構造は，我々の暮らすこの世界に限るのでなければ，無力である。

そう考えると，逆に，流れを表現できる木構造とは，きわめて現世的であって，この我々の住むこの世界，流れに変化を見いだす世界を表現するには，うってつけのモデルであると思えてくるではないか！

第 2 章　XML の基礎

本章では，XML 文書の構造と基本的な用語について学ぶ。

本文中の () 内には，仕様書で使われている英単語を併記している。
本章と次章で取り上げることは，最低限知っておくべきことにすぎないので，
さらに仕様書から情報を得たい場合には，() 内の英単語で仕様書内を検索するとよい。

2.1 XML 文書の構造

2.1.1 XML の概要

XML (Extensible Markup Language) とは，「XML 文書 (XML documents)」と呼ばれる処理対象データのあり方と，この文書を解析処理するプログラムの動作を規定する仕様である。

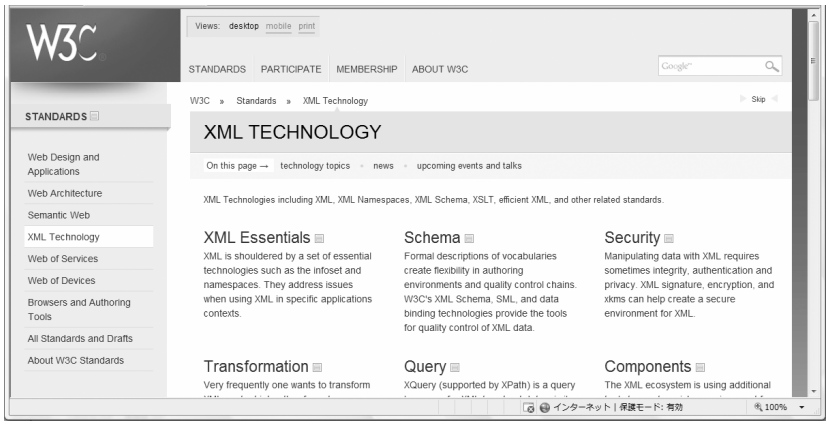
ここでいうプログラムとは，XML プロセッサまたは XML パーサと呼ばれるもので，XML 文書の構造や構文を解析して，内容や構造にアクセスするソフトウェアモジュールである。これは，.NET Framework のようなフレームワークやブラウザ等に搭載されていることがある。たとえば Microsoft Internet Explorer のユーザであれば，Microsoft XML Parser (略称 MSXML)を，それがパーサであるとは気付かないまま，日常的に使っていることになる。

XML 文書自体は，前章で述べたとおり，単純なテキストファイルであるので，テキストエディタや，Microsoft Office，Microsoft InfoPath といった，XML に対応したビジネスソフトなどで，きわめて容易に作成することができる。

XML を策定しているのは，W3C (World Wide Web Consorium，http://www.w3.org/) という機関である。現在，仕様書は， W3C の XML のトップページ (http://www.w3.org/XML/) 中の「XML Technology＞XML Essentials＞XML」コーナから入手することができる (●図 2-1)。また，XML1.0 1998 年 2 月 10 日勧告版の日本語訳は，「Translations」コーナのリンク先から入手することができる。

W3C での策定手順は，おおむね，Working Draft→Last Call→Candidate Recommendation→Proposed Recommendation→Recommendation (勧告) のように進む。1998 年 2 月 10 日に，XML バージョン 1.0 が勧告となり，10 年を過ぎた 2008 年時点では，XML1.0 Fourth Edition，XML 1.1 Second Edition が勧告となっている。ただし，XML 1.1 仕様書の中で，XML プロセッサは，XML1.0 に対する後方互換性を持つことが必須であると既定されている (※1)。XML1.0 こそが，XML を理解する上での基本であるので，本書では，XML1.0 Fourth Edition (http://www.w3.org/TR/2006/REC-xml-20060816/) にしたがって解説を進める (本書執筆後の 2008 年 11 月 26 日には Fifth Edition が発表されている)。

●図 2-1 W3C の XML ページ。仕様書は「XML Technology＞XML Essentials＞XML」コーナから入手することができる。



※1：XML 1.1 processors MUST be able to process both XML 1.0 and XML 1.1 documents. Programs which generate XML SHOULD generate XML 1.0, unless one of the specific features of XML 1.1 is required.（W3C XML 1.1 Second Edition，5 Conformance，5.1 Validating and Non-Validating Processors）

XML 1.1 プロセッサは，XML 1.0 と XML 1.1 両方の文書を処理できなければならない。XML を生成するプログラムは，XML1.1 が不可欠とされる特別なケースを除き，XML1.0 を生成する方がよい。

2.1.2 文書構造

XML 文書（XML documents）は，物理構造（physical structures）と論理構造（logical structures）の両面からとらえることができる。

物理構造の面から見れば，XML 文書は，実体（entities）という記憶単位から成る。XML プロセッサは，名前を持たない文書実体（document entity）である木構造のルート（root）から処理を開始するようになる。

実体には，XML プロセッサによって構文解析されるデータ（parsed data）と，構文解析されないデータ（unparsed data）がある。構文解析されるデータとは，マーク付け（markup）と文字データ（character data）である。つまり，メタデータおよびデータは XML プロセッサによる解析の対象となる。

物理構造が XML プロセッサによる計算可能な構造であるのに対し，論理構造とはヒトによる理解が関わる文法であると捉えても，あながち間違いではない。論理構造の面から見れば，XML 文書は，宣言・要素・コメント・文字参照及び処理命令（以上後述）を，マーク付けしたものによって構成される。

2.1.3 整形式文書と妥当な XML 文書

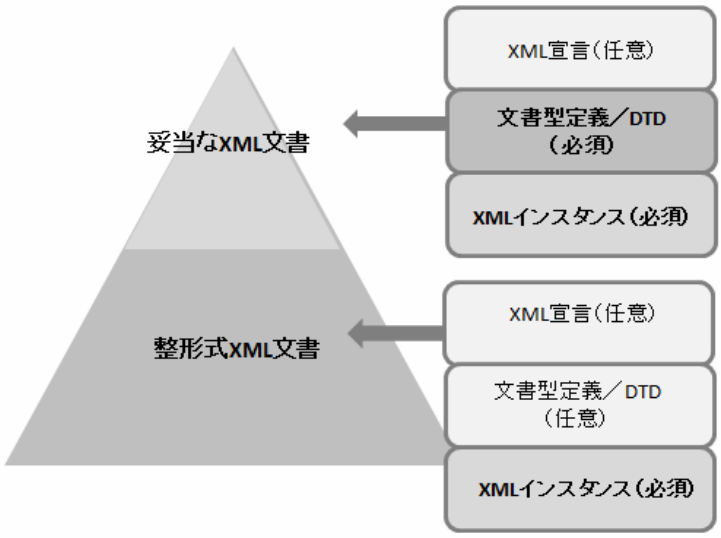
XML1.0 仕様に則って記述された XML 文書は，整形式（well-formed）XML 文書と呼ばれる。

しかしながら，必ずしも設計と寸分違わぬ XML 文書が作成されるとは限らない。実務においては，不特定多数のデータや，既存の大量のデータを XML 化する必要性から，設計者と XML 文書作成者が異なることは多く，入力ミスの可能性を否定できない。そこで，構造やデータ等の正確性を期すために，XML 設計に則っているかどうかを，設計書と照らし合わせて検証することがある。

XML には，設計者が論理構造についての制約条件を定義したり，定義した実体を使うための「文書型宣言」という機構がある。XML1.0 仕様に則った上に，このような制約条件を満たす XML 文書を，妥当な（valid）XML 文書と呼ぶ。基本的に，妥当な XML 文書とは，文書型宣言をもち，その文書型宣言に示す制約条件を満たす文書である。

両者は，●図 2-2 のように，妥当な XML 文書が，整形式 XML 文書の上に成り立つという関係である。つまり，妥当な XML 文書は，必ず整形式 XML 文書である。妥当な XML 文書は整形式 XML 文書でもあるのだが，実務においては一般的に「妥当な XML 文書」と呼び，「整形式であるうえに妥当な」といった言い方はしない。また，単純に「整形式 XML 文書」といえば，図のピラミッドの 1 段目にあたる文書を意味する。本書においては，「XML 文書」といえば，この一般的な意味での整形式 XML 文書を指すものとして以降を読んでほしい。

●図 2-2 妥当な XML 文書は，
整形式 XML 文書の上に成り立つ



XML 文書は，XML 宣言，文書型定義（DTD），XML インスタンスから成る。DTD は Document Type Definition の略で，XML のような構造化文書における論理構造の制約条件を定義するための，最古の仕様である。

整形式 XML 文書では，検証を必ずしも必要としないため，文書型定義は任意である。つまり，整形式 XML 文書は，XML1.0 仕様の整形式制約に従ってはいるが，文書型定義で定義された設計条件を満たす文書であるとは限らないのである。

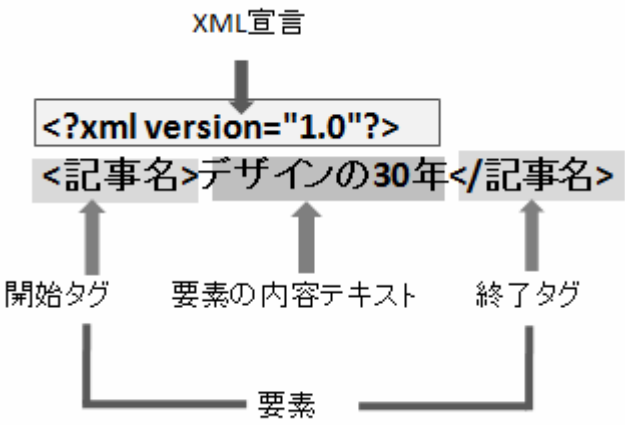
一方，妥当な XML 文書では，設計条件を満たす必要がある。もっとも，XML の構造を定義する仕様は，いまや DTD だけではない。XML Schema，Relax といった仕様も使うことができる。XML 1.0 が勧告となった 1998 年時点では，XML Schema は勧告になっておらず，Relax も存在しなかった。そのため，妥当な XML 文書の定義には，一般的に DTD が使われていた。現在では，XML Schema や Relax が普及しているので，DTD に関する知識は，過去の XML 文書のメンテナンスのために必要という程度ではないかと推測する（筆者も XML 業界の事情を熟知しているわけではないので，あくまで推測である。ただし，筆者の耳にする限り，現在進行形の案件で DTD が採用されたという話は聞いたことがない）。

2.1.4 整形式 XML 文書

整形式 XML 文書の構成

整形式 XML 文書は，prolog，element，0 個以上の misc（その他諸々のもの）から成る。prolog は，XML のバージョン情報と文字エンコードを含む XML 宣言（XML declaration），文書型宣言（document type declarations）などの前書きである。Element は要素，misc は，コメント・処理命令・空白文字（whitespace）で構成される。整形式 XML 文書の最も基本的な構成は，XML 宣言と要素である（●図 2-3）。

●図 2-3 整形式 XML 文書の基本的な構成



XML 宣言

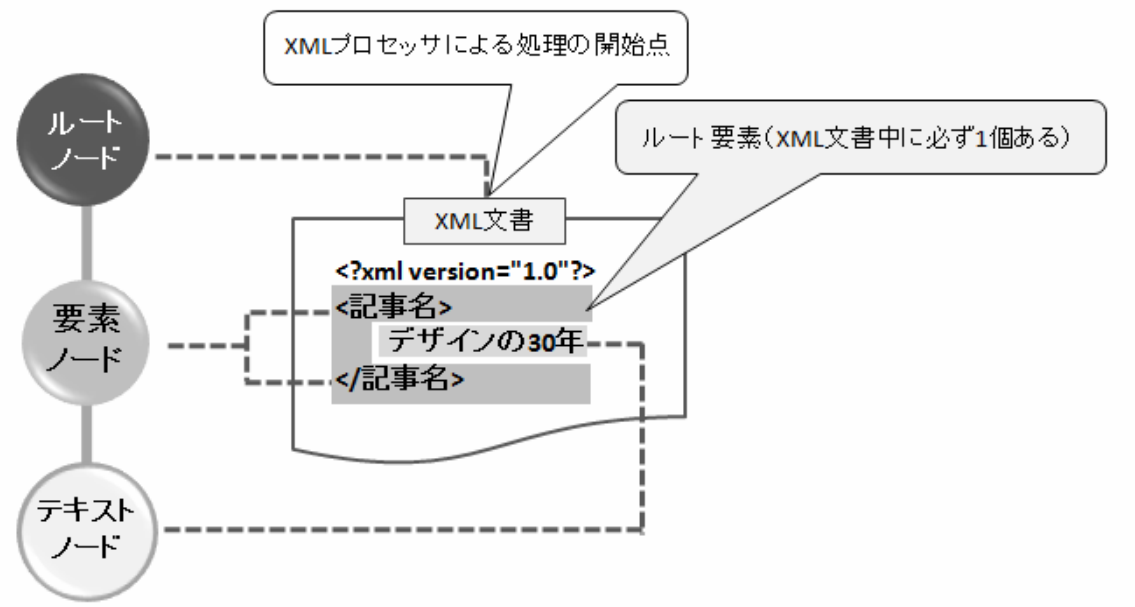
整形式 XML 文書のバージョンを示す XML 宣言は，あってもなくてもよいとされているが，記述する方がよい。XML1.1 (Second Edition) では，XML のバージョンを特定するために，<?xml version="1.1"?>という宣言が MUST 事項になっており，XML 宣言を記述しない場合は，XML1.0 と見なされる。バージョン情報を省略すると，XML 文書を一見しただけでは，XML1.0 の意図的な省略か，XML1.1 の後方互換目的かが判断しにくい。したがって，バージョンは記述しておいた方がよい。

要素

XML 文書には，ルート (root) あるいは文書要素 (document element) と呼ばれる，ただ 1 個の要素が必須である。つまり，XML 文書の最小構成は 1 個の要素である。

開発実務では，この文書要素であるルートを「ルート要素」と呼び，前述の物理構造としての木構造のルートであって，XML プロセッサによる処理の開始点を「ルートノード」と呼んで，混同を避ける (「ノード」については，次章「ノードの扱いと順序」で解説)。前掲●図 2-3 の XML 文書では，<記事名>要素はルート要素である (●図 2-4)。

●図 2-4 ルートノードとルート要素



要素は，XML 宣言の後に記述する。●図 2-3 のように，要素とは，開始タグ（start-tags）と終了タグ（end-tags）によって区切られたものである。終了タグは必須で，対応する開始タグの要素型と同じ名前を持つ。要素型は名前（要素名）によって特定され，大文字と小文字は区別されるので，開始タグ名と終了タグ名は，完全に同じでなくてはならない。たとえば，<Book></book>は誤りである。

要素の開始タグと終了タグとの間のテキストを，「要素の内容」と呼ぶ。リストでは，<記事名>要素の内容テキストは「デザインの 30 年」であるということになる。要素の内容には，マーク付けの開始区切り子<>を使ってはならない。

なお，本書では，解説文中の構文とタグの混同を避ける目的で，英字タグではなく日本語タグを使っているが，日本語にする必然性がないのであれば，英字タグの方がよい。また，<記事名>は仕様上の呼び名でいえば「開始タグ」にあたるが，日本語の要素名を解説文と区別する目的で，便宜上「<記事名>要素」という表記としている。

XML は木構造であるから，要素は，さらに階層化することができる。次の●リスト 2-1 には<公開記事><オンライン記事><記事名>という各要素がある。このように要素は正しく入れ子（nest）になっていなければならない。<公開記事><記事名>デザインの 30 年</公開記事>というように，正しい入れ子になっていないものは誤りである。

●リスト 2-1　正しい入れ子になっている XML 文書の例

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名>デザインの 30 年</記事名>
  </オンライン記事>
</公開記事>
```


さらに，次の●リスト 2-2 のように，要素内に，要素に混在する文字データが含まれる可能性があるものを「混合内容を持つ」という。ただし，混合内容は，実務では再利用困難な事態を引き起こすことがままあるのでお勧めしない。

●リスト 2-2　混合内容を持つ XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>ブログで公開開始
    <記事名>デザインの 30 年</記事名>
  </オンライン記事>
</公開記事>
```

必ずしも内容テキストのない要素もある。次の●リスト 2-3 の<公開日>要素は，空要素である。これは，<公開日 />と書いても同じである。

●リスト 2-3　空要素のある XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名>デザインの 30 年</記事名>
    <公開日></公開日>
  </オンライン記事>
</公開記事>
```

属性

要素は，名前（attribute name）と値(attribute value，AttValue)から成る属性（attribute）を持つことができる。●リスト 2-4 の<公開日>要素を，<記事名>要素に対する"公開日"属性として従属させるには，●リスト 2-5 のように記述する。この属性の名前は「公開日」であり，その値は「2008-07-07」である。

●リスト 2-4　要素のみの XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名>デザインの 30 年</記事名>
    <公開日>2008-07-07</公開日>
  </オンライン記事>
</公開記事>
```

●リスト 2-5　属性のある XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名 公開日="2008-07-07">デザインの 30 年</記事名>
  </オンライン記事>
</公開記事>
```

開始タグあるいは空要素タグ（empty-element tags）には，同じ属性名を二回以上使ってはならない。たとえば，<記事名 公開日="2008-07-07" 公開日="毎週月曜日">デザインの 30 年</記事名>では，同じ名前の属性"公開日"が重複している。これは誤りである。

コメント

XML 文書中に記述するデータ以外のテキストとして，コメントや埋め込みスクリプトなどがある。コメント（comments）は，XML 文書中では，HTML などと同様，<!-- コメント文 -->のように記述する。コメントの文字列として，"--"（二重ハイフォン）は使ってはならない。

処理命令

XML 文書中に，アプリケーションへの命令を記述することもできる。この処理命令（processing instructions）は，<?で始まり，?>で終わる。たとえば，XML 文書の構造を変えたいとき，変換言語である XSLT を使って記述した構造変換用のファイルに対して，参照して実行するよう記述するといった使い方ができる（XSLT については，次章で解説）。

CDATA セクション

スクリプトのようにマーク付けではないものは，要素の中の任意の階層に，CDATA セクション（CDATA section）を設けて記述する。CDATA セクションは，<![CDATA[で始まり，]]>で終わる。この中に書いた文字は，マーク付けとして認識されない。

マーク付け

XML 文書における「マーク付け」の定義は，開始タグ，終了タグ，空要素タグ，実体参照，文字参照，コメント，CDATA セクションの区切りとなるデリミタ，文書型宣言，処理命令，XML 宣言，テキスト宣言，文書実体のトップレベルにある空白文字の形であると既定されている。このうち整形式 XML 文書を扱う実務では，XML 宣言，開始タグ，終了タグのマーク付け作業が，「タグ付け」と呼ばれる。つまり，前章でも述べたように，タグ付けとは，データにメタデータをマーク付けする作業であると考えておいてよい。

2.1.5 使用可能な文字および文字コード

要素名

XML 文書の解析対象データとして使える文字は，タブ（tab），改行（carriage return），復帰（line feed），Unicode 及び ISO/IEC 10646 で規定される図形文字（legal characters）である。タグの名前は，アルファベットや漢字や全角ひらがな，全角カタカナなどの字（letter）で始まり，数字，ハイフォン，下線，コロンやピリオドを使って付けることができる（●表 2-1）。これらを名前文字（name characters）と呼び，名前文字の列を，名前トークン（Nmtoken）という。ただし，X や x，M や m，L や l は予約されており，タグ名を xml などの文字列で始めることはできない。

コロンは，「XML プロセッサが受け付けなければならない」と仕様書で規定されてはいるが，これは XML 関連仕様である「名前空間（後述）」で使われるものである。

●表 2-1 XML の名前文字の一例

使ってもよい例	
<型番>	
<ふりがな>	
<メニュー>	
<ISBN>	
使ってはいけない例	
タグ	**理由**
<2008年>	数字から始まっている
<first name>	スペースがある
<XmlRoot>	予約語が使われている
<ﾙﾋﾞ>	半角カタカナである

特殊文字と空白文字

要素の内容に次の5つの記号文字を使いたい場合は，●表2-2のように置き換えて使うことができる。これらは，XMLプロセッサが認識を義務付けられている定義済み実体(predefined entities)である。

●表2-2 定義済み実体

記号	文字参照
lt	<
gt	>
amp	&
apos	'
quot	"

たとえば"&"を，文字データ内に出現する&として扱うには，●リスト2-6のように記述する。あるいは，文字列"&"を使ってもよい。XMLプロセッサの機能を搭載したブラウザでの表示は，●図2-5のようになる。

●リスト2-6 定義済み実体の使用例

```
<?xml version="1.0"?>
<記事名>デザインの30年&#38;プログラミングの30年</記事名>
```

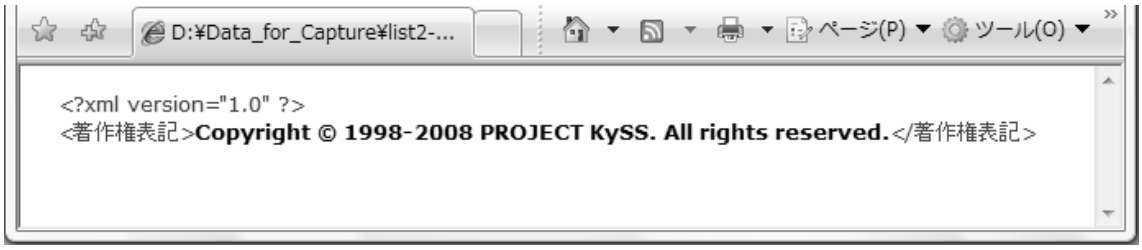
●図2-5 定義済み実体を使用した結果。IEによる表示。

また，キーボード等から直接入力不可能な文字を使いたい場合，文字参照(character references)により，ISO/IEC 10646 文字集合の特定の文字を参照することができる。文字参照が「&#x」で始まれば，終端の「;」までの数字および字は，ISO/IEC 10646の文字コード位置の16進数表現を使う。●リスト2-7は使用頻度の高い文字参照の例である。使用結果は●図2-6のようになる。

●リスト2-7 文字参照の使用例

```
<?xml version="1.0"?>
<著作権表記>Copyright &#xA9; 1998-2008 PROJECT KySS. All rights reserved.</著作権表記>
```

●図2-6 文字参照を使用した結果



空白文字の扱いについても規定されている。XML 文書のデータが複数行にまたがる場合，CARRIAGE RETURN (#xD) および LINE FEED (#xA)の任意の組み合わせによる改行を考えるかもしれない。だが，XML 仕様（2.11 End-of-Line Handling）では，XML プロセッサには，#xD #xA の連続文字と，単独の#xD を，単一の#xA に変換することにより，解析前に改行を正規化することが義務付けられている。その結果，1 個の空白が空くだけになる。

XML プロセッサには，属性値の正規化も義務付けられている（3.3.3 Attribute-Value Normalization）。連続する空白文字を，1 つの空白文字（#x20）に正規化すること，先頭と末尾の空白文字をすべて取り除くこと，である。

たとえば，●リスト 2-8 のように，「PROJECT」と「KySS」の間に複数の空白を入れたとしても，1 個の空白に正規化される。また，<Web サイト>要素の内容テキストである「デザインの 30 年」の後に，空白を続けて入れたとしても，取り除かれる。

視覚詩のように，文字の位置が重大な意味を持つ文書については，テキストの中に空白や改行があると捉えるのではなく，文字のレイアウト座標を指定するという考えかたで，Silverlight や Flash を用いて表現する方が賢明である。

●リスト 2-8　要素の内容に複数の空白を使った例

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名 サイト名="PROJECT&#xD;&#xA;&#xA;&#xA;KySS">
      デザインの 30 年
    </記事名>
  </オンライン記事>
</公開記事>
```

属性値

属性値のような，引用符で囲まれた文字列のことをリテラルデータ（literal data）と呼ぶ。先に挙げた，<記事名 公開日="2008-07-07">デザインの 30 年</記事名>でいえば，「2008-07-07」が，これにあたる。

属性値内には，マーク付け記号である < （大なり記号）を含んではならない。必要であれば<で置き換える。また，もし属性値が一重引用符及び二重引用符の両方を含むのであれば，アポストロフィあるいは一重引用符(')は"'"として表現し，二重引用符(")は，"""として表現する。

文字コード

作成した XML 文書は，拡張子.xml を付けてファイルとして保存し（※2），蓄積することができる（本書では，これを XML 文書ファイルと呼ぶ）。XML 文書ファイルの文字コードは UTF-8 で作成した方がよいだろう。本書でも UTF-8 を基本として解説を進める。仕様書では，XML プロセッサは，UTF-8 符号化方式及び UTF-16 符号化方式の文字コードに対応しなければならないことになっている。先に挙げた

XML 宣言のように，文字コードを指定していなければ，XML 文書は Unicode であると見なされる。テキストエディタで XML 文書を作成する際には，保存時に，宣言した文字コードを選択する必要がある。

専用のタグ付けツールで XML 文書を作成する場合は，デフォルトの文字コードを確認しておく必要がある。XML 文書ファイルを生成して出力するプログラムを開発する際には，XML 文書を出力する際の，開発フレームワークにおけるデフォルトの文字コードを確認しておく必要がある。デフォルト値によっては，プログラムの中で出力する際の文字コードを指定する必要がある。

XML 設計時には，最初に文字コードを決めてから構造設計に着手すべきである。

※2 XML で記述された文書のファイルの拡張子が必ずしも.xml であるとは限らない。たとえば，ブログの更新案内などを配信する RSS ファイルは XML 文書の一種であるが，拡張子は.rss である。

2.1.6 妥当な XML 文書

本節は実務で役立つことはないかもしれないが，DTD の概要について一応述べておくものである。

整形式 XML 文書である上に，妥当な XML 文書であるには，文書型定義（DTD）と呼ばれる文法を提供するマーク付け宣言が必要となる。このマーク付け宣言は，XML の設計書として，文書の妥当性の検証に用いられる。設計書である DTD と，設計に基づく XML 文書内容を関連付けるには，XML 文書内で DTD を定義する内部サブセットと，DTD を外部ファイルとして作成する外部サブセットの 2 つの方法がある。

ここでいうマーク付け宣言には，要素型宣言（element type declaration），属性リスト宣言（attribute-list declaration），実体宣言（entity declaration），記法宣言（notation declaration）がある。

要素型宣言では，要素名や要素の階層構造，出現順序や出現回数を定義することができる。また，属性リスト宣言では，従属させる要素名，属性名，属性値の候補，デフォルト値を定義することができる。実際の記述例を●リスト 2-9 に示す。

記法宣言は，外部ファイルとして参照する実体の名前を指定するために用いる。XML 文書が実体という記憶単位から成っていることは本章冒頭の「文書構造」の項で述べたが，実体は，XML 文書内容の中で参照する一般実体と，DTD 内で参照するパラメタ実体に分類される。さらに，一般実体は，XML プロセッサの解析対象となる実体と，画像ファイルの参照などに用いられる解析対象外実体に分けられる。解析対象外実体は，テキストであってもなくてもよく，テキストの場合であっても XML でなくともかまわない。つまり，jpeg や bmp 画像などを記法宣言によって指定することができる。

こういった DTD の文法と，使用する XML プロセッサが妥当性を検証するかどうかといった，XML プロセッサのふるまいの詳細については，XML 仕様書の「4 Physical Structures（物理構造）」で規定されている。

●リスト 2-9 要素型宣言と属性リスト宣言の記述例

```
<?xml version="1.0"?>
<!DOCTYPE 公開記事 [
    <!ELEMENT 公開記事 (オンライン記事)+>
    <!ELEMENT オンライン記事 (記事名)>
    <!ELEMENT 記事名 (#PCDATA)>
    <!ATTLIST 記事名 公開日 CDATA #REQUIRED>
  ]>
<公開記事>
  <オンライン記事>
    <記事名 公開日="2008-07-07">デザインの 30 年</記事名>
  </オンライン記事>
</公開記事>
```

なお，前述の文字参照（●リスト 2-7）は，実体参照を使って記述すると●リスト 2-10 のようになる。

●リスト 2-10 既出の●リスト 2-6 を実体参照で記述した例

```
<?xml version="1.0"?>
<!DOCTYPE 著作権表記 [
      <!ENTITY changeData "&#xA9;">
      <!ENTITY copy "Copyright&changeData; 1998-2008 PROJECT KySS. All rights
  reserved.">
  ]>
<著作権表記>
    &copy;
</著作権表記>
```

本書では，妥当な XML 文書に必要な定義については，DTD ではなく，XML Schema という仕様を用いる（次章参照）。XML Schema は，DTD と比べ，木構造であって理解しやすく，データ型や制約を定義できるため，より厳密な検証が可能であり，名前空間（後述）にも対応している。また，主要 XML データベース製品がサポートしているという利点もある。

2.1.7 XML 仕様書の記法

XML 仕様書では，文法は，EBNF（Extended Backus-Naur Form の略，拡張された BNF）という言語によって記述されている。文法の各々の規則は「symbol ::= expression」という形式で記号を定義している。XML1.0 仕様書においては，「6 Notation」で EBNF の読み方が述べられている。このうち必ずおさえておきたいのが「式(expression) 」である。これらの記号は，XML の構造定義に始まり，次章でとりあげる XPath といった XML 周辺仕様でも，しばしば使われている（●表 2-3）。

たとえば仕様書に「document ::= prolog element Misc*」と記載されていれば，これは，「document は，prolog と，element と，0 回以上の Misc から成る」という意味である。

●表 2-3 式の一覧

式	意味
A?	Aは出現（文書中に存在）してもよいし，出現しなくてもよい。オプション扱いである。
A ¦ B	AもしくはBのいずれか一方と合致する。 ¦ はパイプ記号といい，orを表す。
A B	Aの次にBが出現する。演算子は選択より優先される。A B ¦ C D は (A B) ¦ (C D) と同じである。
A - B	AからBを除外した任意の文字列と合致する。差集合と考えればよい。
A+	Aの1回以上の繰返しと合致する。連結は選択より優先される。A+ ¦ B+ は (A+) ¦ (B+) と同じである。
A*	Aの0回以上の繰返しと合致する。連結は選択より優先される。A* ¦ B* は (A*) ¦ (B*) と同じである。

2.2 名前空間による識別

2.2.1 XML 名前空間

名前空間の意義

前章で述べたように，XML 文書とは，ﾋﾄが情報を分類して，構造を決定し，データの意味を適切に表すﾒﾀデータを，タグの形で付加したものである。端的に言えば，XML 仕様書とは，ﾒﾀデータを付加するためのマニュアルといっていい。

しかしながら，ﾋﾄの判断には共通性もおおいにあるから，目的の異なる文書に同じ名前のﾒﾀデータを付加するケースは十分に考えられる。したがって，ﾋﾄが開発するXML文書データ処理のためのソフトウェアモジュールでも，XML形式のビジネス文書フォーマットでも，XMLで記述された他の言語でも，同じ要素型や属性が使われる可能性は少なくない。

また，先行して公開され広く使われていたり，業界団体で取り決めたものなど，既存の標準化された定義は増えている。それらの定義を，複数のソフトウェアモジュールから共通利用できるのであれば，新たに要素型や属性を定義するよりも，ﾘﾕｰｽする方がよい。
ところが，そうなると，要素名や属性名の認識問題が発生する。他のソフトウェアパッケージで同じ要素型が属性名が使われていると，どのソフトウェアモジュールで処理すべき要素や属性であるかが認識できなければ，正しい処理ができなくなってしまう。

そこで，同じ要素型や属性名を区別して衝突を回避する必要性が生じる。このような名前の衝突を回避するために，XML 名前空間（XML namespaces）という機構がある。

仕様書の入手先

XML 名前空間は，XML 文書作成時点で使われるものであるが，XML とは別個に，「Namespaces in XML」として策定されている。

XML 名前空間の仕様書は，W3C の XML のトップページ（http://www.w3.org/XML/）中の「XML Technology＞XML Essentials＞XML」コーナから入手することができる。また，Namespaces in XML 1.0 1999 年 1 月 14 日版の日本語訳は，「W3C Translations」コーナ（http://www.w3.org/2005/11/Translations/Lists/ListLang-ja.html）から入手することができる。

Namespaces in XML 1.0 は，2006 年 8 月 16 日に Second Edition が勧告となっている。また同日，Namespaces in XML 1.1 (Second Edition)も勧告となっている。1.1 では，URI のみならず IRI も利用可能となっている。URI と IRI については，次のページに掲載されている。

http://www.w3.org/2004/11/uri-iri-pressrelease.html.ja

本書では，Namespaces in XML 1.0（Second Edition）2006 年 8 月 16 日勧告版にしたがって解説する。Namespaces in XML 1.1 (Second Edition)では，本文中の「URI」は「IRI」となる。基本は同じである。

2.2.2 名前空間の宣言と適用

宣言の方法

XML 名前空間は，URI（1.1 仕様では IRI）によって特定される名前空間に，XML 文書中の要素名や属性名を結びつけることによって，要素や属性を識別する機構である。名前空間は，指定した URI によって識別される，XML 文書中で使われる要素名や属性名の集合であって，プログラミングで使われる「名前空間」とは意味合いが異なる。

基本的には，次のような手順で，識別されるようになる。

・任意の URI 参照によって特定できる名前空間を宣言する。
・名前空間の宣言の際に，接頭辞を指定する。
・接頭辞を要素名や属性名に付加する。
・アプリケーションは，接頭辞によって，要素や属性が特定の名前空間に属することを識別する。

実際の宣言例

次の●リスト 2-11 は，ルート要素<公開記事>に続く「xmlns:pk="http://www.PROJECTKySS.NET/schema"」で，接頭辞 pk の付く名前が「http://www.PROJECTKySS.NET/schema」という URI で指定される名前空間に属することを指定している。接頭辞は属性にバインドすることもできる。

●リスト 2-11 名前空間を使った XML 文書の例

```
<?xml version="1.0"?>
<pk:公開記事 xmlns:pk="http://www.PROJECTKySS.NET/schema">
  <pk:オンライン記事>
    <pk:記事名 pk:公開日="2008-07-07">デザインの 30 年</pk:記事名>
  </pk:オンライン記事>
</pk:公開記事>
```

名前空間は，接頭辞 xmlns:あるいは xmlns で始まる属性名を使って宣言する。属性値に指定した URI 参照は，名前空間を識別する名前空間名（namespace name）として使われる。アプリケーションは，指定されている名前空間 URI に接続して識別するわけではないので，名前空間 URI に指定したアドレスやファイルが存在しなければならないわけではない。あくまでユニークで識別可能なものであればよい。

接頭辞の付いてもよい名前のことを修飾名（qualified names）と呼ぶ。修飾名は，名前空間の接頭辞（prefix）と，':'（コロン）と，ローカル部分（local part）から成る。●リスト 2-11 の<pk:記事名>要素でいえば，pk が接頭辞，記事名がローカル部分である。名前空間名（namespace name）とローカル名（local name）のペアを，展開された名前（expanded name）という。この展開方法は，W3C の名前空間仕様書の「附属書 A」（A.3 Expanded Element Types and Attribute Names）に記載されている。

接頭辞は，名前空間宣言の URI 参照に結び付けられる。ただし，接頭辞「xml」で始まる文字は，予約されているので使うべきでない。また，URI 参照のような文字列は要素名や属性名として使えない文字を含むことがあるから，そのまま接頭辞に使うことはできない。

さらに，名前空間制約として，名前空間の接頭辞は，xml あるいは xmlns でない限り，接頭辞が使われる要素の開始タグか，接頭辞付きのマーク付けが内容となる要素（すなわち，上の階層にある要素）の中で宣言されなければならない。

スコープ

名前空間には適用できるスコープがある。名前空間宣言がある開始タグから，その対となる終了タグまでがスコープの範囲である。プログラムでいえばローカル変数のスコープといったところである。名前空間宣言は，指定された接頭辞に合致するスコープの範囲内にある，すべての要素に適用される。

●リスト 2-12 は，2 種類の名前空間を使った例である。

●リスト 2-12　2 種類の名前空間を使った例

```
<?xml version="1.0"?>
<公開記事
xmlns:pk="http://www.PROJECTKySS.NET/schema"
xmlns:sd="http://www.SeinDesign.NET/schema">
    <pk:記事名 pk:公開日="2008-07-07">デザインの 30 年</pk:記事名>
    <sd:記事名 sd:公開日="2008-07-07">デザインの 30 年</sd:記事名>
</公開記事>
```

まず，接頭辞 xmlns:で始まる属性名 xmlns:pk を使って名前空間を宣言し，接頭辞 pk を，名前空間宣言の URI 参照「http://www.PROJECTKySS.NET/schema」に結び付けている。さらに，接頭辞 sd を，名前空間宣言の URI 参照「http://www.SeinDesign.NET/schema」に結びつけている。これにより，接頭辞 pk の付いた<pk:記事名>要素は，名前空間「http://www.PROJECTKySS.NET/schema」の要素であり，接頭辞 sd の付いた<sd:記事名>要素は，名前空間「http://www.SeinDesign.NET/schema」の要素であると識別することができる。

このように，名前空間を使うことにより，同じ名前の要素や属性が，バインドされた接頭辞を利用して識別できるようになる。

2.2.3 デフォルト名前空間

デフォルト名前空間は，宣言された要素に名前空間の接頭辞が付いていない場合，宣言された要素と，その要素の内容である接頭辞なしのすべての要素に適用される。デフォルト名前空間は，属性には直接適用されない。

デフォルト名前空間宣言がスコープの範囲内にあれば，接頭辞なしの要素名に対応する展開名は，デフォルト名前空間の URI を名前空間名として持つことになる。

次の●リスト 2-13 の例では，最初の<オンライン記事>と，その中の<記事名>要素は，「http://www.PROJECTKySS.NET/schema」で指定される名前空間の要素である。2 個目の<オンライン記事>と，その中の<記事名>要素は，「http://www.SeinDesign.NET/schema」で指定される名前空間の要素である。

●リスト 2-13 デフォルト名前空間の例

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事 xmlns="http://www.PROJECTKySS.NET/schema">
    <記事名>デザインの 30 年</記事名>
  </オンライン記事>
  <オンライン記事 xmlns="http://www.SeinDesign.NET/schema">
    <記事名>デザインの 30 年</記事名>
  </オンライン記事>
</公開記事>
```

以上のような機構により，同名の要素や属性を識別することができるようになる。

XML 設計を行うとき，他の開発者による XML 文書の定義をすべて事前に知ることはできない。同じ要素型や属性名を部分的に含む同分野同業種の XML 文書が，世界のどこかのサーバに点在している可能性は高い。そのうえ，複数文書を連結したり縦断して処理するケースは増えている。それらがインターネットという開かれた環境の上で活用される場合には，衝突問題の発生を常に考えておく必要があるだろう。

また，同一名であっても，立場によってメタデータの意味が変わることもある。たとえば，販売側から見た<品名>と，仕入側から見た<品名>は，同じ要素名であっても，意味が異なる。販売側の<品名>データは売上に関連付けられ，仕入側では製造原価に関連付けられるので，販売側と仕入れ側の関係によっては，区別する必要が生じるかもしれない。さらに，デスクトップ・アプリケーションと Web アプリケーションの障壁がなくなり，アプリケーションの提供形態も，データの管理蓄積場所も，大きく変わりつつある。企業の統廃合などから文書管理方法が変わることもある。そうなると，同じ名前の文書が一か所に集められる可能性も生じる。

商用ベースでの利用だけでなく，他の一般的な文書も増え続けている。ブログのデータの転載や，不特定多数ユーザによる異なるサーバでのコンテンツの引用が増え続けると，レガシーデータの識別のために，標準化された識別機構がもとめられるであろう。名前空間を，そのような機構の代替えとして利用することも可能であろう。

2.3 ノード間の関係

2.3.1 異なる階層間の関係

親子関係

XML1.0 仕様書では，あらゆる階層関係にある要素間や属性間の関係について詳しくは言及していない。XML 仕様書で明確に規定されているのは，次のような，各要素の入れ子構造と，親子関係だけである。「ルート要素ではない<C>要素の他に 1 個<P>要素があって，<C>が<P>に含まれはするが，<P>の内容にある他の要素には含まれない場合，<P>を<C>の親（parent）といい，<C>を<P>の子（child）という（仕様書「2.1 Well-Formed XML Documents」）」。これは，たとえば，●リスト 2-14 のような状態のことを指す。

●リスト 2-14 XML で規定されている親子関係

```
<?xml version="1.0"?>
<P>
  <C>要素の内容</C>
  <B>要素の内容</B>
  <A>要素の内容</A>
</P>
```

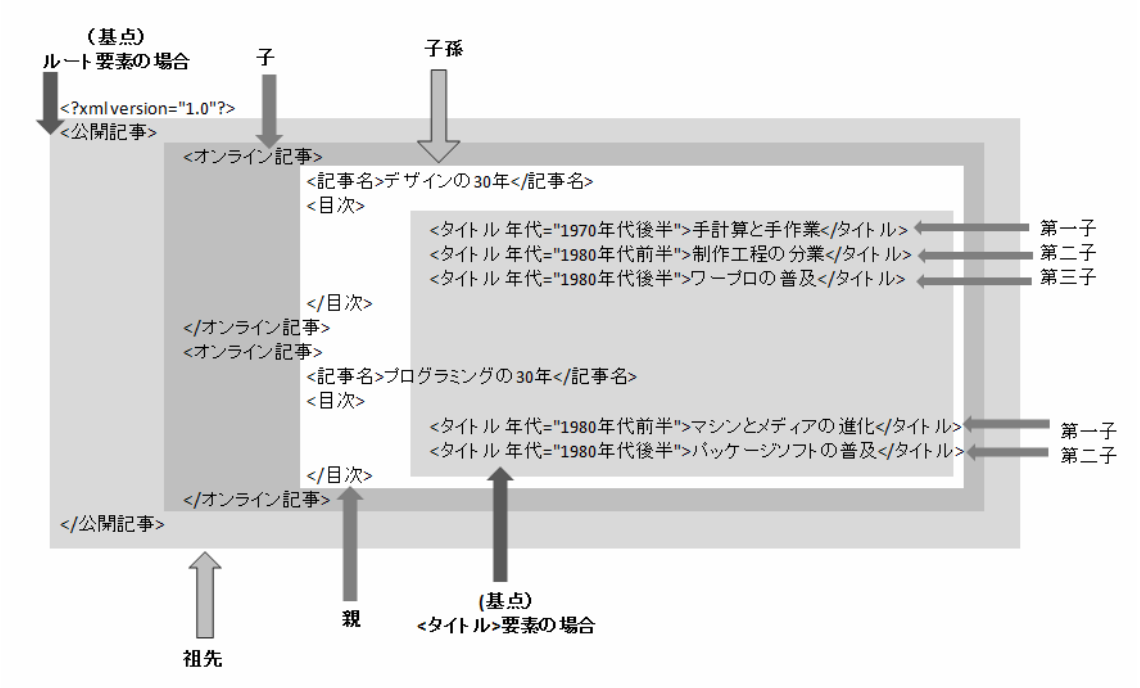
このリストは，<P>および<C><B><A>の 2 階層から成り，且つ<C><B><A>が1回ずつ出現している定型構の文書である。しかしながら，実際には，こういった 2 階層定型構造の文書は，RDB で表現できるため，あえて XML 文書として作成する必要性は薄い。XML では，その柔軟な階層構造を生かして，RDB では定義しにくい 3 階層以上の多階層，さらには不定形の構造を扱うことが多い。

2 階層以上の文書においては，「親」「子」だけで要素や属性の関係を表現することができない。とはいえ，XML 文書ファイルを蓄積するにとどまらず，XML 文書ファイル中のデータを活用するには，処理対象とする要素や属性をピンポイントで特定して選択する必要がある。この選択方法は，W3C により，XPath（次章で解説）という仕様で既定されている。

祖先子孫関係

XML 仕様書本体では親子についてしか言及されていないが，祖先・子孫・兄弟といった関係こそが，XML の基本中の基本なのである。そこで次章に進む前に，それらの概念を把握しておく必要がある。ここでは，●図 2-7 のような，要素が 4 階層になっている XML 文書を例にとって説明する。

●図 2-7 要素の最大階層が 4 階層の XML 文書



このリストには，<公開記事>というルート要素がある。

2 階層目の<オンライン記事>要素は，<公開記事>の子である。3 階層目の<記事名><目次>は，それぞれ「孫」にあたる。4 階層目の<タイトル>要素ノードはすべて「ひ孫」にあたるが，仕様にはひ孫を直接意味するものはなく，ひっくるめて「子孫 (descendant)」と呼ぶ。

逆に，子孫の側から見れば，親や祖先といった関係が成立する。<タイトル>要素から見れば，<目次>要素は親にあたる。また，<オンライン記事>要素は，親の親であって祖父にあたる。<公開記事>要素は，親の親の親であって曾祖父にあたる。これも先の「子孫」と同様，仕様には祖父を直接意味するものはなく，「祖先 (ancestor)」と呼ぶ。

2.3.2 同階層間の関係

兄弟関係

<公開記事>の中の 1 個目の<オンライン記事>の中の<目次>要素は，子として 3 個の<タイトル>要素を持っている。これら<タイトル>要素の関係は兄弟である。"年代"属性値が「1970 年代後半」の<タイトル>要素が第一子，「1980 年代前半」の<タイトル>要素が第二子，「1980 年代後半」の<タイトル>要素が第三子である。第一子から見れば，第二子は後に続く弟，第三子はその後に続く弟である。第二子から見れば，第一子は兄であり，第三子は弟である。第三子から見れば，第二子はすぐ前の兄，第一子は二つ前の兄である。このように，同一文書内の同一階層にあるノード間の関係は，相対的な関係で「前にある兄弟」「後に続く兄弟」として表わす。

2.3.3 その他の縁戚関係

叔父甥，従兄関係

親子，祖先子孫，兄弟以外の縁戚関係も，以上の関係の組み合わせで表現することができる。

たとえば，1 個目の<オンライン記事>要素の子である<記事名>から見ると，2 個目の<オンライン記事>要素は，親である 1 個目の<オンライン記事>要素の「後に続く兄弟」にあたる。親の弟であるから，叔父にあたる。だが，XML にも XML 周辺仕様にも叔父を直接指定する式はない。さらに，その子である，<記事名>要素は，叔父の子であるから従兄である。ところが，XML にも XML 周辺仕様にも従兄を表す式はない。

そこで，次のように段階を追って指定する。

・1 ステップ目：1 個目の<オンライン記事>要素の子の<記事名>を基点として，1 階層上の<オンライン記事>要素を指定する。これは「親」である。
・2 ステップ目：1 個目の<オンライン記事>要素を基点として，次の<オンライン記事>要素を指定する。これは「後に続く兄弟」である。
・3 ステップ目：2 個目の<オンライン記事>要素を基点として，その中にある<記事名>要素を指定する。これは「子」である。

このように，従兄を指定したい場合は，複数のステップを踏んで，親の後に続く兄弟の子，あるいは，親の前にある兄弟の子，というように指定する。

以上，駆け足で XML および XML 名前空間の最低限の事項について解説した。だが，XML 仕様書に則ってメタデータを付加しても，メタデータの付加されたデータを処理するための仕様は，別個に存在する。次章で，それらの予備知識について述べる。

第 3 章　XML の処理

本章では，XML 文書の定義および XML データ処理に必要な仕様の概要を解説する。

3.1 XML と XML 周辺仕様

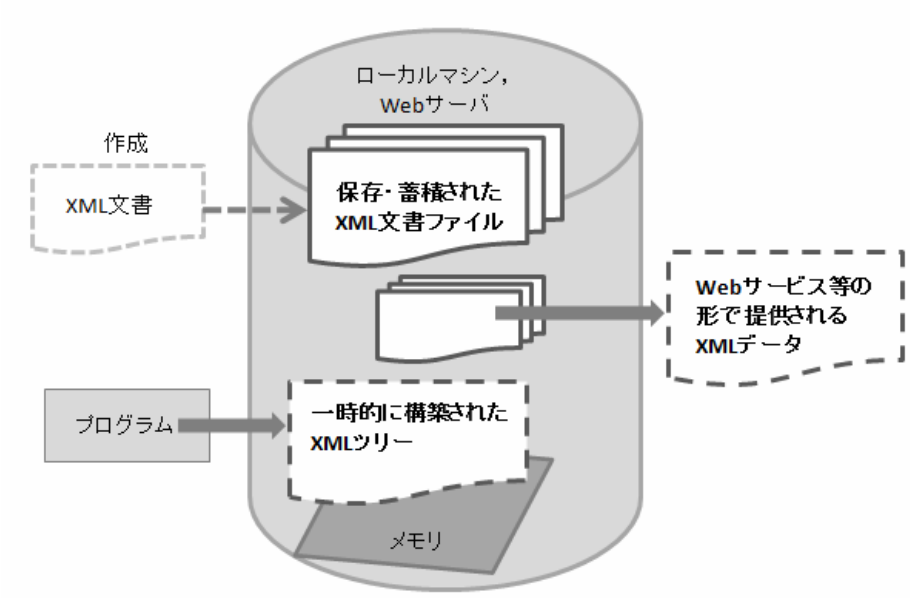
3.1.1 基本の XML 周辺仕様

各仕様の役割

整形式の木構造データ（以降，本書では XML データと呼ぶ）は，さまざまな形で存在している。ローカルマシンや Web サーバの中に，XML 文書ファイルとして保存・蓄積されていることもあれば，Web サービスなどの形で広く提供されていることもある。ファイルとして保存されてはいないが，プログラムによってメモリ上に展開された XML のツリーもある（●図 3-1）。

それらの XML データを蓄積したまま眠らせることなく，生きたデータとするには，XML 周辺仕様を用いて，データを処理する必要がある（●表 3-1）。前章で取り上げた「XML1.0」や「Namespace in XML」は，あくまでメタデータの付け方に関する仕様であって，データ処理の方法を提供するものではない。

●図 3-1　さまざまな形で存在する XML データ



XML 周辺仕様の併用

XML 周辺仕様には，W3C 仕様だけでなく，XML をサポートするミドルウエアや開発・実行環境のフレームワークを提供しているベンダの独自仕様もある（次ページ，●表 3-1）。本章では，W3C の代表的ないくつかの仕様と，ベンダの独自仕様として，Microsoft LINQ to XML を取り上げる。

●表 3-1 の各仕様については，いずれかひとつの仕様の知識を得ただけでは不十分である。
XML データ処理を含む独自開発アプリケーションの企画にあたっては，複数の仕様を組み合わせて用いることによって実現可能となるであろう新しい処理を考え出さなければならない。

また，受託開発の設計では，どの仕様を選択して組み合わせれば目的の処理を実装できるかを判断しなければならない。企画設計を担うヒトの「判断」ひとつで，処理の速度・精度・操作性・安定性・拡張性は大いに異なってしまう。

実装（プログラミング）においても，いずれかひとつの仕様を単独で用いることは極めて少なく，併用を避けることができない。

詳しくは後述するが，XSLT および XQuery を用いるには，XPath の知識が必須である。また，LINQ における範囲変数の考え方は，XPath にほぼ同じである。DOM と XSLT，あるいは LINQ と XSLT の双方に通じていなければ，使い分け方を検討することができない。これらの併用方法の種類は，XML 周辺仕様の数の階乗と，独自拡張仕様を発表しているベンダ数を乗算した数より，はるかに多い。

●表 3-1 XML および XML 周辺仕様の名称と機能

工程	仕様の名称	役割	策定機関
文書作成	**XML**（Extensible Markup Language）	XMLの基本仕様（第2章で解説）。	W3C
	Namespace in XML	同じ要素型や属性名を持つXML文書を利用／管理するための名前空間を指定する（第2章で解説）。	
データ処理	**XPath**（XML Path Language）	XML文書中の任意のノードをアドレス指定する仕様。XMLに関わる，あらゆる業務に必要（ノード間の関係については第2章で解説）。演算子と関数はXQueryとの共通仕様。	W3C
	XQuery（XML Query）	XML文書への問い合わせ言語。XMLデータベース機能を持つ複数のミドルウエアがサポートしており，中規模以上のXMLアプリケーション開発で多用される。XPathと組み合わせて使う。	
	DOM（Document Object Model）	XMLオブジェクトにアクセスして操作するためのAPI。プログラムコードの中で用いることにより，XML文書の構築，構造のナビゲート，要素や内容の追加・変更・削除を実装することができる。	
	XSLT（XSL Transformations）	XML文書を別のXML文書やHTMLに変換する言語。XML Webアプリケーション開発で多用される。XHTMLにも変換することができる。	
	LINQ（Language-Integrated Query）	.NET Framework3.5以降で動作するベンダ独自のクエリ技術。クエリの基本パターンは，RDB(LINQ to SQL)，XML(LINQ to XML)，Object(LINQ to Object)を問わず同一。XQuery，DOM，XSLTによる処理を代替え可能。	Microsoft
定義・検証	**XML Schema**	XML文書の構造を定義する。妥当なXML文書を検証する場合に必須。RelaxやDTDという選択肢もある。	W3C

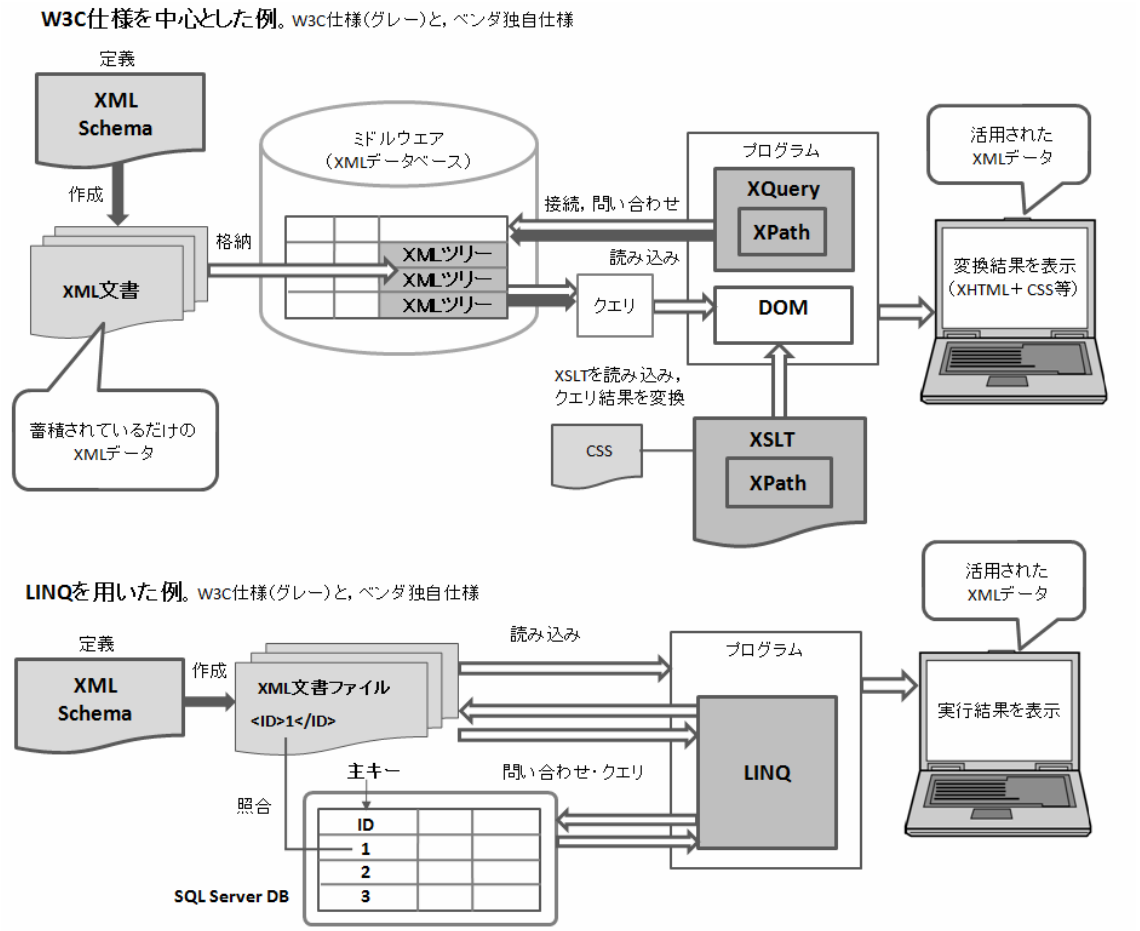
●図 3-2 に，膨大な併用方法の中の一例を示す。上図は，SQL Server をミドルウエアとして用い，XQuery を中心として処理する方法である。SQL Server に XML ツリーを格納し，XQuery の中で XPath 式を用いてクエリを取得，ベンダ独自仕様の DOM で読み込んで XSLT を適用して表示している。

下図は，XQuery ではなく LINQ を採用し，SQL Server に格納している RDB 形式のデータと，XML 文書ファイルを，LINQ で結合して処理する方法である。

本書では，これ以降，XML Schema の節を除き，整形式であって妥当でなくてもよい XML 文書を主体に話を進める。

なお，外部 DTD サブセットのあるものは XML 文書，文書のみのものは文書実体と呼ぶが，本書では便宜上，文書実体であるルートノード以下を「XML 文書」，ファイルの形で保存・蓄積されたものを「XML 文書ファイル」と表記する。また，文書ファイルであれ，メモリ上に展開された木構造のオブジェクトであれ，どのような形で存在しようと，整形式の構造を持つ XML の木を「XML ツリー」と表記する。そして，XML 文書ファイルあるいは XML ツリー中の個々のデータおよびデータ群を「XML データ」と表記する。

●図 3-2 XML 周辺仕様の併用方法の例



3.2 XPath によるアドレス指定

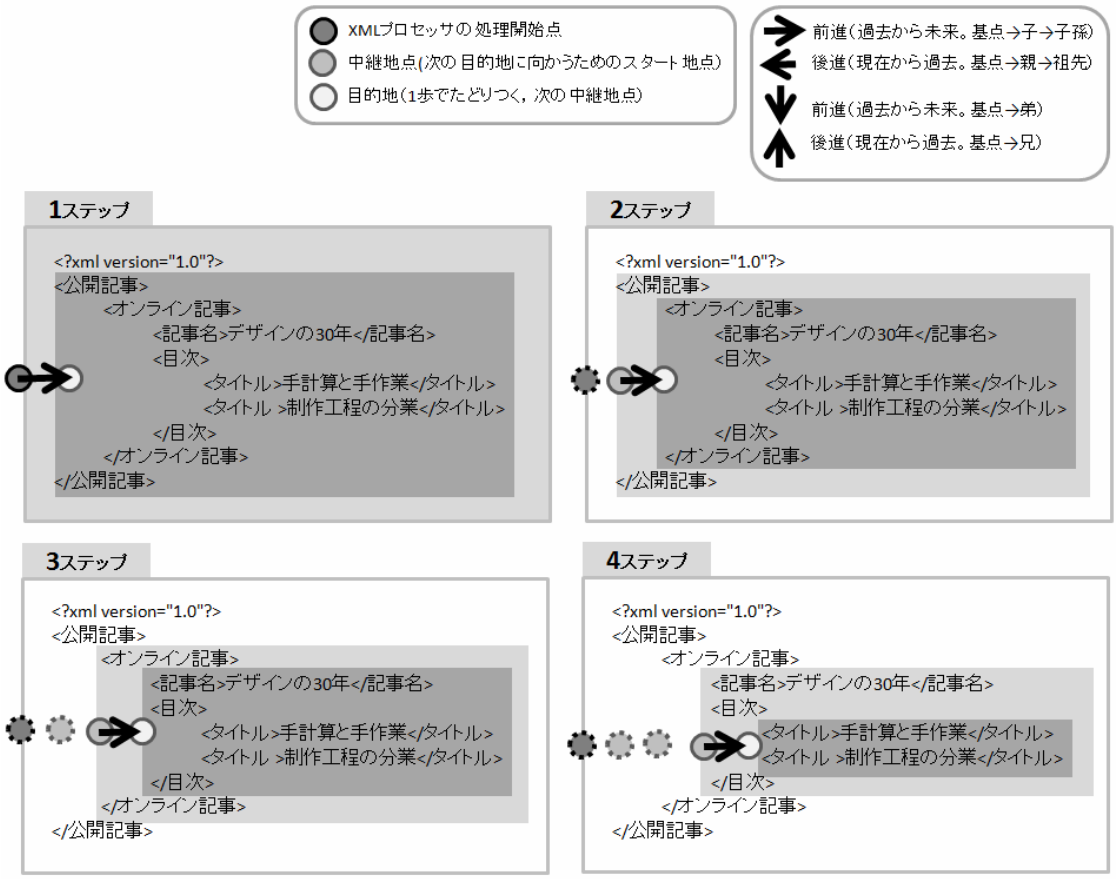
3.2.1 XPath の概要

XPath の役割

いかなる XML 周辺仕様を使うにせよ，必ず理解しておかなければならない仕様がある。それは XPath (XML Path Language)である。XPath の「式」をおさえてから，他の仕様に関する知識をスパイラル式に増やしていくと，理解が進むであろう。

XML データを処理するにあたっては，まず「XML の木の中の何を処理するのか (処理対象)」を指定し，次に「いかに処理するのか (処理方法)」を記述する必要がある。このような，処理対象の選択のために，XPath を用いる。XPath のパス式 (Path Expressions) では，地図上で道 (パス) をたどるかのごとく，XML ツリー上で目的の要素や属性等という目的地までの道筋をたどることができる。信号の度に一度歩を止め，次の目標になる信号を目指してステップを踏み出す，という動作を繰り返しながら目的地に向かう様子をイメージすれば分かりやすいだろう (●図 3-3)。

●図 3-3 XPath による処理対象の指定



パス式は，XQuery や XSLT の中で処理対象を指定するためにも用いられる。(利用頻度としては多くはないが) ベンダ独自仕様の DOM のメソッドを用いる場合，パラメータにパス式を直接指定することもあり，Microsoft LINQ to XML では XPath 拡張機能を用いることもある (詳細は後述)。

どの仕様を採用するにせよ，階層構造を正確にたどるためには，パス式の考え方への理解がもとめられるので，XPath は知っておいた方がよい仕様である。

XPath 仕様書と XQuery 仕様書の関係

XPath の機能について語る前に，仕様書の状況について説明しておこう。これ以降で取り上げる仕様についても，仕様の状況について触れている。複数の仕様が関連し合いながら，短期間にバージョンアップし続けているので，どの仕様書を参考にして学べばよいのか大いに悩むことになるのは必至だからである。各々の仕様書は，「XML Technology＞Query」コーナから入手することができる。

XPath 1.0 は，1999 年 11 月 16 日に W3C 勧告となった。当初は，XSLT1.0 および XPointer のために設計された，XML 文書中の部分をアドレス指定するための言語であった。また，文字列･数値･ブール値を操作するための多数の関数や演算子が提供されていた。この XPath 1.0 と XSLT 1.0 (後述) が W3C 勧告となった後，XQuery (後述) が発表され，XPath，XSLT，XQuery の 3 仕様は関連付いた形で進化してきている。XQuery 1.0 は，XPath2.0 の拡張であり，この両者は，構文の正しい式においては同じ評価結果を返す。

本書執筆時点で，XPath に関わる主要な勧告仕様は，次の 3 本である。

・XQuery 1.0 and XPath 2.0 Data Model (XDM)　2007 年 1 月 23 日勧告版

XPath 1.0 および草稿段階での XML Query Data Model のデータモデルに基づき，XPath 2.0，XSLT 2.0，XQuery，関連仕様のデータモデルを規定する。

・XPath 2.0 2007 年 1 月 23 日勧告版

パス式（path expression）を用いて，XML 1.0 または XML 1.1 文書中のノード群の階層構造をたどる，アドレス指定（hierarchic addressing）を行うための仕様。「XQuery XPath Data Model」で定義されたデータモデルに従った値を処理するための式言語である。このデータモデルでは，integers（整数型），strings（文字列型），booleans（ブール型）といったアトミック値（an atomic value，分割できない最小限の値，「原始値」とも呼ばれる），ノード群とアトミック値への参照を含むシーケンス（sequence）を木構造の形で扱うことができる。XPath 式の評価結果は，文書，アトミック値，シーケンスから得られるノード群（nodes）となる。

XPath 2.0 は，XPath1.0 に対してのスーパーセットという位置づけである。XPath1.0 は XML Schema に先立って勧告されたが，後に策定された XPath2.0 は，XML Schema による検証でも用いられる，豊富なデータ型（data types）と型情報をサポートする仕様となっている。XPath1.0 に対する後方互換モードは，XPath2.0 仕様書の「I Backwards Compatibility with XPath 1.0」で規定されている。

・XQuery 1.0 and XPath 2.0 Functions and Operators　2007 年 1 月 23 日勧告版

「XML Schema Part 2: Datatypes Second Edition」および，「XQuery 1.0 and XPath 2.0 Data Model」で定義されているデータ型に基づく関数ライブラリ。「XQuery 1.0 and XPath 2.0 Data Model」で定義されたノード群および，ノードのシーケンスを計算する演算子と関数に関する仕様である。「XPath 2.0」「XQuery 1.0」「XSLT 2.0」および関連 XML 標準仕様で使われるために定義されたものである。

本書では，上記「XPath 2.0 2007 年 1 月 23 日勧告版」のアドレス指定に関する仕様のうち，実務で最も使用頻度の高い「XPath バージョン 1.0」との共通部分について取り上げた。バージョン 1.0 とバージョン 2.0 では，若干の用語の違いがあるため，2.0 仕様書の表記で解説した。なお，開発フレームワークやミドルウエアが，これらの仕様に対応していなければ，正しく記述しても動作しないので注意しなければならない。XPath1.0 のパス式および関数は，Microsoft Internet Explorer や Firefox に搭載されている XML プロセッサで動作する。

3.2.2 ノードの扱いと順序

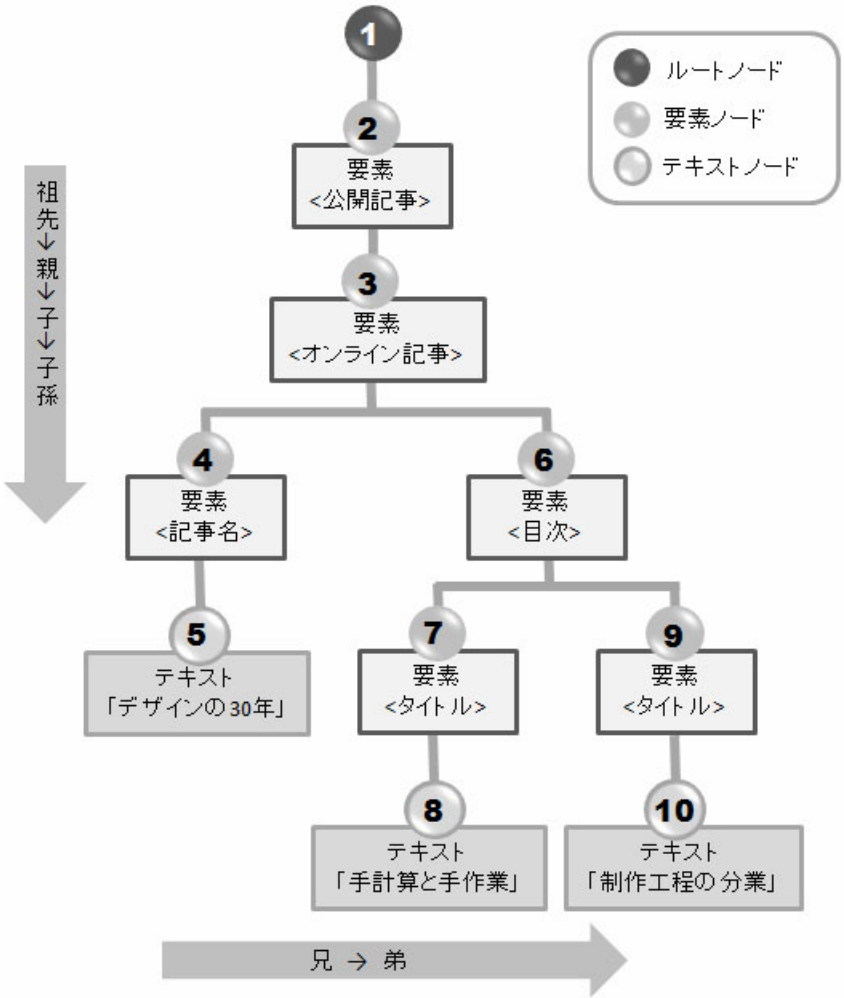
7 つのノード

XPath において，要素および属性間等の関係は，木構造の持つノードの関係で表現される（●図 3-4）。XPath では，7 種類のノードが規定されている（「XQuery 1.0 and XPath 2.0 Data Model」6 Nodes）。それは，文書（document），要素（element），属性（attribute），テキスト（text），名前空間（namespace），処理命令（processing instruction），コメント（comment）である。

文書順，逆文書順

XPathでは，XMLツリーのノード群を，階層構造をたどるパス式を用いて特定することができる。パス式を処理することによりアクセスできるノード群には，1 個以上の XML 文書や木構造の断片における順番がある。それは，順次進行する文書順（document order）と，逆行する逆文書順（reverse document order）である。前述の●図 3-3 の XML でいえば，文書順の場合の順序は，●図 3-4 のようになる。

●図 3-4 XML ツリーのノードの順序



図のように，木構造では，ルートノードが最初のノードである。各ノードは，ルートノードの子あるいは子孫である。子や子孫は，後に続く兄弟より先であり，兄弟の相対順序は親ノードの子プロパティの中での順番になる。

また，要素ノードには名前空間ノード群が従属し（implementation-dependent，must 事項ではない），属性ノード群が従属する。名前空間ノードがなければ，要素ノードに属性ノードが従属する（これも implementation-dependent）。分割したツリーについては，各々のツリーの中での順序は固定だが，複数のツリーは接ぎ木した順番になる。

3.2.3 動的コンテキスト

XPath の用語と概念

XPath 2.0 における処理には，2 段階のフェーズがある。一つは，静的コンテキスト（static context）に基づく静的解析フェーズ（static analysis phase）で，操作ツリー（operation tree）を生成する。静的コンテキストとは，XPath1.0 との後方互換性の可否や名前空間といった，XML プロセッサが XML 文書を解析する際に前提条件となる情報である。

もう一つは静的解析フェーズ完了後，動的コンテキスト（dynamic context）が使われる動的解析フェーズ（dynamic evaluation phase）である。動的コンテキストは，XML プロセッサがパス式の指定に従って XML 文書を走査するにつれ，パス式を構成する各部の評価からリアルタイムに得られる情報である。

パス式で処理対象とするノードまたはノード群を特定する時，基点と特定する対象の間には相対的な関係が成り立つ（第 2 章「2.3 ノード間の関係」参照）。基点から処理対象までのパスを，複数のステップを経てたどるには，ステップを進める度に中継地点のノードを経由することになる。XML プロセッサは，この走行した中継地点の記録を保持する。つまり，式のフォーカスが移動する度に，式の評価結果であるアイテム群の軌跡が残っていくことになり，動的コンテキスト（文脈）という概念が生じる。

動的コンテキストを使ったパス式を記述するには，次の 5 つを明確にする必要がある。

・基点：処理の開始地点。
・処理対象：特定したいノード。
・軸（axis）：コンテキスト・ノード（中継地点のノード）と，次に選択するノードとのツリーの関係を示す，方向指示器のようなもの。
・式のフォーカス（the focus of the expression）：基点から対象までの間を移動する。動的コンテキストは，式のフォーカスと呼ばれる「コンテキスト・アイテム（context item）」「コンテキスト・ポジション（context position）」「コンテキスト・サイズ（context size）」という 3 本柱から成る。
・カレント・アイテム：XML プロセッサが処理するアイテム。

パス式の記述例

次に実際のパス式の記述例を挙げる。ここでは，●リスト 3-1 のような XML 文書の中で，2 番目の<タイトル>を選択するパス式について解説する。

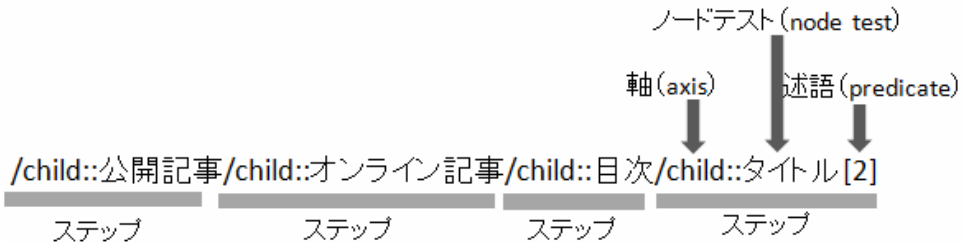
まず，文書ノードに基点を置いて，2 番目の<タイトル>を特定していく。子要素を順番にたどっていくため，子を表す軸である「child::」を使う。次の XPath 式で，<公開記事>の中の，<オンライン記事>の中の<目次>の中の，2 番目の<タイトル>を選択することができる。/で区切られた部分を中継地点として，基点から対象までフォーカスが移動していく。

このパス式（●図 3-5）では，child を軸（axis），公開記事・オンライン記事・目次・タイトルをそれぞれノードテスト（node test），［2］を述語（predicate）と呼ぶ。

●リスト 3-1　最大階層が 4 階層のサンプル XML 文書（SampleData_Article.xml）

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名>デザインの 30 年</記事名>
    <目次>
      <タイトル 年代="1970 年代後半">手計算と手作業</タイトル>
      <タイトル 年代="1980 年代前半">制作工程の分業</タイトル>
      <タイトル 年代="1980 年代後半">ワープロの普及</タイトル>
      <タイトル 年代="1990 年代前半">DTP の普及</タイトル>
      <タイトル 年代="1990 年代後半">Web サイトの連携</タイトル>
      <タイトル 年代="2000 年代前半">情報デザインの始まり</タイトル>
    </目次>
  </オンライン記事>
</公開記事>
```

●図 3-5　リスト 3-1 で，2 番目の<タイトル>を選択するパス式



XML プロセッサは，まず，<公開記事>を評価し，これをコンテキスト・ノードとして，その子である<オンライン記事>で評価する。次に，<オンライン記事>を評価し，これをコンテキスト・ノードとして，その子である<目次>で評価する。さらに，<目次>の子である<タイトル>で評価して結果のシーケンスを作り，サブ式である「コンテキスト・ポジションが 2 番目である」というフィルタリング条件式（filter expressions）で評価するために，各々のアイテムに新しいフォーカスを作る。最後に，この条件で，<タイトル>の評価結果のシーケンス中の各々のアイテムを再評価する。評価されているシーケンスには，評価の間のみインナー・フォーカス（inner focus）が存在するが，評価が完了すると，次の処理に備えてフォーカスは，この式の基点に戻る。

式のフォーカスを成すコンテキスト・アイテムは走査中のカレント・アイテムでもあり，1 個のアイテムは，アトミック値あるいはノードである。コンテキスト・アイテムがノードの場合は，コンテキスト・ノードとして参照されるようになる。コンテキスト・アイテムは，.（シングルドット）による式で表すことができる。

コンテキスト・ポジションは，走査中のカレント・アイテムのシーケンスにおけるコンテキストのポジションを表す，0 より大きい整数値である。

コンテキスト・ポジションを得るには，fn:position()式を使う（XPath1.0 では position 関数として知られる）。シーケンス中のポジションの最初のアイテムは常に「1」であるので，プログラマは開発フレームワークの

提供する「0」から始まるインデックス番号や，DOM で使われるインデックス番号と混同しないように注意しなければならない（第 6 章「処理に必要な値の取得」参照）。<目次>の子の<タイトル>シーケンスでは，1 個目の<タイトル>のコンテキスト・ポジションは 1 であり，2 個目の<タイトル>のコンテキスト・ポジションは 2 である。

コンテキスト・サイズは，走査中のカレント・アイテムのシーケンスのアイテム数であり，0 以上の整数値である。fn:last()式を使うと，コンテキスト・サイズを得られる（XPath1.0 では last 関数として知られる）。<目次>の子の<タイトル>シーケンスでは，コンテキスト・サイズは 6 である。したがって，コンテキスト・ポジションは，コンテキスト・サイズ以下ということになる。

パス中の各々のステップは，次のステップのコンテキスト・ノード群となる。その結果，パス中の最後のステップは，評価後に，アトミック値のシーケンスを返すことがある。

図 3-5 に挙げた記述方法は，XPath1.0 では，/child::公開記事/child::オンライン記事/child::目次/child::タイトル[position() = 2]と記述する。これは，●図 3-5 のように，/child::公開記事/child::オンライン記事/child::目次/child::タイトル[2]と置き換えられる。さらに，子を表す child 軸は，省略構文では/（スラッシュ）で代替えできる。これを，省略構文（abbreviated syntax）という。child 軸を省略構文で記述すると，**●リスト 3-2** のようになる。XPath2.0 でも XPath1.0 でも同じである。

●リスト 3-2　図 3-5 に挙げたパス式で，child 軸を省略構文で記述した例

/公開記事/オンライン記事/目次/タイトル[2]

さらに，XPath 式の記述方法は，ただひとつというわけではない。このケースでも，child 軸を繰り返して使う必要はない。第 2 章で触れたように，ノード間の関係は，親―子だけでなく，祖先―子孫や，兄―弟関係でも表現できる。<タイトル>要素は，木構造のルートから見れば子孫である。したがって，XPath で用意されているいくつかの軸のうち，子孫を表す descendant 軸を使えば，/descendant::タイトル[2] といった簡潔なパス式でも，同様に選択することができる。

なお，XPath1.0 では，/で区切られた部分はロケーションステップ，ロケーションステップを続けて記述したものはロケーションパスと呼ばれている。

3.2.4 基本構文

パス式

パス式は，XPath 仕様書で規定された構文を使って記述する。その式は，基本的に，「/」あるいは「//」で区切られたステップの連続であるが，「/」や「//」で始めなければならないわけではない。パス式のステップの基点は，コンテキストノードを含む木構造のルートノードであり，これを表す「/」は，ルート以下を指定する document-node()/ の省略形であるが，これは記述しなくてもよい。また，「//」は子孫を指定するパスの省略形であるが，属性ノード群や名前空間ノード群は子孫には含まれない。「/*」あるいは，文字間に半角スペースが空いている「/ *」という式では，*は，ワイルドカードを表す。

ステップ

パス式の部分をステップ(step)という。ステップは，アイテム群のシーケンスを生成したり，0 個以上の述語でシーケンスをフィルタリングする条件である。つまり，軸ステップ(axis step)あるいはフィルタリング条件式(filter expression)のいずれかである。軸ステップは，軸とノードテストと述語リストから成る。述語はリストとして複数列記することができる。

軸

軸(axes)には，前進軸(forward axis)と後進軸(reverse axis)がある(●表 3-2)。文書順でコンテキストノードに続く，コンテキストノードおよびノード群を含む軸は前進軸である。一方，文書順でコンテキストノードに先行する，コンテキストノードおよびノード群を含む軸は逆行軸である。各軸は主なノード型を持つ。attribute 軸のノード型は attribute，namespace 軸のノード型は namespace，それ以外は element である。

●表 3-2 XPath の軸

前進軸(ForwardAxis)		後進軸(ReverseAxis)	
軸名	選択対象	軸名	選択対象
child	コンテキストノードの子を含む。	parent	コンテキストノードの親を返す。親がなければ空のシーケンスを返す。 属性ノードから見れば要素ノードは親にあたる。但し，要素ノードから見て属性ノードは子にはあたらない。
descendant	コンテキストノードの子要素をひっくるめた子孫を含む。これは，コンテキストノードの子と，子の子．．．というように続くものを指す。	ancestor	コンテキストノードの親要素をひっくるめた祖先を含む。これは，コンテキストノードの親と，親の親．．．というように続くものを指す。コンテキストノードがルートノードでない限り，ルートノードを含む。
descendant-or-self	コンテキストノードと，コンテキストノードの子孫群を含む。	ancestor-or-self	コンテキストノードと，コンテキストノードの祖先ノード群を含む。この軸は木をさかのぼるので，常に，ルートノードを含むことになる。
following-sibling	コンテキストノードの親の子供たちの中で，文書順ではコンテキストノードより後に出現する兄弟ノードを含む。コンテキストノードが属性あるいは名前空間ノードであれば，この軸は空である。DOM(後述)のnextSiblingと混同して憶えないように注意。	preceding-sibling	コンテキストノードの親の子供たちの中で，文書順ではコンテキストノードより前に出現する兄弟ノードを含む。コンテキストノードが属性あるいは名前空間ノードであれば，この軸は空である。DOM(後述)のpreviousSiblingと混同して憶えないように注意。
following	コンテキストノードと同じ文書内にあり，文書順でコンテキストノードの後に出現し，コンテキストノードの子孫を除くすべてのノード群を含む。	preceding	コンテキストノードと同じ文書内にあり，文書順でコンテキストノードの前に出現し，コンテキストノードの祖先を除くすべてのノード群を含む。
attribute	コンテキストノードの属性群を含む。コンテキストノードが要素でなければ空。		
self	コンテキストノード自身を含む。		
namespace	コンテキストノードの名前空間ノード群を含む。コンテキストノードが要素ノードでなければ空。この軸はXPath 2.0では非推奨。XPath 1.0互換モード用の軸。		

ノードテスト

ノードテスト(node test)とは，ステップで選択するノードが真となる条件のことである。ノードテストとして指定する条件は，軸の主要なノードの型や，修飾名あるいはワイルドカードである。もし，名前空間を使っているのであれば，「attribute::pk:年代」は，"pk:年代"属性を選択する。「child::*」は，コンテキストノードのすべての要素の子を選択し，「attribute::*」は，すべての属性を選択する。その他の主な記述例は●表 3-3 の通りである。

●表 3-3　ノードテストの一例

ノードテストの例	機能
node()	あらゆるノード型で真である。
text()	あらゆるテキストノードで真である。
comment()	あらゆるコメントノードで真である。
element()	あらゆる要素ノードで真である。
attribute()	あらゆる属性ノードで真である。
document-node()	あらゆる文書ノードで真である。

述語

式の述語（predicate,「述部」とも呼ばれる）は，式を角括弧で挟み，[式] のように記述する。シーケンスをフィルタリングする機能を持ち，複数列記して，左側にあるものから順次絞り込むことができる。

たとえば先の●リスト 3-1 で，"年代"属性の値が「1980 年代前半」である，任意のコンテキストノードのすべての子孫要素<タイトル>を選択するのであれば，●リスト 3-3 のように述語を指定する。但し，後述の XSLT スタイルシート内で指定する場合は，XPath 式全体を""（ダブルクォート）で挟むために""が重複するので，述語は''（シングルクォート）で挟む。

●リスト 3-3　述語の記述例

・XPath 式

descendant::タイトル[attribute::年代="1980 年代前半"]

・XSLT の中で使う場合（XSLT については後述）

<xsl:value-of select="descendant::タイトル[attribute::年代='1980 年代前半']"/>

省略構文

省略構文を使えば，attribute 軸であればattribute::と記述するところを，@ひとつで置き換えることができる。[attribute::年代="1980 年代前半"] は，[@年代="1980 年代前半"]と省略できる。また，..は，parent::node()と同じで，一階層上を指す。軸ステップが省略されている時のデフォルト軸は，通常は，child である。たとえば，「オンライン記事/記事名」というように，child::が省略されていても，デフォルトは child であるから，これは，「child::オンライン記事/child::記事名」と同じである。

記述例

次ページに，使用頻度の高い記述例を示す（●表 3-4）。左側が基本構文，右側が省略構文である。

以上のように，XPath のパス式を用いることにより，親子，祖先子孫，兄弟とポジション，および自分自身，属性，テキスト等の指定や，それらの組み合わせによって，あらゆるノードを特定することができる。

●表 3-4 XPath 式の記述例

選択対象	基本構文	省略構文	パス式の意味
子	child::要素C	要素C	コンテキストノードの子である 要素C を選択する
	child::*	*	コンテキストノードのすべての子要素を選択する
	child::node()	node()	コンテキストノードのすべての子を選択する。但し，属性は子ではないので返されない。
	child::要素C[fn:position() = 1]	要素C[1]	コンテキストノードの子の最初の要素Cを選択する
	child::要素C[fn:position() = fn:last()]	要素C[fn:last()]	コンテキストノードの子の最後の要素Cを選択する
	child::要素B[child::要素D = '概要']	要素B[要素D='概要']	<要素D>が「概要」という値を持つという条件に合致する，コンテキストノードの子である要素Bを選択する。
	child::要素B[attribute::属性A and attribute::属性B]	要素B[@属性A and @属性B]	"属性A"と"属性B"の両方を属性として持つ，コンテキストノードの子のすべての<要素B>を選択する。
親	parent::node()	..	コンテキストノードの親を選択する。コンテキストノードが属性ノードであれば，属性が付加されていても要素ノードを返す。
	parent::node()/child::要素B	./要素B	コンテキストノードの親の子の要素Bを選択する。
子孫	child::要素A/descendant-or-self::node()/child::要素D	要素A//要素D	子の要素Aのすべての子孫の要素Dを選択する。
祖先	ancestor::要素A		コンテキストノードの祖先のすべての要素Aを選択する。
兄弟	following-sibling::要素B[fn:position() = 1]	following-sibling::要素B[1]	コンテキストノードの後に続く兄弟である要素Bを選択する。
	preceding-sibling::要素B[fn:position() = 1]	preceding-sibling::要素B[1]	コンテキストノードの前に出現する兄弟である要素Bを選択する。
自分自身	self::要素C		コンテキストノードが要素Cであれば選択するが，そうでなければ空のシーケンスを返す
属性	attribute::属性A	@属性A	コンテキストノードの属性Aを選択する
	attribute::*	@*	コンテキストノードのすべての属性を選択する
テキスト	child::text()	text()	コンテキストノードのすべてのテキストノードを選択する

※ XPath1.0では，接頭辞fnを用いることなく関数を指定する。

演算子

パス式の中では，演算子を使うことができる。XPath1.0 と XPath2.0 に共通する主な演算子には，次のようなものがある。

加算，乗算には，+，*を使う。浮動小数の除算には，パス式で使う/ではなく，div を使う。剰余には mod を使う。なお，減算では，-演算子の前に空白を入れて使う。XML で，要素の名前に-を使ってもよいことになっているため，識別が必要だからである。比較演算子としては，=，!=，<，<=，>，>=が共通している。

XPath1.0 でオペランドの結合（union）に用いられる｜演算子は，XPath2.0 ではオペランドのノード・シーケンスの結合に用いられる。XPath2.0 では，他にも多数の演算子が用意されている。union（和集合）があるということは，当然，intersect（積集合）や except（差集合）も用意されている。また，関数も多数用意されている。関数については，次の XQuery の節で触れる。

3.3 XQuery によるクエリ

3.3.1 XQuery の概要

XML 文書は，そのまま XML 文書ファイル（ネイティブ XML 文書ファイル）として蓄積し，利用することができる。ネイティブ XML 文書ファイルは，実に処理しやすいデータベースといえよう。必ずしも文書構造の定義を必要としない整形式 XML 文書であれば，文書作成作業も比較的容易である。ネイティブ XML 文書ファイルの利用方法として代表的なものには，検索処理がある。プログラムコードの中で XPath 式を使って検索条件を指定することにより，条件に合致した XML データを取得することができる。

ネイティブ XML 文書ファイルは，単独で利用するだけでなく，RDB と併用することにより，両者の特長を生かした処理を実装することができる。行列で十分に表現可能なデータは RDB として蓄積し，開示可能な木構造の文書ファイルは，ネイティブ XML データとして蓄積し，プログラムによって両者を結合したり参照して処理するという方法をとる。もし，官公庁や企業間取引における案件で，取り扱うデータが RDB では表現しづらく，階層構造の方が適しているのであれば，部分的に XML を採用することも考えられる。

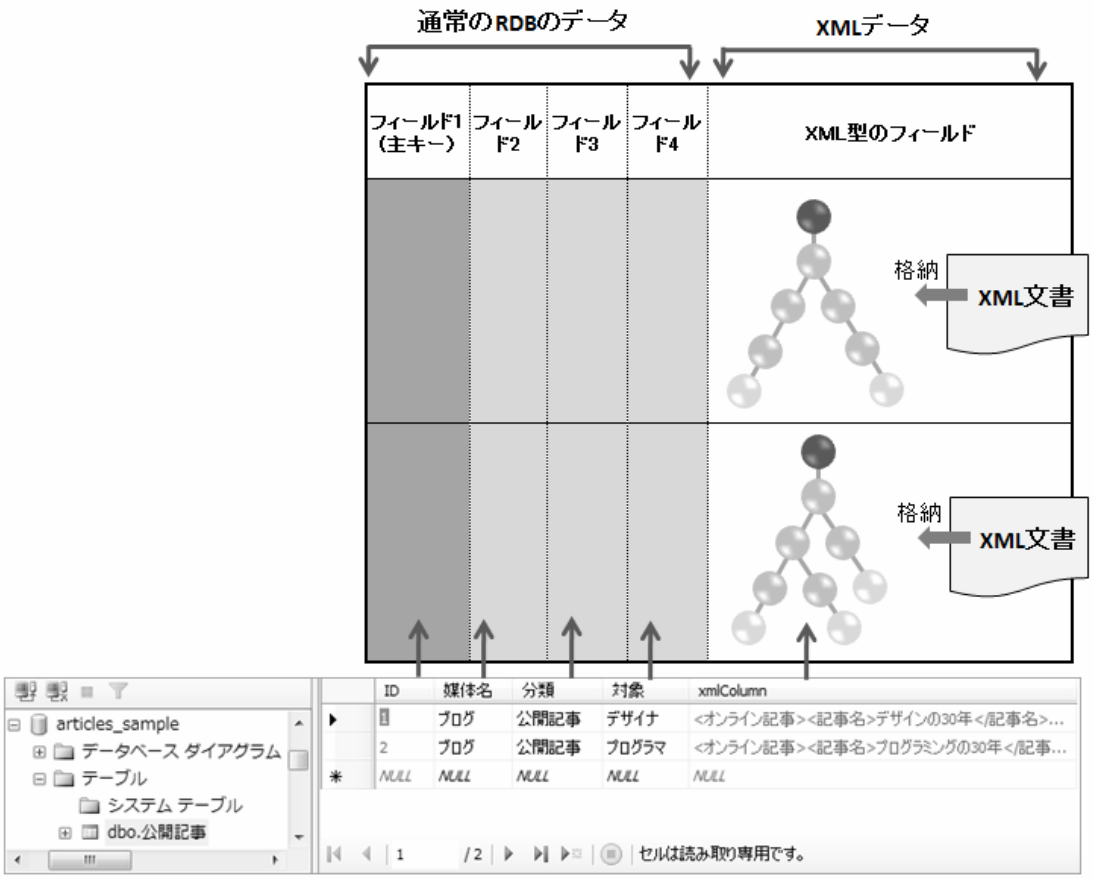
しかしながら，そういった案件には，情報公開を前提とする（つまり，XML データ内に機密情報を含まない）ものだけではなく，セキュア且つトランザクションを要するものもある。そのため，単純なテキストファイルであるネイティブ XML 文書をそのまま利用するのではなく，ミドルウエア（XML データベース）に XML ツリーを格納して処理することがある。

このようなミドルウエアを採用するケースでの XML 利用を視野に入れた場合，知っておきたい W3C 策定のクエリ言語が，XQuery（XML Query）である（●図 3-6）。XML の高い汎用性を生かし，構造化文書だけでなく，半構造化文書（フラットな定型構造と，非定形多階層構造から成る文書）や RDB などのデータ形式の壁を越えた処理に，幅広く適用できる仕様となっている。

XQuery の基本仕様書は，「XQuery 1.0: An XML Query Language」であり，本書執筆時点での最新版は 2007 年 1 月 23 日勧告である。

これ以外に，主要な仕様書として，XPath2.0 との共通仕様として，データモデルを定義する「XQuery 1.0 and XPath 2.0 Data Model (XDM)」および，演算子と関数ライブラリの「XQuery 1.0 and XPath 2.0 Functions and Operators」がある。

●図 3-6 XQuery ではミドルウエアに格納した XML データを処理できる。



ただし，Microsoft .NET Framework をフレームワークとして採用する場合は，XQuery ではなく，LINQ を用いる（後述，3.6 参照）。

3.3.2 FLWOR 式

XQuery のクエリ文の記述は，SQL 文に酷似している。その基本は，FLWOR 式（FLWOR Expressions，花を意味する flower と同じ発音）と呼ばれるもので，中間結果への変数バインドや繰返し処理をサポートしている。また，複数文書の結合や，データの再構築にも利用することができる。

FLWOR は，for 句（for clauses），let 句（let clauses），where 句（where clauses），order by 句（order by clauses），return 句（return clauses）の頭文字をとったものである。それぞれの句の機能は次の通りである。

・for 句は，処理を繰り返す。
・let 句は，タプルストリーム（tuple stream）と呼ばれる，変数にバインドした値の組であるタプル群（tuples）の順序あるシーケンスを生成する。
・where 句はオプションで，タプルストリームをフィルタリングする。
・order by 句もオプションで，タプルストリームを並べ替える。
・return 句は，FLWOR 式の結果を構築する。

FLWOR 式の構文は次の通りである（●表 3-5）。構文中の「単一式」の指定に，前述の XPath によるパス式を使う。つまり，XQuery の利用には，パス式の習得が前提条件として必要である。

なお，for 句（for clauses）等の clauses は「節」と呼ぶ方が適切ではあるが，本書では，SQL 文で一般的に知られる，「句」という表記にしている。

●表 3-5 FLWOR 式と各句の基本構文

FLWOR式	(For句あるいはLet句)+ Where句? OrderBy句? return 単一式
For句	for $ 変数名 as シーケンスの型? at $ 変数名? in 単一式 (, $ 変数名 as シーケンスの型? at $ 変数名? in 単一式)*
Let句	let $ 変数名 as シーケンスの型? := 単一式 (, $ 変数名 as シーケンスの型? := 単一式)*
Where句(オプション)	where 単一式
Return句	return 単一式
OrderBy句(オプション)	(order byあるいは stable order by) 単一式 ソート条件 (, 単一式 ソート条件)*

※第2章で触れたが，?，*，+は，出現回数を表す。? はオプション，* は0回以上，+ は1回以上である。
※For句で複数変数が使われる場合，たとえば，for $i in (a, b), $j in (p, r) では，(i,j) 成分という考え方でタプルストリームの生成を期待できる。
※For句とLet句の型宣言はオプションで，「2.5.3 SequenceType Syntax」の文法に従う。宣言した変数の型に，バインドされる値の型が合致しなければ，型エラーになる。
※OrderBy句のソート条件には，(ascending あるいは descending)? (empty (greatest あるいは least))? (collation リテラル URI)? を指定する。

for 句，let 句，return 句

for 句と let 句の役割は，変数に各タプル（tuple）が格納された，タプルストリーム（tuple stream）を生成することである。for 句は繰り返し処理に使う。最もシンプルな記述は，変数と式が 1 対 1 対応のもので，「for $ 変数名 in 単一式」のように記述する。この記述により，in の後に指定した単一式の返すシーケンスが変数にバインドされる。for 句の中で変数に格納される式の値を，バインディング・シーケンス（binding sequence）と呼ぶ。return 句は，タプルストリーム中の各々のタプルを 1 回ずつ評価し，FLWOR 式の結果を構築する。

●リスト 3-4 は，for 句を使ったクエリの記述例である。この例では，in 以降の式に対して処理が繰り返され，<記事名/>，<目次/>という個々のバインドに対してタプルが生成される。return 句は，●リスト 3-5 のように，各々の出力結果を返すことが期待される。

●リスト 3-4　for 句を使ったクエリの例

```
for $article in (<記事名/>, <目次/>)
return <オンライン記事>{$article}</オンライン記事>
```

●リスト 3-5　for 句によるクエリの期待される出力結果

```
<オンライン記事>
  <記事名/>
</オンライン記事>
<オンライン記事>
  <目次/>
</オンライン記事>
```

for 句と異なり，let 句は，繰り返しなしに，式の結果を変数にバインドする。let 句のみを使った●リスト 3-6 の例では，変数$article は，in 以降の式の結果である(<記事名/>，<目次/>)にバインドされる。この例では，for 句が使われていないので，タプルは 1 個生成される。return 句には，●リスト 3-7 のような出力結果を返すことが期待される。

●リスト 3-6　let 句を使ったクエリの例

```
let $article := (<記事名/>, <目次/>)
return <オンライン記事>{$article}</オンライン記事>
```

●リスト 3-7　let 句によるクエリの期待される出力結果

```
<オンライン記事>
  <記事名/>
  <目次/>
</オンライン記事>
```

where 句

Where 句 (where clause) を使った式を，where 式 (where-expression) という。where 句に続けてフィルタリング条件を指定す**ることにより，for 句や let 句で生成された変数バインディングのタプル群をフィルタ**リングする。where 式の評価結果のブール値が true であればタプルが残り，変数バインディングが return 句の実行に使われる。ブール値が false であれば，タプルは破棄される。たとえば，リスト●3-8 のような XML 文書から，「"公開日"属性値が"2008-07-31"である」ということを条件として，属性 (属性値ではない) を取得するクエリ式は，●リスト 3-9 のように記述する。これにより，●リスト 3-10 の出力結果を得ることができる。

●リスト 3-8　サンプル XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名 公開日="2008-07-07">デザインの 30 年</記事名>
    <記事名 公開日="2008-07-31">プログラミングの 30 年</記事名>
  </オンライン記事>
</公開記事>
```

●リスト 3-9　Where 句を使ったクエリの例

```
for $update in /公開記事/オンライン記事/記事名/@公開日
where $update = "2008-07-31"
return <記事名>{$update}</記事名>
```

●リスト 3-10　Where 句によるクエリの期待される出力結果

```
<記事名 公開日="2008-07-31">
```

Order By 句

Order By 句は，タプルストリームの中のタプル群をソートする。ソート条件に ascending を指定すると昇順，descending を指定すると降順となる。規定値は昇順である。●リスト 3-11 は，●リスト 3-8 のサンプル XML 文書から，<記事名>要素の"公開日"属性値の降順にソートして<記事名>の内容を取得するクエリの例である。

●リスト 3-11　Order By 句を使ったクエリの例

```
for $article in 公開記事/オンライン記事/記事名
order by $article/@公開日 descending
return<公開予定リスト>{$article}</公開予定リスト>
```

コンストラクタ

XQuery には，コンストラクタ（constructors）という機能があり，element，attribute，document，text，processing-instruction，comment といったノード群を生成することができる。ダイレクト・コンストラクタ（direct constructors）を使うと，XML ライクな記法でタグを記述してノードを生成することができる。

最も使用頻度が高いのは，要素ノードを生成する要素コンストラクタ（element constructor）である。構文は簡潔で，デリミタとして中括弧{}を使い，その中に生成する要素を記述するというものである。デリミタの中括弧はリテラルなテキスト部分と区別するために必要である。また，XML を生成するのであるから，当然，開始タグと終了タグの対応といった，整形式 XML の規則に従う必要がある。

これまで例にあげたクエリの結果の中には，●リスト 3-5 のように，ただ 1 個のルート要素を持つべき整形式の XML 文書の構造にはならないものがある。ルート要素を生成するには，●リスト 3-12 のように記述する。

なお，要素コンストラクタは自分自身の中に新しい要素ノードを生成する機能であるため，属性や子孫ノードもコピーされるようになる。return 句に指定したコンストラクタを使って，要素の内容テキストや属性値のみを取得したい場合は，後述する fn:string 関数や fn:data 関数を併用する必要がある。

●リスト 3-12 要素コンストラクタを使う場合の基本構文

```
<ルート要素名>{
        for $変数名 in 単一式
        return コンストラクタ
}</ルート要素名>
```

XQuery では，{}は式を囲むために使われるので，要素内容や属性値として{}を使いたい場合は，文字参照{ および}を使う。

3.3.3 演算子と関数

仕様書の利用方法

XPath1.0 では，パス式と関数および演算子は一個の仕様となっていたが，XPath2.0 では，XQuery と共通の仕様「XQuery 1.0 and XPath 2.0 Functions and Operators」が別に策定されている。XQuery 関連仕様のうち「XQuery 1.0 and XPath 2.0 Data Model (XDM) 」は，W3C サイト XPath のコーナからリンクされており，関数と演算子に関する仕様書は，XQuery のコーナに分類されている。W3C サイトの左側のメニュー「XML Query」から，「Contents/Resources for Implementers」をたどると，右側に勧告となった仕様書の一覧「W3C Recommendations!」がある。勧告版の仕様書はここから得られる。本書執筆時点での最新仕様は 2007 年 1 月 23 日勧告版である。

XPath2.0 には，接頭辞 op の付いた Operator（演算子），接頭辞 fn の付いた Function（関数）がある。これらの接頭辞によって，名前空間を使った XML 文書中の接頭辞と区別することができる。仕様書では膨大な演算子と関数が規定されているので，本項では，使いたい関数を探し当てて構文を参考にするための，仕様書の見方について説明する。

たとえば，実務でニーズの高い，部分一致検索を実装する場合を想定してみよう。部分一致検索では，検索キーとして指定した文字列が，検索対象の XML 文書ファイル中の要素の内容テキストに含まれているかどうかを判別し，含まれていれば合致した要素の内容を検索結果として表示し，含まれていなければ「該当するデータはありません」といったメッセージを表示する，といった処理が一般的である。これを実装するために，何か利用できそうな関数はあるだろうか？

検索キーとして指定した文字列が，検索対象の XML データ中に「含まれているかどうか」を判別するのであるから，まず，関数名は「contains」ではないかと予測してみる。そして，判別するのであるから，関数による処理結果は，ブール値で返されるはずであると予測する。ここまで予測できたら，接頭辞 fn の付いた contains 関数を，仕様書の目次の中に探せばよい。すると，「7.5 Functions Based on Substring Matching / 7.5.1 fn:contains」が見つかる。「7.5.1 fn:contains」には，●リスト 3-13 のような構文が記載されている。

●リスト 3-13　W3C が規定する，fn:contains 関数の構文（仕様書からの抜粋）

```
fn:contains($arg1 as xs:string?, $arg2 as xs:string?) as xs:boolean

fn:contains( $arg1    as xs:string?,
$arg2    as xs:string?,
$collation    as xs:string) as xs:boolean
```

●リスト 3-13 の構文では，fn は関数を表す接頭辞，contains が関数名（FunctionName），() 内が関数に与える引数（Argument）である。第 1 引数 arg1 には string 型のデータを与え，第 2 引数 arg2 にも string 型のデータを与える。戻り値は，ブール型である

接頭辞 xs:は，妥当な XML 文書の構造を定義する仕様である XML Schema（後述）の名前空間 http://www.w3.org/2001/XMLSchema に関連付いている（XQuery2.0，2.5.1 Predefined Schema Types）。

collation は文字列の照合順序で，「Unicode Collation Algorithm」で規定されている最低限の合致に該当するかどうかを判別するためのものである。

仕様書では，構文に続いて，サンプルが掲載されている（●リスト 3-14）。この関数の使い方は，次の 2 つの例で，一目瞭然である。最初の例は，「第 1 引数に指定した文字列"tattoo"に，第 2 引数に指定した文字列"t"が含まれているので戻り値は true となる」，というものである。次の例は，「第 1 引数に指定した文字列"tattoo"に，第 2 引数に指定した文字列"ttt"が含まれていないので戻り値は false となる」，というものである。

●リスト 3-14　W3C 仕様書にある，fn:contains 関数の使用例（仕様書からの抜粋）

```
fn:contains ( "tattoo", "t") returns true.
fn:contains ( "tattoo", "ttt") returns false.
```

XQuery 仕様書では，膨大な関数が規定されている。ブラウザ上で読むにはスクロールすることすら難しい。テキスト形式で保存し直して検索するとよいだろう。ここでは，基本的な XQuery の関数と演算子の例を，●表 3-6（次ページ）に挙げてみた。これらの関数の一部は，名前空間なしで XPath1.0 でも利用することができる。

ここで紹介した機能以外にも，XQuery には，if 文を使った条件分岐処理や，関数宣言（function declaration）を用いて処理を関数化して呼び出す機能がある。また，複数データソースを縦断して結合する記述もある（I.1 Joins）。

仕様が膨大であるので，本節ですべてを語ることはできないし，それが本書の主目的でもない。本書を基礎として，仕様書を熟読し，知識を積み上げていくとよいだろう（※1）。

※1：XQuery の演算子を使った処理では，次の処理でアトミック化を実装できる。

算術式（arithmetic expressions），比較式（comparison expressions），関数呼び出し（function calls and returns），型キャスト（cast expressions），ノード群におけるコンストラクタ（constructor expressions for various kinds of nodes），FLWOR 式における order by 句（order by clauses in FLWOR expressions）。算術式，比較式，関数呼び出し，型キャストは，XPath に共通とされている（2.4.2 Atomization）。

アイテム群のシーケンスを取得して，アトミック化されたシーケンスを返すには，前掲の●表 3-6 中の fn:data 関数を使う。

代表的な関数と演算子

●表 3-6 XQuery の関数と演算子の仕様（一部）

関数名，演算子名	主な機能	構文
fn:avg	平均値を算出できる。引数の値の合計を値の数で除算し，そのシーケンスの中での平均値を返す。 引数が空シーケンスの時は，空シーケンスを返す。 引数にxs:untypedAtomic型の値を含まれる場合は，xs:doubleにキャストされる。	fn:avg($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:ceiling	引数の値よりも小さくはない関数の部分でない最も小さい値（負の無限大に近い）を返す。 引数の型がnumeric型（xs:float, xs:double, xs:decimal, xs:integer）の一つの場合，結果の型は，引数のに同じ。引数がnumeric型の一種であれば，numeric型をベースとするインスタンスになる。	fn:ceiling($arg as numeric?) as numeric?
fn:concat	，（カンマ）区切りで指定した文字列を連結できる。 2個以上のxs:anyAtomicTypeの引数を，xs:stringにキャストし，連結したxs:stringを返す。 引数が空であれば，0文字として扱う。 連結する文字によっては，正規化が必要になる。構文は，return normalize-unicode(concat($第1引数, $第2引数))	fn:concat($arg1 as xs:anyAtomicType?, $arg2 as xs:anyAtomicType?, …) as xs:string
fn:contains	第1引数の文字列内に，（文字の並び順はそのままで）少なくとも最低限一致する第2引数の文字列が含まれているかどうかを判別し，その結果をブール値で返す。部分一致検索に利用できる。	fn:contains($arg1 as xs:string?, $arg2 as xs:string?) as xs:boolean fn:contains($arg1 as xs:string?, $arg2 as xs:string?, $collation as xs:string) as xs:boolean
fn:count	引数の値の中のアイテム群のアイテム数を返す。空の場合は0を返す。ノードの個数を取得したい場合に利用できる。	fn:count($arg as item()*) as xs:integer
fn:data	引数に指定されたアイテム群のシーケンスを取得して，アトミック値のシーケンスを返す。属性値を取得したり，子要素すべての内容テキストをシーケンスとして取得するために利用できる。	fn:data($arg as item()*) as xs:anyAtomicType*
fn:distinct-values	引数の値から重複する集合の値を削除した結果のシーケンスを返す。 xs:untypedAtomic型の値はxs:string型であるとみなして処理される。重複しない要素の内容を取得したい場合に有効。	fn:distinct-values($arg as xs:anyAtomicType*) as xs:anyAtomicType* fn:distinct-values($arg as xs:anyAtomicType*, $collation as xs:string) as xs:anyAtomicType*
fn:empty	引数の値が空のシーケンスであればtrueを返し，そうでなければfalseを返す。要素や属性の有無を調べることができる。	fn:empty($arg as item()*) as xs:boolean
fn:false	値がfalseであるブール値を返す。 xs:boolean("0")に等しい。 fn:true() as xs:booleanでは，trueつまりxs:boolean("1")。	fn:false() as xs:boolean
fn:floor	一言で言えば，端数を切り捨てる。引数より大きくはない整数の最大値を返す。 Round関数が，引数に1.5を指定すれば2を返すのに対し，floor関数は1を返す。 引数と結果の型については，fn:ceilingに同じ。	fn:floor($arg as numeric?) as numeric?
fn:id	引数に指定された1個以上のIDREF値群の値に，合致するID値がある要素ノード群のシーケンスを返す。	fn:id($arg as xs:string*) as element()* fn:id($arg as xs:string*, $node as node()) as element()*
fn:last	動的コンテキストから，コンテキスト・サイズを返す。つまり，最後のノードを特定することができる。	fn:last() as xs:integer
fn:local-name	引数として指定した名前のローカル部分を返す。つまり，ローカル名を取得できる。 xs:stringの引数の名前のローカル部分を返す。引数が省略されている場合，デフォルトはコンテキスト・アイテム。	fn:local-name() as xs:string fn:local-name($arg as node()?) as xs:string
fn:local-name-from-QName	引数のローカル部分を表すxs:NCName（No Colon Name，コロンの付いていない名前）を返す。引数が空シーケンスの時は空シーケンスを返す。	fn:local-name-from-QName($arg as xs:QName?) as xs:NCName?
fn:max	最大値を取得するために使う。引数に与えられたシーケンスの中の最大値のアイテムを選択する。	fn:max($arg as xs:anyAtomicType*) as xs:anyAtomicType? fn:max($arg as xs:anyAtomicType*, $collation as string) as xs:anyAtomicType?
fn:min	最小値を取得するために使う。引数に与えられたシーケンスの中の最小値のアイテムを選択する。	fn:min($arg as xs:anyAtomicType*) as xs:anyAtomicType? fn:min($arg as xs:anyAtomicType*, $collation as string) as xs:anyAtomicType?
fn:namespace-uri-from-QName	xs:string型の引数の名前空間URIを返す。引数が空シーケンスであれば，空シーケンスを返す。引数に名前空間がなければ，0の文字列を返す。	fn:namespace-uri-from-QName($arg as xs:QName?) as xs:anyURI?

fn:not	第1引数の文字列内に，第2引数の文字列が含まれていないかどうかを判別結果を，ブール値で返す。つまり，fn:contains関数とは逆で，含まれていないものを抽出する。	fn:not($arg as item()*) as xs:boolean
fn:number	引数を示す値を返すか，引数が指定されていない場合は，xs:doubleに変換された，アトミック化の後のコンテキストアイテムを返す。	fn:number() as xs:double fn:number($arg as xs:anyAtomicType?) as xs:double
fn:position	動的コンテキストから，コンテキスト・ポジションを返す。ポジションを指定してノードを選択するために利用できる。	fn:position() as xs:integer
fn:string	文字列値として表わされる，アイテムの引数の値を返す。属性付き要素で，要素の内容テキストのみを取得することができる。	fn:string() as xs:string fn:string($arg as item()?) as xs:string
fn:string-length	文字列中の文字数の整数値を返す。引数の値が空のシーケンスの場合は0を返す。テキストノードの文字数を取得することができる。	fn:string-length() as xs:integer fn:string-length($arg as xs:string?) as xs:integer
fn:substring	文字列の始まりの位置を指定してデータを取得するために利用できる。第3引数で指定される文字数の，第2引数で指定した開始ロケーションのポジションから始まる，第1引数に指定した対象文字列の部分を返す。	fn:substring($sourceString as xs:string?, $startingLoc as xs:double) as xs:string fn:substring($sourceString as xs:string?, $startingLoc as xs:double, $length as xs:double) as xs:string
fn:sum	引数の値を加算した値を返す。	fn:sum($arg as xs:anyAtomicType*) as xs:anyAtomicType fn:sum($arg as xs:anyAtomicType*, $zero as xs:anyAtomicType?) as xs:anyAtomicType?
fn:round	いわゆる「まるめ」。引数に最も近い整数の値を返す。 引数と結果の型については，fn:ceilingに同じ。	fn:round($arg as numeric?) as numeric?
op:union	重複するノード群を除いた2つの引数の和集合のノードのシーケンスを構築し，文書順で返す。	op:union($parameter1 as node()*, $parameter2 as node()*) as node()*
op:intersect	重複するノード群を除いた2つの引数の積集合のノードのシーケンスを構築し，文書順で返す。	op:intersect($parameter1 as node()*, $parameter2 as node()*) as node()*
op:except	第1引数にあって第2引数にない差集合のノードのシーケンスを構築し，文書順で返す。	op:except($parameter1 as node()*, $parameter2 as node()*) as node()*

実務で利用する場合の注意事項

ここまでの例としてあげた，ごく簡易なクエリ式であっても，XML データを格納するミドルウエアによっては，サポート状況が異なり，利用可能な関数も構文も異なる。また，XML データに接続するコード自体も異なる。残念ながら，実務においては，W3C 仕様が 100%標準という状況ではないので，学習方法としては，まずは W3C 仕様の基本を押さえた後，採用予定のミドルウエアの仕様に則って実際にコードを記述してテストし，W3C の構文と比較しながら覚えていく方が合理的である。

たとえば，SQL Server 2005（XQuery1.0 ワーキングドラフト 2005 年 9 月 15 日版をサポート，●リスト 3-9，●リスト 3-11，および●表 3-6 の大半は実行可能）に，前掲の●リスト 3-8 のような XML 文書ファイルを格納し，XQuery を使ってクエリを返す例を，●リスト 3-15 に示す。これは「article」SQL Server データベースの「articleTable」テーブルの，「xmlColumn」列に格納した XML データから，特定の語句，ここでは「デザインの 30 年」を指定してクエリを返す処理である。

●リスト 3-15 SQL Server 2005 に XML データを格納して XQuery を使った例

```
SELECT xmlColumn.query('
for $articles in /公開記事
return <公開記事>{
                for $articleName in 公開記事/descendant::記事名
                where $articleName[contains(.,"デザインの 30 年")]
                return <検索結果>{$articleName}</検索結果>
}</公開記事>')
FROM article.dbo.articleTable
```

本節で取り上げた以外にも，勧告あるいはワーキングドラフト段階の XQuery1.0 および XPath2.0 関連の仕様は多々あり，それらが頻繁にバージョンアップされている（※2）。

バージョンアップ差異をリアルタイムで把握するよりも，フレームワークやミドルウエアのサポートする仕様を優先して習得する方が現実的である。基本的には，XPath2.0 で規定されているパス式の構文，XQuery 1.0 の句と基本構文，XQuery 1.0 and XPath 2.0 Functions and Operators の中で業務での使用頻度が高いと思われる関数をおさえておき，コーディングで必要になった時に，使い方を仕様書で調べればよい。

※2：本書執筆後の 2009 年 4 月 21 日に，以下の 8 本が，W3C Proposed Edited Recommendation となっている。

XSL Transformations (XSLT) Version 2.0 (Second Edition)

XQuery 1.0: An XML Query Language (Second Edition)

XML Syntax for XQuery 1.0 (XQueryX) (Second Edition)

XML Path Language (XPath) 2.0 (Second Edition)

XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition)

XQuery 1.0 and XPath 2.0 Formal Semantics (Second Edition)

XQuery 1.0 and XPath 2.0 Formal Semantics (Second Edition)

XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)

3.4 DOM によるデータ処理

3.4.1 DOM の概要

DOM の三大機能

ここ数年，サーバに蓄積されている既存の XML 文書ファイルや，Web サービス提供先から取得した XML データを処理するために，DOM（Document Object Model）の用いられるケースが増えているとも聞く。XML データは，独自開発した Web アプリケーション，ローカルアプリケーション，モバイルアプリケーション，あるいは Macromedia Flash などで幅広く利用できるため，Ajax（※3）の爆発的流行もあいまって，DOM 利用者の裾野は広がっているそうである（ただし，筆者自身は実務では後述の LINQ を使っており，DOM はほとんど使っていない。また，Microsoft Silverlight で XML データを処理する際にも LINQ を用いる）。

DOM は，プログラム可能な XML および HTML のインタフェース定義を目的として策定された仕様であり，いわば，HTML および XML 文書対応の API（application programming interface）である。DOM は，プラットフォームや開発言語には非依存の仕様であって，JavaScript，VBScript，ActionScript，ECMAScript，C#，VisualBasic など，さまざまな言語によるプログラムから扱うことができる。そのメソッドやプロパティは，XML 文書中の内容・構造・スタイルに動的にアクセスしたり，それらを更新するために用いられる。

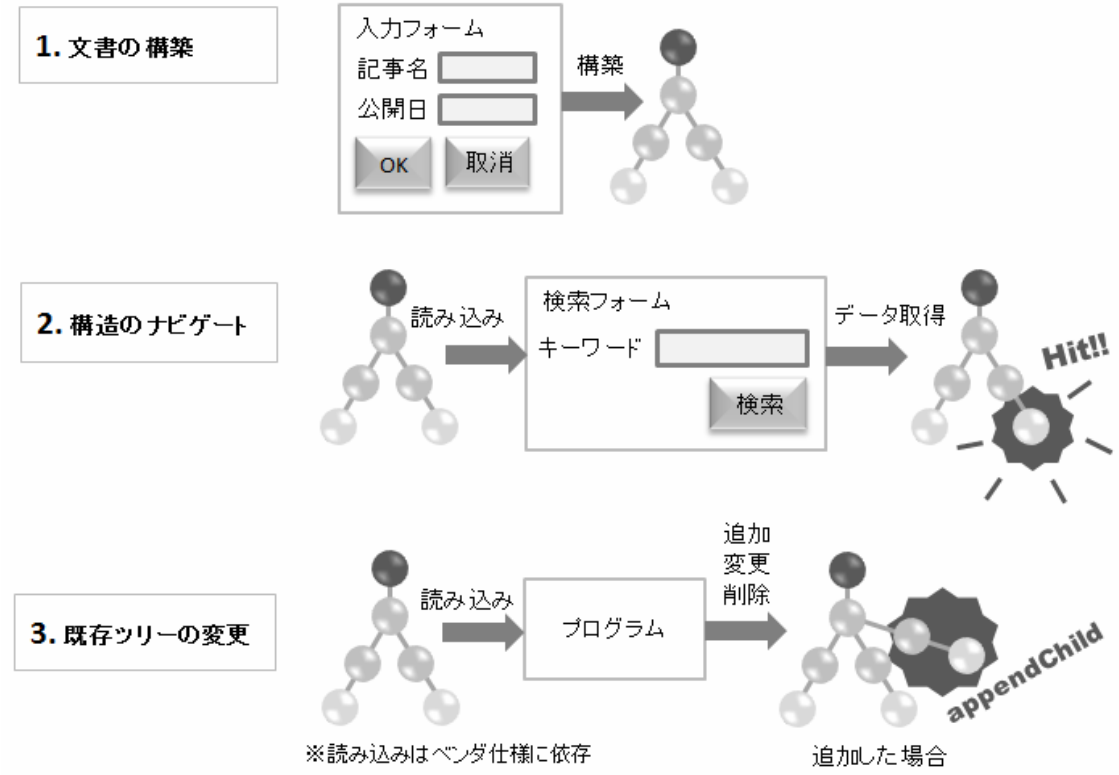
DOM を用いて実装できる主な機能は，大きく分けて 3 つある（●図 3-7）。

1 つは，文書の構築である。たとえば，Web 上のフォームなどから入力されたテキストデータにタグを付け，新しく XML ツリーを構築することができる。また，プログラムによる処理の利便性のために，元の XML ツリーとは異なる構造の XML ツリーをメモリ上に一時的に展開することができる。

2 つ目は，構造のナビゲートである。DOM を用いてノードリストやデータを取得することができ，検索処理等を実装することができる。処理対象ノードの指定に，XPath を併用することもある。

3 つ目は，既存文書の変更である。XML 文書ファイル中の任意のデータを修正したり，あるいはノードごと除去したり，ノードを挿入，追加して，更新することができる。要素やデータの追加･変更･削除は，実務ではニーズの高い機能である。

●図 3-7 DOM の三大機能



※3：Ajax は，HTTP 通信機能を利用した非同期通信により，ページのリロードを伴わずに Web サーバへのリクエストを実装する手段である。クライアント―サーバ間でやりとりされる XML データを動的に処理するために DOM を使う。W3C では，クライアント―サーバ間のデータ転送のためにクライアントスクリプトの API を定義する「The XMLHttpRequest Object」が発表されている。XMLHttpRequest オブジェクトは，XML を含むテキストベースの形式をサポートする。

http://www.w3.org/TR/XMLHttpRequest/，http://www.w3.org/TR/XMLHttpRequest2/

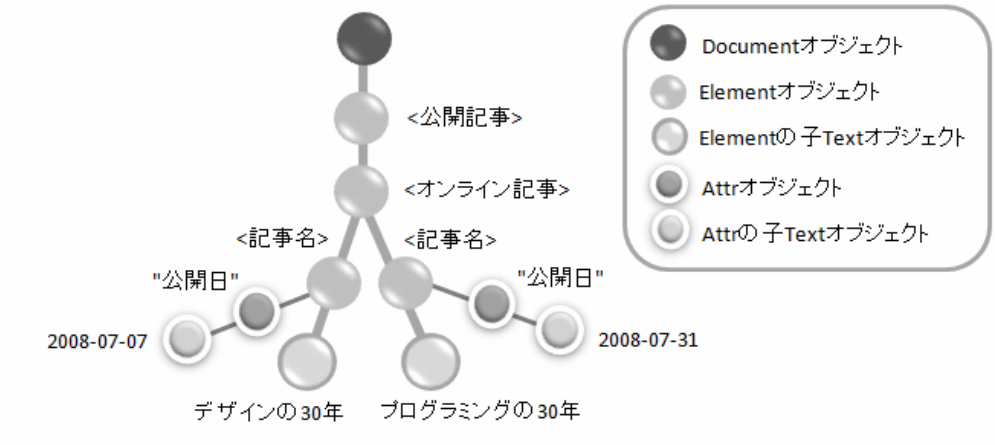
構造モデル

DOM では，モデル化する文書に酷似した論理的構造を持つ木を扱う。この木構造を表現するものを「構造モデル」と呼ぶ。たとえば、前掲の●リスト 3-8 の XML 文書は、DOM では●図 3-8 のように表現される。

●リスト 3-8（再掲載） サンプル XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名 公開日="2008-07-07">デザインの 30 年</記事名>
    <記事名 公開日="2008-07-31">プログラミングの 30 年</記事名>
  </オンライン記事>
</公開記事>
```

●図 3-8 リスト 3-8 の DOM による表現



DOM の構造モデルでは，操作対象とする文書を，階層構造を持つオブジェクトとして取り扱う。ノード群には，種々の型の子ノード群（child nodes）を持つものと，階層下に何も持たないリーフノード群（leaf nodes）がある。構造モデルのオブジェクトは●表 3-7 の通りである。

●表 3-7 構造モデル（DOM Level 1，Level2 「The DOM Structure Model」）

親のオブジェクト	子として持つことができるオブジェクト
Document	Element（ルート要素），ProcessingInstruction，Comment，DocumentType
DocumentFragment	Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference
EntityReference	Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference
Element	Element，Text，Comment，ProcessingInstruction，CDATASection，EntityReference
Attr	Text，EntityReference
Entity	Element，ProcessingInstruction，Comment，Text，CDATASection，EntityReference
DocumentType	なし
ProcessingInstruction	なし
Comment	なし
Text	なし
CDATASection	なし
Notation	なし

3.4.2 仕様の現状

各仕様書の入手方法

DOM Level 1 は，1998 年 10 月 1 日に W3C 勧告となっており，2000 年 5 月 1 日に「Errata（訂正）」が発表されている（http://www.w3.org/DOM/updates/REC-DOM-Level-1-19981001-errata.html）。

DOM Level 1 仕様書は，Core と HTML の 2 つの節に分かれており，XML を扱う上で必要なのは、DOM Level1 コアである。コアは，構造化文書を表現する基本的なインタフェースの低水準セットを提供し，XML 文書を表現する拡張インタフェースを定義している。仕様書は，pdf，zip，txt 等の形式で公開されている。解凍した zip ファイルでは，cover.html がトップページである。

その後，2000 年 11 月 13 日に，DOM Level 2 に関する仕様が，一斉に勧告になっている。その内訳は，次の通り。DOM Level 2 Core，DOM Level 2 Views，DOM Level 2 Events，DOM Level 2 Style，DOM Level 2 Traversal and Range。それぞれ「Errata」が発表されている。なお，HTML DOM については，2003 年 1 月 9 日に，DOM Level 2 HTML が勧告となっている。

DOM Level 2 の仕様書は，一式圧縮の zip ファイルを入手することができる。core/Overview.html から参照するとよい。

DOM Level 1 からの変更点は，「Appendix A: Changes」A.1: Changes between DOM Level 1 Core and DOM Level 2 Core」に記載されている。最大の変更点は，名前空間がサポートされたことである（1.1.8. XML Namespaces）。これに伴い，インタフェースも多数追加されている（後述の●表 3-8 参照）。

「DOM Level 2 Core」を基盤として，Views（XML の表示），Events（イベントの定義），Style（スタイルの定義），Traversal and Range（走査方法と範囲指定）といった，処理目的別の関連仕様も策定されている。XML データ処理に関わりが深いのは，ツリーを走査する際の対象ノードやポジションの表現を規定する「Traversal and Range」で，動的な文書構造の走査について定義した「Traversal」，文書内容の階層を特定してプログラムやスクリプトから扱う機能について定義した「Range」から成る。

さらに，その後，2003 年 1 月 27 日に DOM Level 3 Validation が，2004 年 4 月 7 日に，DOM Level 3 Core，DOM Level 3 Load and Save が勧告となっている。

DOM の仕様は膨大であり，特に Level 2 と Level 3 の間での追加変更はいちじるしい。本書でそのすべてを取り上げることはできない。また，開発実務においては開発フレームワークのサポートする独自仕様の DOM を用いるケースも少なくない。そこで本節では，DOM Level1 コアを基本に，DOM Level 2 Core の変更点を加味し，DOM プログラミングに取り組むために必要な基礎知識を選んで解説する。

なお，本書執筆後，W3C の Web サイトの構成が変更となり，現在，DOM の各仕様書は，「Web Design and Applications＞Scripting and Ajax 」コーナから入手することができる。

3.4.3 DOM のメソッド

IDL 定義の見方

DOM の各オブジェクトにはインタフェースがあり，各インタフェースには属性とメソッドがある。実践で DOM を使うということは，プログラムコードの中で，インタフェース別の属性とメソッドを利用することに他ならない。DOM のインタフェースは、基本インタフェースと拡張インタフェースから成る。

基本インタフェースは，DOMImplementation，DocumentFragment，DOMImplementation，DocumentFragment，Document，Node，NodeList，NamedNodeMap，CharacterData，Attr，Element，Text，Comment，拡張インタフェースは、CDATASection，DocumentType，Notation，Entity，

EntityReference, ProcessingInstruction である。

仕様書では，これらの各インタフェース別に，IDL 定義（Interface Definition Language，インタフェース定義言語）が既定されている。利用目的に合ったメソッドやプロパティは，次の手順で知ることができる。例えば「タグ名をキーにして内容を抽出する」処理について考えてみよう。タグ名は tagname であるから，この言葉をキーとして，仕様書内を検索する。すると，Document インターフェイスに getElementsByTagName メソッドがあることが分かる。このメソッドの IDL 定義を見ると，●リスト 3-17 の通りである。getElementsByTagName のプロパティに DOMString として tagname を指定する仕様となっている。戻り値は NodeList オブジェクトである。

●リスト 3-17　Document インターフェイス/getElementsByTagName メソッドの IDL 定義の一部分（仕様書からの抜粋）

```
interface Document : Node {
    readonly attribute DocumentType        doctype;
    readonly attribute DOMImplementation   implementation;
    readonly attribute Element             documentElement;
                                           ～略～
    NodeList          getElementsByTagName(in DOMString tagname);
};
```

つまり，指定するタグ名を持つ要素のノードリストを取得するには，document インタフェースの getElementsByTagName メソッドを使い，Document.getElementsByTagName(タグ名);のように記述すればよいということが分かる。戻り値として，指定するタグ名に合致する全ての要素を含む，新しい NodeList オブジェクトが得られる。メソッドの使い方は，基本的に●リスト 3-18 のようになる。但し，実コードは，採用するプログラム言語によって異なる。

●リスト 3-18　メソッドの使い方

戻り値を格納するオブジェクト変数名 = インタフェース名.メソッド名(パラメータ);

メソッド一覧

DOM Level1 コアおよび DOM Level 2 で追加されたメソッドの一覧を●表 3-8 にまとめた。これらは，各インタフェースの属性とメソッド、メソッドの記述とパラメータについての一覧である。DOMImplementation 等は，インタフェースの種類である。各メソッドにはパラメータがある。採用する開発･実行環境によって構文の異なる場合があるので，表には基本的な記述を掲載している。なお，表中の「拡張インタフェース」は，DOM Level1 コアの一部をなす仕様で，HTML DOM には無関係である。

●表 3-8 DOM Level1 コア，DOM Level 2 メソッド一覧 （「1.0」は Level1 のみ，「2.0」は Level2 での追加）

■基本インタフェース（Fundamental Interfaces ）

DOMImplementation			
特定のインスタンスに依存しない、独立した機能を提供する。			
メソッド	**createDocument**	文書要素の型で指定されたXMLのDocumentオブジェクトを生成する。	2.0
DOMImplementation.createDocument(namespaceURI, qualifiedName, doctype);			
パラメータ	namespaceURI	生成される文書要素のnamespaceURI。	
	qualifiedName	生成される文書要素の修飾名。	
	doctype of type	生成される文書の型あるいはnull。	
戻り値		新規Documentオブジェクト。	
メソッド	**createDocumentType**	空のDocumentTypeノードを生成する。	2.0
DOMImplementation.createDocumentType(qualifiedName, DOMString publicId, DOMString systemId);			
パラメータ	qualifiedName	生成される文書要素の修飾名。	
	publicId	外部サブセットの公開識別子。	
	systemId	外部サブセットのシステム識別子。	
戻り値		新規DocumentTypeオブジェクト。使われていない場合はnull。	
メソッド	**hasFeature**	指定する機能がDOMで実装されているかどうかを調べる	
DOMImplementation.hasFeature(feature,version);			
パラメータ	feature	テストする機能。Level1ではHTMLとXML（大文字小文字不問）	
	version	テストする機能のバージョン番号。	
戻り値		ブール値。実装されている場合true、実装されていない場合false。	
DocumentFragment			
新規生成された文書の断片を保持する。操作対象の親代わりとして移動させることによって、部分抽出、ノードの移動、子ノードの挿入などを実装する。Nodeからの継承。			
Document			
XML文書のツリーのルートを表す。文書中のデータにアクセスする基点となる。Nodeからの継承。			
属性	**doctype**	文書型宣言。これがない場合はnullを返す。	
	implementation	文書を処理するDOMImplementationオブジェクト。	
	documentElement	文書のルート要素の子ノードへの直接アクセスを許可する。	
メソッド	**createElement**	指定した型の要素を生成する。	
Document.createElement(tagName);			
パラメータ	tagName	インスタンス化する要素型。つまり要素名。	
戻り値		新規Elementオブジェクト。	
メソッド	**createElementNS**	修飾名と名前空間URIを指定して要素を生成する。	2.0
Document.createElementNS(namespaceURI,qualifiedName);			
パラメータ	namespaceURI	生成する要素の名前空間URI。	
	qualifiedName	インスタンス化される要素型の修飾名。	
戻り値		新規Elementオブジェクト。	
メソッド	**createDocumentFragment**	文書の断片のオブジェクトを生成する 。	
Document.createDocumentFragment();			
戻り値		新規DocumentFragmentオブジェクト。	
メソッド	**createTextNode**	指定した文字列からテキストノードを生成する。	
Document.createTextNode(data);			
パラメータ	data	テキストノードとなるデータ。	
戻り値		新規Textオブジェクト。	
メソッド	**createComment**	指定した文字列からコメントノードを生成する。	
Document.createComment(data);			
パラメータ	data	コメントノードとなるデータ。	
戻り値		新規Commentオブジェクト。	
メソッド	**createCDATASection**	指定した文字列からCDATASectionオブジェクトを生成する。	
Document.createCDATASection(data);			
パラメータ	data	CDATAセクションの内容となるデータ。	
戻り値		新規CDATASectionオブジェクト。	
メソッド	**createProcessingInstruction**	指定した名前とデータ文字列から処理命令ノードを生成する。	
Document.createProcessingInstruction(target, data);			
パラメータ	target	処理命令のターゲット部分。（<?の後続の文字列）	
	data	データ部分。（targetの後続の文字列）	
戻り値		新規ProcessingInstructionオブジェクト。	
メソッド	**createAttribute**	与えられた名前のAttrを生成する。Attrインスタンスは、setAttributeメソッドを使って要素にセットすることができる。	
Document.createAttribute(name);			
パラメータ	name	属性の名前。	
戻り値		新規Attrオブジェクト	
メソッド	**createAttributeNS**	修飾名と名前空間URIを指定して属性を生成する。	2.0
Document.createAttributeNS(namespaceURI,qualifiedName);			
パラメータ	namespaceURI	生成する属性の名前空間URI。	
	qualifiedName	生成する属性の修飾名。	
戻り値		新規Attrオブジェクト	
メソッド	**createEntityReference**	実体参照オブジェクトを生成する。	
Document.createEntityReference(name);			
パラメータ	name	参照する実体の名前。	
戻り値		新規EntityReferenceオブジェクト。	

メソッド	**getElementById**	elementIdに指定されたIDを持つElementを返す。	2.0
Document.getElementById(elementId);			
パラメータ	elementId	要素のユニークID名。	
戻り値		合致する要素。	
メソッド	**getElementsByTagName**	与えられたタグ名を持つ全ての要素のノードリストを返す。	
Document.getElementsByTagName(tagname);			
パラメータ	tagname	合致するタグの名前。*は全てのタグに合致する。	
戻り値		合致する全ての要素を含む新規NodeListオブジェクト。	
メソッド	**getElementsByTagNameNS**	修飾名と名前空間URIを指定して全ての要素のノードリストを返す。	2.0
Document.getElementsByTagNameNS(namespaceURI,localName);			
パラメータ	namespaceURI	合致する要素の名前空間URI。*は全ての名前空間に合致する。	
	localName	合致する要素のローカル名。*は全てのローカル名に合致する。	
戻り値		合致する全ての要素を含む新規NodeListオブジェクト。	
Node			
文書ツリーの中の単一のノードを表す。子を扱うためのメソッドを実装する。			
属性	**nodeName**	ノードの名前。	
	nodeValue	ノードの値。	
	nodeType	オブジェクトの型を表すコード。	
	parentNode	ノードの親。	
	childNodes	ノードの全ての子を含むノードリスト。	
	firstChild	ノードの最初の子。	
	lastChild	ノードの最後の子。	
	previousSibling	指定するノードの直前のノード。	
	nextSibling	指定するノードの後続のノード。	
	attributes	ノードの属性リストであるNamedNodeMapオブジェクト。	
	ownerDocument	【1.0】指定するノードに関連する文書オブジェクト。新規ノード群生成のために使われた文書オブジェクト。 【2.0】一部変更。	2.0
	localName	ノードの修飾名のローカル部分	
	namespaceURI	ノードの名前空間URI。	
	prefix	ノードの名前空間接頭辞。	
メソッド	**insertBefore**	既存の子ノードrefChild の前にnewChildノードを挿入する。refChildがnullの場合は、子のリストの最後にnewChildを挿入する。	
Node.insertBefore(newChild, refChild)			
パラメータ	newChild	挿入されるノード。	
	refChild	参照されるノード(新規ノード挿入位置の前にあるノード)。	
戻り値		挿入されたノード。	
メソッド	**replaceChild**	子のリストの中でoldChildをnewChildで置き換え、oldChildノードを返す。	
Node.replaceChild(newChild,oldChild);			
パラメータ	newChild	子のリストの中に置かれた新規ノード。	
	oldChild	リストの中で置き換えられるノード。	
戻り値		置き換えられたノード。	
メソッド	**removeChild**	子のリストからoldChildで示される子ノードを取り除き、これを返す。	
Node.removeChild(oldChild);			
パラメータ	oldChild	リムーブするノード(※注1参照)。	
戻り値		リムーブしたノード。	
メソッド	**appendChild**	ノードの子のリストの最後にnewChildを追加する。	
Node.appendChild(newChild);			
パラメータ	newChild	追加するノード。	
戻り値		追加されたノード。	
メソッド	**hasAttributes**	要素ノードが属性を持っているかどうかを調べる。	2.0
Node.hasAttributes();			
戻り値		属性を持っている場合はtrue、持っていない場合はfalse。	
メソッド	**hasChildNodes**	ノードが子を持っているかどうかを調べる。	
Node.hasChildNodes();			
戻り値		子を持っている場合はtrue、持っていない場合はfalse。	
メソッド	**cloneNode**	ノードの複製を返す。ノード群を複製するコンストラクタ。	
Node.cloneNode(deep);			
パラメータ	deep	trueの場合は、指定したノードの下にサブツリーを複製し、falseの場合は、ノード自身および(ノードが要素の場合は)その属性も複製する。	
戻り値		複製されたノード。	
メソッド	**isSupported**	指定したノードで機能がサポートされているかどうかを調べる。	2.0
Node.isSupported(feature,version);			
パラメータ	feature	テストする機能。Level1ではHTMLとXML(大文字小文字不問)	
	version	テストする機能のバージョン番号。	
戻り値		ブール値。実装されている場合true、実装されていない場合false。	
メソッド	**normalize**	【2.0】変更。全てのテキストノードを、隣接するテキストノードがないように、単一のノードに正規化する。	2.0
Node.normalize();			
NodeList			
順序のあるノード群の集合を表す。ノードリストのインデックス(添字)は0から始まる。※※注2参照。			
メソッド	**item**	指定するインデックス番号にあたるノード集合中のノードオブジェクトを返す。	
NodeList.item(index);			
パラメータ	index	ノード集合の中のインデックス番号。	
戻り値		ノードリストの中のインデックス番号を持つノード。	
属性	**length**	リストのノードの数。添字は0からlength-1までの範囲の値。	

NamedNodeMap			
名前でアクセスできるノードの集合を表す。NodeListからの継承はない。			
メソッド	**getNamedItem**	名前によって指定されたノードを取得する。	
NamedNodeMap.getNamedItem(name);			
パラメータ	name	取得するノードの名前。	
戻り値		指定された名前のノード。これがない場合はnull。	
メソッド	**getNamedItemNS**	ローカル名と名前空間URIによって指定されたノードを取得する。	2.0
NamedNodeMap.getNamedItemNS(namespaceURI, localName);			
パラメータ	namespaceURI	取得するノードの名前空間URI。	
	localName	取得するノードのローカル名。	
戻り値		ローカル名と名前空間URIによって指定されたノード。これがない場合はnull。	
メソッド	**setNamedItem**	nodeName属性を使っている属性をセットする。	
NamedNodeMap.setNamedItem(arg);			
パラメータ	arg	ノードマップにストアするノード。	
戻り値		既存ノードを新規ノードが同じ名前で置き換えた場合は既存のノード、それ以外の場合はnull。	
メソッド	**setNamedItemNS**	namespaceURIとlocalNameを使ってノードをセットする。	2.0
NamedNodeMap.setNamedItemNS(in Node arg);			
パラメータ	arg	ノードマップにストアするノード。	
戻り値		既存ノードを新規ノードが同じ名前で置き換えた場合は既存のノード、それ以外の場合はnull。	
メソッド	**removeNamedItem**	名前で指定されたノードをリムーブする。	
NamedNodeMap.removeNamedItem(name);			
パラメータ	name	リムーブするノードの名前。	
戻り値		マップからリムーブされたノード。指定する名前がない場合はnull。	
メソッド	**removeNamedItemNS**	ローカル名と名前空間URIによって指定されたノードをリムーブする。	2.0
NamedNodeMap.removeNamedItemNS(namespaceURI,localName);			
パラメータ	namespaceURI	リムーブするノードの名前空間URI。	
	localName	取得するノードのローカル名。	
戻り値		ローカル名と名前空間URIのあるマップからリムーブされたノード。これがない場合はnull。	
メソッド	**item**	マップ内のインデックス番号を持つアイテムを返す。	
NamedNodeMap.item(index);			
パラメータ	index	マップ内のインデックス番号。	
戻り値		指定するインデックス番号を持つマップ内のノード。	
属性	**length**	マップ内のノードの数。	
CharacterData			
DOM内の文字データにアクセスするための機能を提供する。Nodeからの継承。			
属性	**data**	処理対象となるノードの文字データ。	
	length	substringDataメソッドを使って利用可能な文字数。	
メソッド	**substringData**	ノードから一連のデータを抽出する。	
CharacterData.substringData(offset,count);			
パラメータ	offset	抽出する文字列の開始位置。	
	count	抽出する文字の数。	
戻り値		指定された文字列。	
メソッド	**appendData**	ノードの文字データの後に文字列を追加する。	
CharacterData.appendData(arg);			
パラメータ	arg	追加する文字列。	
メソッド	**insertData**	指定した位置に文字列を挿入する	
CharacterData.insertData(offset,arg);			
パラメータ	offset	挿入する文字の位置。	
	arg	挿入する文字列。	
メソッド	**deleteData**	ノードから指定する範囲の文字列をリムーブする。	
CharacterData.deleteData(offset,count);			
パラメータ	offset	文字をリムーブする位置。	
	count	削除する文字数。	
メソッド	**replaceData**	指定する文字で、指定するキャラクタオフセットで始まるキャラクタを置き換える。	
CharacterData.replaceData(offset,count,arg);			
パラメータ	offset	置換開始位置。	
	count	置換する文字数。	
	arg	置換する文字列。	
Attr			
要素オブジェクトの属性を表す。Nodeからの継承。			
属性	**name**	属性名を返す。	
	specified	元の文書の中で明らかに属性値がある場合はtrue，そうでなければfalse。	
	value	属性値を文字列として返す。	
	ownerElement	属性付き，あるいは属性が使われていない場合はnullの，Elementノード。	2.0

Element			
要素へのアクセスと操作を実装する。Nodeからの継承。			
属性	**tagName**	要素の名前。HTML DOMでは大文字が返されるが、XMLでは大文字小文字が区別されるので注意する。	
メソッド	**getAttribute**	指定した名前の属性値を取得する。	
Element.getAttribute(name);			
パラメータ	name	取得する属性の名前。	
戻り値		文字列としてのAttr値。属性値やデフォルト値がない場合は空の文字列。	
メソッド	**getAttributeNS**	ローカル名と名前空間URIで指定した属性値を取得する。	2.0
Element.getAttributeNS(namespaceURI,localName);			
パラメータ	namespaceURI	取得する属性の名前空間URI。	
	localName	取得する属性のローカル名。	
戻り値		文字列としてのAttr値。属性値やデフォルト値がない場合は空の文字列。	
メソッド	**setAttribute**	新しい属性を追加する。	
Element.setAttribute(name,value);			
パラメータ	name	生成または変更する属性の名前。	
	value	セットする文字列値。	
メソッド	**setAttributeNS**	新規属性を追加する。要素に同じローカル名と名前空間URIを持つ属性がすでにある場合は，修飾名の接頭辞は変更され，valueパラメータでvalueが変更される。	2.0
Element.setAttributeNS(in DOMString namespaceURI,qualifiedName,value);			
	namespaceURI	生成後の属性の名前空間URI。	
パラメータ	localName	生成後の属性のローカル名。	
	value	セットする文字列値。	
メソッド	**removeAttribute**	指定する名前の属性をリムーブする。	
Element.removeAttribute(name);			
パラメータ	name	リムーブする属性の名前。	
メソッド	**removeAttributeNS**	ローカル名と名前空間URIで指定される属性をリムーブする。	2.0
Element.removeAttributeNS(namespaceURI,localName);			
パラメータ	namespaceURI	リムーブする属性の名前空間URI。	
	localName	リムーブする属性のローカル名。	
パラメータ	name	リムーブする属性の名前。	
メソッド	**getAttributeNode**	名前によってAttrノードを取得する	
Element.getAttributeNode(name);			
パラメータ	name	取得する属性名。	
戻り値		指定する属性名を持つAttrノード。これがない場合はnull。	
メソッド	**getAttributeNodeNS**	ローカル名と名前空間URIで指定したAttrノードを取得する	2.0
Element.getAttributeNodeNS(namespaceURI,localName);			
パラメータ	namespaceURI	取得する属性の名前空間URI。	
	localName	取得する属性のローカル名。	
戻り値		指定する属性名を持つAttrノード。これがない場合はnull。	
メソッド	**setAttributeNode**	新しい属性を追加する。	
Element.setAttributeNode(newAttr);			
パラメータ	newAttr	属性リストに追加するAttrノード。	
戻り値		newAttr属性が既存の属性を同じ名前で置き換える場合は、既存のAttrノード。それ以外の場合はnull。	
メソッド	**setAttributeNodeNS**	新しい属性を追加する。	2.0
Element.setAttributeNodeNS(newAttr);			
パラメータ	newAttr	属性リストに追加するAttrノード。	
戻り値		newAttr属性が既存の属性を同じローカル名と名前空間URIで置き換える場合は、既存のAttrノード。それ以外の場合はnull。	
メソッド	**removeAttributeNode**	指定する属性をリムーブする。	
Element.removeAttributeNode(oldAttr);			
パラメータ	oldAttr	属性リストからリムーブするAttrノード。	
戻り値		リムーブされたAttrノード。	
メソッド	**getElementsByTagName**	与えられたタグ名の全ての子孫要素のノードリストを返す。	
Element.getElementsByTagName(name);			
パラメータ	name	合致するタグ名。*は全てのタグに合致する。	
戻り値		合致する要素ノードのリスト。	
メソッド	**getElementsByTagNameNS**	ローカル名と名前空間URIで与えられたタグ名の全ての子孫要素のノードリストを返す。	2.0
Element.getElementsByTagNameNS(namespaceURI,DOMString localName);			
パラメータ	namespaceURI	合致する要素の名前空間URI。*は全てに合致する。	
	localName	合致する要素のローカル名。	
戻り値		合致する要素すべてを含むノードのリスト。	
メソッド	**hasAttribute**	指定される名前を持つか，あるいはデフォルト値を持つ属性があるかどうかを調べる。	2.0
Element.hasAttributeNS(namespaceURI,localName);			
パラメータ	name	要素もしくはデフォルト値で指定される名前を持つ属性。	
戻り値		ブール値。要素で指定される名前を持つか，あるいはデフォルト値を持つ属性があればtrue，なければfalse。	
メソッド	**hasAttributeNS**	ローカル名と名前空間URIで指定される属性があるかどうかを調べる。	2.0
Element.hasAttributeNS(namespaceURI,localName);			
パラメータ	namespaceURI	検索される属性の名前空間URI。	
	localName	検索される属性のローカル名。	
戻り値		ブール値。ローカル名と名前空間URIで指定される属性が特定できるか，要素においてデフォルト値を持つ場合はtrue，それ以外はfalse。	
メソッド	**normalize**	全てのテキストノードを、隣接するテキストノードがないように、単一のノードに正規化する。	1.0

Text		
要素および属性のテキスト内容を表す。CharacterDataからの継承。		
メソッド	**splitText**	オフセット位置を指定して2つのテキストノードを兄弟ノードに区切る。
Text.splitText(offset);		
パラメータ	offset	0から始まるオフセット位置。
戻り値		新規テキストノード。
Comment		
コメントの内容、<!--と-->との間の全てのキャラクタを表す。CharacterDataからの継承。		

■**拡張インタフェース**（Extended Interfaces ）

CDATASection		
CDATAセクションを表す。（CDATAセクションは、マークアップと見なされるキャラクタを含むテキストブロックをエスケープするために使われる。CDATAセクションのデリミタは、]]>である。CDATAセクションは入れ子にできない。Textからの継承。		
DocumentType		
DTDを表す。Nodeからの継承。		
属性	**name**	DOCTYPEに続く，DTDの名前。
	entities	内部実体、外部実体を含むNamedNodeMap。
	notations	DTD内の記法宣言を含むNamedNodeMap。
	internalSubset	文字列としての内部サブセット。
	publicId	外部サブセットの公開識別子。
	systemId	外部サブセットのシステム識別子。
Notation		
記法宣言を表す。Nodeからの継承。		
属性	**publicId**	公開識別子。指定されていない場合はnull。
	systemId	システム識別子。指定されていない場合はnull。
Entity		
パース対象実体、あるいはパース対象外実体を表す。Nodeからの継承。		
属性	**publicId**	公開識別子。指定されていない場合はnull。
	systemId	システム識別子。指定されていない場合はnull。
	notationName	パース対象外実体の記法の名前。パース対象実体についてはnull。
EntityReference		
実体参照を表す。Nodeからの継承。		
ProcessingInstruction		
処理命令を表す。Nodeからの継承。		
属性	**target**	処理命令のターゲット。
	data	処理命令の内容。

2.0

※注1
removeは構造モデルを変える時に使い、deleteは構造モデルの中にあるものを取り除く時に使う。deleteメソッドを使った場合は一度除去したものは返されないが、removeメソッドを使った場合は、返されることがある。

※※注2
DOMでは0から始まるインデックスを指定するが，XPathでは1から始まるポジションを指定する。プログラミング時に混同しないように注意が必要である。

3.4.4 実際の記述例

XML ツリーの生成処理

本節では，プログラムコードからDOMを使う例として，使用頻度の高い機能である，XMLツリーの構築と，検索処理によく用いられる要素ノードリストの取得を取り上げてみる。

●リスト 3-20 は，●リスト 3-19 のような簡易な XML ツリーを構築するコードの例である。

ここでは，要素を生成するために document インタフェースの createElement メソッド，テキストノードを生成するために createTextNode を使っている。次に，生成した要素をルート要素に追加したり，生成したテキストノードを要素の内容として追加するために Node インタフェース，appendChild メソッドを用いている。さらに，Element インタフェース setAttribute メソッドで属性をセットしている。

●リスト 3-19 生成する XML ツリー

```
<?xml version="1.0"?>
<オンライン記事>
  <記事名 公開日="2008-07-07">デザインの 30 年</記事名>
</オンライン記事>
```

●リスト 3-20 XML ツリーを生成する VB プログラムの一部（Visual Studio 2008 使用）

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
  Dim xmldoc As XmlDocument = New XmlDocument()
'XML 宣言とルート要素のみの XML 文書ファイルの読み込み
  xmldoc.LoadXml("<?xml version='1.0'?><オンライン記事 />")
'<記事名>要素の生成
  Dim articleTitleElement As XmlElement = xmldoc.CreateElement("記事名")
'属性名と属性値を指定
  Dim articleUpdateAttr As String = "公開日"
  Dim articleUpdateAttrValue As String = "2008-07-07"
'属性を生成して要素に追加
  articleTitleElement.SetAttribute(articleUpdateAttr, articleUpdateAttrValue)
'<記事名>要素の内容テキストの生成と追加
  articleTitleElement.AppendChild(xmldoc.CreateTextNode("デザインの 30 年"))
  xmldoc.DocumentElement.AppendChild(articleTitleElement)
  TextBox1.Text = xmldoc.OuterXml
End Sub
```

要素ノードリストの取得

次に，ノードを選択して内容を抽出する例を示す。最も使いやすく分かりやすいのは，Element インタフェースの getElementsByTagName メソッドである。指定するタグ名を持つ要素ノードのリストを取得することができる。

ここでは，既出の●リスト 3-1 のような XML 文書の中から，<タイトル>の内容だけを抽出する処理を実装してみる。まず<タイトル>という要素ノードのリストを取得する。ノードリストが取得できたら，その中に含まれる要素の内容を抽出する（次ページ，●リスト 3-21）。但し，これが目的の処理を実装するための唯一の方法というわけではない。

この getElementsByTagName メソッドは，機能がその名の通りであって分かりやすいメソッドであるので，こういった解説用のサンプルプログラムではよく用いられるが，説明のしやすさや使いやすさと実務での使用頻度は必ずしも一致しない。

また，DOM はプラットフォームや言語に非依存のインタフェースではあるが，実際は開発言語および動作環境と切り離すことができない。実際，開発現場でデータ処理を実装するとなると，テキストエディタでスクリプトを記述よりも，何らかの開発・実行環境を利用することが少なくない。

ベンダ側が W3C DOM の API を拡張したクラス，メソッド，プロパティを提供していることもある。たとえば，.NET Framework を開発･実行環境して DOM を使うのであれば，XPath 式と一致するノードリストを選択する SelectNodes メソッドや，XPath 式と一致する最初のノードを選択する SelectSingleNode メソッドが提供されている。選択するノードのデータが，<ID>や<プロダクト番号>のような一意のものであれば，選択対象は 1 個だけであるので，指定したタグを一通り走査していく getElementsByTagName メソッドよりも，一個見つかれば結果を返す SelectSingleNode メソッドを用いる方が，効率が良い。さらに，.NET でいえば，カーソルモデルと XPath を使った XPathNavigator クラスといった，走査効率の良い機能も提供されている。

●リスト 3-21 <タイトル>要素の内容テキストを抽出する VB プログラムの一部（Visual Studio 2008 使用）

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
  Dim filePath As String = Server.MapPath("./")
  Dim xmldoc As XmlDocument = New XmlDocument()
  xmldoc.Load(filePath & "App_Data/SampleData_Article.xml")
'<タイトル>要素ノードリストを取得している
  Dim titleNodes As XmlNodeList = xmldoc.GetElementsByTagName("タイトル")
  For i As Integer = 0 To titleNodes.Count - 1
    TextBox1.Text &= titleNodes(i).InnerXml & vbCrLf
  Next i
End Sub
```


さらに，開発･実行環境が.NET Framework である場合，もはや DOM は，ほとんど使わないといっても過言ではない。.NET 3.5 以降であれば，LINQ to XML が用いられる。皆さんが，今後，どのような開発･実行環境下で仕事をすることになるのかは分からないが，分からないからこそ，「特定のタグ名を持つデータの取得には，getElementsByTagName を使えばいい」などと頭から思い込むのではなく，開発･実行環境や開発目的によって，採用する仕様やメソッドを柔軟に選択した方がよいと思われる。

実務では，仕様書の規定と，実装方法の間に，しばしば溝がある。仕様に従うか動作を重視するかという問題については，どちらの方が最終的に XML 化の目的を達成できるかによる。

W3C 仕様はすべての基本であり，比較的普遍であるから，尊重すべきではある。そのうえで，どのような状況下であえて独自仕様を用いるかという判断も必要となる。XML アプリケーションの安定稼働が，W3C 仕様あるいは開発環境･動作環境のバージョンアップにできるだけ左右されることのないよう，慎重且つ的確な判断がもとめられる。

実務者には，理論と現実のどちらか一方に偏らない，バランス感覚がもとめられる。

なお，W3C 仕様に限定した場合であっても，DOM を使って実装可能な機能のすべてが，必ずしも DOM を使わなければ実装できない，というわけではない。むしろ異なるツリーの生成には，後述の XSLT を使った方がよい場合もある。

3.5 XSLT による変換

3.5.1 XSLT の概要

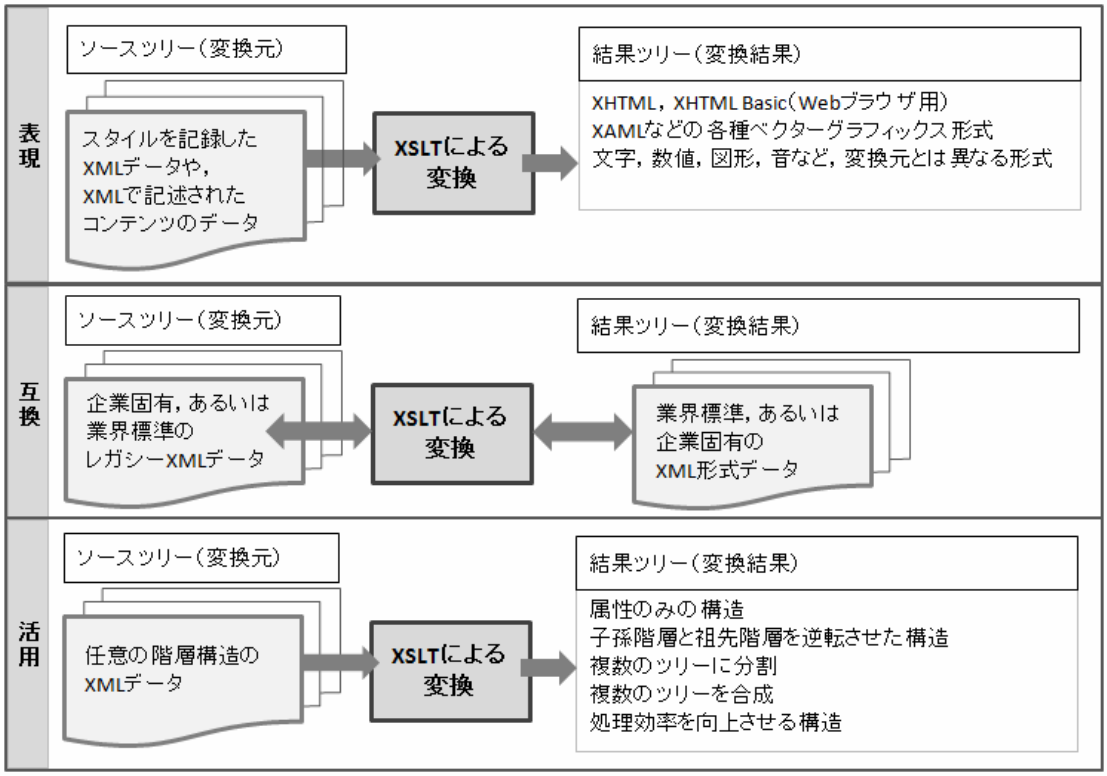
XSLT の機能

1 個の XML 文書を複数の目的に使用したり，Web ページとして表示するためのデータとして活用したい場合，木の構造変更を余儀なくされることがある。そのような場合に役立つのが，XSLT (Extensible Stylesheets Language Transformations) である。XSLT は，XML 文書を異なる構造の XML 文書に変換するための言語である。XSLT を使えば，XML 文書中の中の要素･属性･データを追加･変更したり，フィルタリングやソートを実行した結果の木を，生成することができる。

XSLT では，変換前の元の XML 文書のツリーを「ソースツリー (source tree)」，変換結果ととして生成される XML 文書の木を「結果ツリー (result tree)」と呼ぶ。XSLT とは，ソースツリーを結果ツリーに変換するためのルールを定めたものなのである。

実務で XSLT が使われる目的は，主に次の 3 つである (●図 3-9)。

●図 3-9 XSLT を用いて異なる構造の XML 文書に変換することができる。



目的の1つは「表現」である。各種 Web ブラウザやプラグインで表示可能な形式に変換することができる。もちろん，変換方法によっては，アクセシビリティに配慮することもできる (第 5 章「5.5 ユーザインタフェースの設定と適用」参照)。それが整形式の XML 文書ファイルであれば，文字，数値，図形，音といった一見無関係に見えるデータ同士であっても変換でき，新しい表現手法を編み出すことも可能である。なお，現在 Web サイトの記述言語として用いられている XHTML は，HTML を XML の仕様に則って再定義した言語であるので，XSLT 変換により XHTML を出力することも可能である。

2 つ目は「互換」である。XML をデータ交換フォーマットとして用い，企業固有のレガシーXML データと，業界標準の XML データを相互に変換することができる。他の XML 関連仕様に変換することも可能である。

3 つ目は，「活用」である。元の XML 文書が蓄積には適しているが処理には難点がある場合，構造を変換してから処理することがある。第 1 章で触れたように，データは同じであっても，いろいろな枝ぶりの木構造が考えられる。変換した XML ツリーをメモリ上に展開する場合には DOM を用いるが，XSLT で変換して一度 XML 文書ファイルとして保存する方法もある。どちらの方法を採用するかは，異なる木構造文書の，処理目的や運用期間の長短から判断するとよいだろう。

仕様の現状

XSLT 1.0 は 1999 年 11 月 6 日に W3C 勧告となった（http://www.w3.org/TR/xslt）。その後，XSLT 2.0 が，2007 年 1 月 23 日に勧告となっている（http://www.w3.org/TR/2007/REC-xslt20-20070123/）。XSLT2.0 は，XPath 2.0 の 3 仕様（XPath2.0, Data Model, Functions and Operators）との連携で用いられる。これらの仕様書は，「XML Technology＞Transformation」コーナから入手することができる。

本書では，XSLT1.0 をサポートするプロセッサが広く使われていることから，XSLT1.0 について解説を進める。XSLT1.0 で変換対象を指定するには XPath1.0 を用いるが，XPath2.0 であってもパス式の基本的な構文は変わらない。

これ以降は，便宜上，XSLT の変換規則を記述したコードを「XSLT スタイルシート」，XSLT コードを拡張子.xsl で保存したものを「XSLT ファイル」と表記している。Web ページの表現を定義する CSS もスタイルシートと呼ばれるが，XSLT による変換を表す「スタイルシート（stylesheet）」とは，全く異なるものである。XSLT は変換のための仕様であり，CSS は表現のための仕様である。XSLT コードの中に CSS を記述すると，変換結果にビジュアル表現を適用することが可能となる。

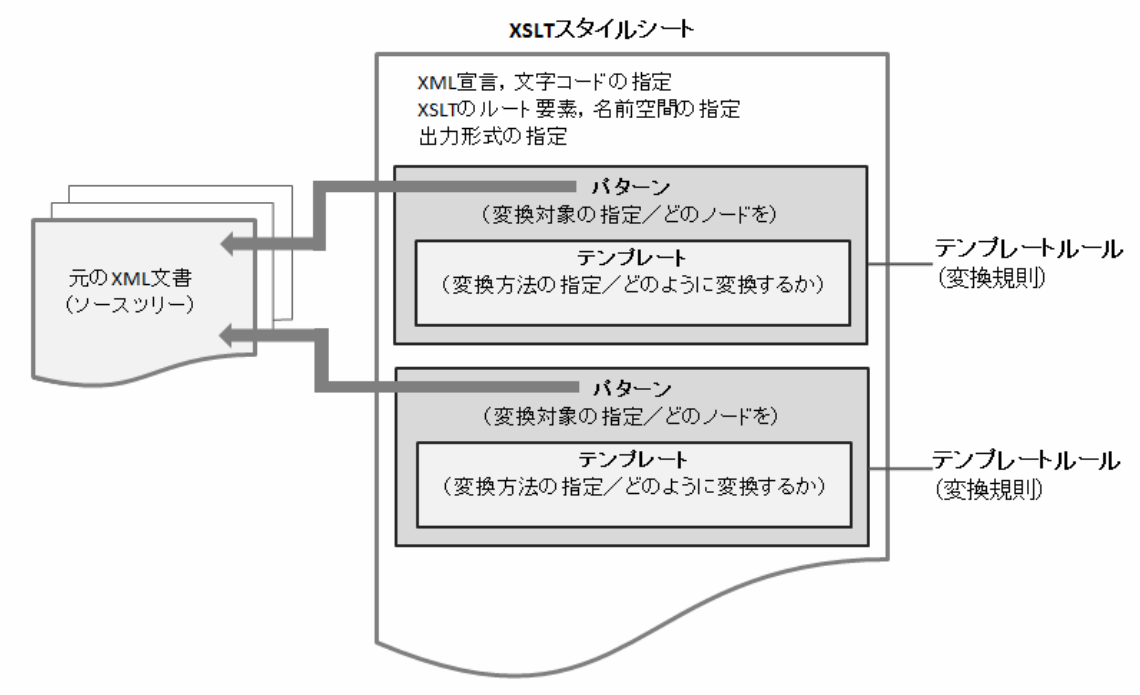
3.5.2 基本構文

テンプレートルール

XSLT スタイルシートとは，ソースツリーから結果ツリーに変換するルール（rules）を記述したものである。この変換規則をテンプレートルール（template rule）という。XSLT スタイルシートは，テンプレートルールの集合である。

テンプレートルールは，テンプレート（template）とパターン（pattern）の組み合わせである。指定したパターンがソースツリー中の変換対象に合致（match）すると，1 個のテンプレートはインスタンス化されて，結果ツリーの一部を生成する（●図 3-10）。

●図 3-10 XSLT スタイルシートの構成



XSLT のルート要素と名前空間

XSLT の名前空間 URI は，http://www.w3.org/1999/XSL/Transform，ルート要素は，<xsl:stylesheet>である（同義語として<xsl:transform>がある）。XSLT 要素の接頭辞は，通常は，xsl:を使う。構文は●リスト 3-22 の通りである。

●リスト 3-22 XSLT のルート要素

```
<xsl:stylesheet
  id = 埋め込みスタイルシートを識別する id
  extension-element-prefixes = 拡張名前空間の接頭辞のリスト
  exclude-result-prefixes = 結果ツリーから除外する名前空間の接頭辞のリスト
  version = バージョン>
  <!--内容(xsl:import*，トップレベル要素) -->
</xsl:stylesheet>
```

XSLT の要素一覧

トップレベル要素（top-level element）としては，次のものを指定できる。

xsl:import，xsl:include，xsl:strip-space，xsl:preserve-space，xsl:output，xsl:key，xsl:decimal-format，xsl:namespace-alias，xsl:attribute-set，xsl:variable，xsl:param，xsl:template

これらトップレベル要素を含む XSLT 1.0 の各要素の機能は●表 3-9 の通りである。各要素の構文および使い方については，本節ですべてを語ることはできないので，本節では，XSLT の目的と基本的な使用方法について解説していく。

●表 3-9 XSLT1.0 の要素

スタイルシート要素		
xsl:stylesheet	他の全ての要素を含む，XSLTスタイルシートのルート要素。	<xsl:stylesheet id = id extension-element-prefixes = 接頭辞のリスト exclude-result-prefixes = 接頭辞のリスト version = バージョンを表す数字> <!-- テンプレートの内容: (xsl:import*, top-level-elements) --> </xsl:stylesheet>
xsl:transform	xsl:template要素とmsxsl:script要素を含むスタイルシートのルート要素。xsl:stylesheetと同義。	<xsl:transform id = id extension-element-prefixes = 接頭辞のリスト exclude-result-prefixes = 接頭辞のリスト version = バージョンを表す数字> <!-- テンプレートの内容: (xsl:import*, top-level-elements) --> </xsl:transform>
スタイルシートの結合		
xsl:include	別のXSLTスタイルシートをインクルードする。	<!-- Category: top-level-element --> <xsl:include href = URI参照 />
xsl:import	別のXSLTスタイルシートをインポートする。	<xsl:import href = URI参照 />
空白の扱い		
xsl:strip-space	空白を除去する。	<!-- Category: top-level-element --> <xsl:strip-space elements = 空白区切りのNameTestsのリスト />
xsl:preserve-space	空白を保持する。	<!-- Category: top-level-element --> <xsl:preserve-space elements = 空白区切りのNameTestsのリスト />
テンプレートルール		
xsl:template	テンプレートルールを定義する。	<!-- Category: top-level-element --> <xsl:template match = パターン name = テンプレートの名前 priority = 優先順位 mode = モード名> <!-- テンプレートの内容: (xsl:param*, template) --> </xsl:template>
xsl:apply-templates	テンプレートルールの適用。式で選択したノードを処理する。式が指定されていない場合は，テキストノードを含むカレントノードのすべての子を処理する。	<!-- Category: instruction --> <xsl:apply-templates select = ノード集合の式 mode = モード名> <!-- テンプレートの内容: (xsl:sort | xsl:with-param)* --> </xsl:apply-templates>
xsl:apply-imports	オーバーライドするテンプレートルールを呼び出す。	<!-- Category: instruction --> <xsl:apply-imports />
名前付けテンプレート		
xsl:call-template	名前でテンプレートを呼び出す。	<!-- Category: instruction --> <xsl:call-template name = テンプレートに付ける名前> <!-- 内容: xsl:with-param* --> </xsl:call-template>
結果ツリーの生成		
xsl:namespace-alias	別のネームスペースの接頭辞に置き換える。	<!-- Category: top-level-element --> <xsl:namespace-alias stylesheet-prefix = 変更元の接頭辞 | "#default" result-prefix = 変更後の接頭辞 | "#default" />
xsl:element	name属性に指定した名前の要素を生成する。	<!-- Category: instruction --> <xsl:element name = { 修飾名 } namespace = { URI参照 } use-attribute-sets = 属性セットの名前> <!-- テンプレートの内容: template --> </xsl:element>
xsl:attribute	name属性に指定した名前の属性を生成する。	<!-- Category: instruction --> <xsl:attribute name = { 属性名 } namespace = { URI参照 }> <!-- テンプレートの内容 --> </xsl:attribute>
xsl:attribute-set	名前を持つ属性の集合を定義する。	<!-- Category: top-level-element --> <xsl:attribute-set name = use-attribute-setsで使う属性集合の名前 use-attribute-sets = 空白で区切られた属性集合名のリスト> <!-- 内容: xsl:attribute* --> </xsl:attribute-set>
xsl:text	出力するテキストを生成する。	<!-- Category: instruction --> <xsl:text disable-output-escaping = "yes" | "no(デフォルト)"> <!-- テキスト化したいデータ: #PCDATA --> </xsl:text>
xsl:processing-instruction	出力する処理命令を生成する。	<!-- Category: instruction --> <xsl:processing-instruction name = { PIの名前 }> <!-- テンプレートの内容 --> </xsl:processing-instruction>
xsl:comment	出力するコメントを生成する。	<!-- Category: instruction --> <xsl:comment> <!-- コメントノードの文字列を示すテンプレート --> </xsl:comment>

xsl:copy	出力されるソースのカレントノードをコピーする。	<!-- Category: instruction --> <xsl:copy use-attribute-sets = 属性セット名> <!-- テンプレートの内容 --> </xsl:copy>
xsl:value-of	テキストを動的に生成する。	<!-- Category: instruction --> <xsl:value-of select = 文字列式 disable-output-escaping = "yes" | "no" />
ナンバリング		
xsl:number	結果ツリーに番号を挿入する。	<!-- Category: instruction --> <xsl:number level = "single" | "multiple" | "any" count = パターン from = パターン value = 数値式 format = { カウンタの表示形式 } lang = { 言語 } letter-value = { "alphabetic" | "traditional" } grouping-separator = { 桁区切り記号 } grouping-size = { 桁区切り桁数 } />
繰り返し		
xsl:for-each	select属性で指定した式で選択されるノードを繰り返し処理する。	<!-- Category: instruction --> <xsl:for-each select = ノード集合の式> <!-- テンプレートの内容: (xsl:sort*, template) --> </xsl:for-each>
条件分岐		
xsl:if	式を評価した結果が真ならテンプレートを処理する。	<!-- Category: instruction --> <xsl:if test = ブール式> <!-- テンプレートの内容 --> </xsl:if>
xsl:choose, xsl:when, xsl:otherwise	複数の条件を指定してテンプレートを処理する。 (xsl:when+, xsl:otherwise?)	<!-- Category: instruction --> <xsl:choose> <xsl:when test = ブール式> <!-- テンプレートの内容 --> </xsl:when> <xsl:otherwise> <!-- テンプレートの内容 --> </xsl:otherwise> </xsl:choose>
ソート		
xsl:sort	xsl:for-eachやxsl:apply-templatesで指定されたノードリストをソートする。	<xsl:sort select = 文字列式 lang = { 言語 } data-type = { "text" | "number" | qname-but-not-ncname } order = { "ascending" | "descending" } case-order = { "upper-first" | "lower-first" } />
変数とパラメータ		
xsl:variable	変数を宣言し，バインドする値を指定する。	<!-- Category: top-level-element --> <!-- Category: instruction --> <xsl:variable name = 変数名 select = 式> <!-- select属性を指定しない場合，処理されると変数値になるテンプレートの内容 --> </xsl:variable>
xsl:param	xsl:stylesheetまたはxsl:templateの中で使われる名前の付いたパラメータを宣言する。	<!-- Category: top-level-element --> <xsl:param name = パラメータ名 select = expression> <!-- select属性を指定しない場合，処理されるとパラメータ値になるテンプレートの内容 --> </xsl:param>
xsl:copy-of	結果ツリーにサブツリーと結果ツリーのフラグを挿入する。	<!-- Category: instruction --> <xsl:copy-of select = 式 />
xsl:with-param	テンプレートにパラメータを引き渡す。	<xsl:with-param name = パラメータ名 select = 式> <!-- テンプレートの内容 --> </xsl:with-param>
キー		
xsl:key	key関数で使う名前の付いたキーを宣言する。	<!-- Category: top-level-element --> <xsl:key name = キーの名前 match = パターン use = 式 />
数値の扱い		
xsl:decimal-format	format-numberで使われるフォーマットパターンの解釈の制御を指定する。	<!-- Category: top-level-element --> <xsl:decimal-format name = 名前 decimal-separator = 小数点記号，デフォルトは.(ピリオド) grouping-separator = グループ化の区切り記号，デフォルトは,(カンマ) infinity = 無限大を表す文字列，デフォルトはinfinity minus-sign = マイナス記号を表す文字，デフォルトはハイフンのマイナス文字 NaN = NaN型の値を表す文字列，デフォルトはNaN percent = パーセント記号を表す文字，デフォルトはパーセント文字(%) per-mille = 1000分の1記号，デフォルトはUnicodeのパーミル文字(#x2030) zero-digit = ゼロに使う文字，デフォルトはdigit文字のzero(0) digit = 数字の表記に使われる文字，デフォルトはナンバー記号文字の(#) pattern-separator = プラスとマイナスの区切り記号，デフォルトはセミコロン(;) />

メッセージ		
xsl:message	ダイアログボックスにテキストのメッセージを送信する。	<!-- Category: instruction --> <xsl:message terminate = "yes" | "no"> <!-- メッセージの内容 --> </xsl:message>
フォールバック		
xsl:fallback	解釈不可能な新しい要素に対して，代替となるテンプレートの内容を呼び出す。	<!-- Category: instruction --> <xsl:fallback> <!-- 代替になる処理可能なテンプレートの内容 --> </xsl:fallback>
出力		
xsl:output	結果ツリー構築のためのオプションを指定する。	<!-- Category: top-level-element --> <xsl:output method = "xml" | "html" | "text" | qname-but-not-ncname version = バージョン encoding = 文字コード omit-xml-declaration = "yes" | "no" standalone = "yes" | "no" doctype-public = 公開識別子 doctype-system = システム識別子 cdata-section-elements = CDATAセクション出力に使う要素名のリスト indent = "yes" | "no" media-type = MIMEコンテンツタイプ />

出力形式の指定

XSLT により構築した結果ツリーは，XML，HTML，テキストのいずれかの形式で出力することができる。変換結果は，ブラウザ画面やフォーム上に出力するだけでなく，プログラムにより異なる XML 文書ファイルとして保存することもある。

変換結果を XML1.0 形式で出力する XSLT スタイルシートの基本構文は，●リスト 3-23 のとおりである。version 属性値には出力方法のバージョンを， indent 属性値には，XSLT プロセッサによる結果ツリーの出力時に空白を追加するかどうかを指定する。

●リスト 3-23　変換結果を XML1.0 形式で出力する基本のコード

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" indent="yes" />
    ～内容としてテンプレートルールを記述する～
</xsl:stylesheet>
```

3.5.3 結果ツリーの生成

テンプレートルールの定義

次に，実際の記述例を示す。●リスト 3-24 のような XML 文書（ソースツリー）を，●リスト 3-25 のような XML 文書（結果ツリー）に変換するには，●リスト 3-26 のような XSLT スタイルシートを記述する。

●リスト 3-24　サンプル XML 文書（ソースツリー）

```
<?xml version="1.0"?>
<dataroot>
  <record1>
    <theme>デザインの 30 年</theme>
    <title>手計算と手作業</title>
```

```
    <period>1970 年代後半</period>
  </record1>
  <record1>
    <theme>デザインの 30 年</theme>
    <title>制作工程の分業</title>
    <period>1980 年代前半</period>
  </record1>
  <record1>
    <theme>デザインの 30 年</theme>
    <title>ワープロの普及</title>
    <period>1980 年代後半</period>
  </record1>
  <record1>〜</record1>繰り返し略
</dataroot>
```

●リスト 3-25　期待する結果ツリー

```
<?xml version="1.0"?>
<公開記事>
  <オンライン記事>
    <記事名>デザインの 30 年</記事名>
    <目次>
      <タイトル　年代="1970 年代後半">手計算と手作業</タイトル>
      <タイトル　年代="1980 年代前半">制作工程の分業</タイトル>
      <タイトル　年代="1980 年代後半">ワープロの普及</タイトル>
      <タイトル　年代="1990 年代前半">DTP の普及</タイトル>
      <タイトル　年代="1990 年代後半">Web サイトの連携</タイトル>
      <タイトル　年代="2000 年代前半">情報デザインの始まり</タイトル>
    </目次>
  </オンライン記事>
</公開記事>
```

テンプレートルールの定義には，<xsl:template>要素を使う。match 属性には，ソースツリーの中のノードや，テンプレートルールを適用するノードを識別するためのパターンを，XPath1.0 のパス式（XPath expressions）を用いて指定する。

要素の生成には<xsl:element>要素を，属性の生成には<xsl:attribute>要素を用いる。それぞれ"name"属性に，生成する要素名や属性名を指定する。ここでは使っていないが，オプションとして namespace 属性があり，URI 参照を指定することができる。

ソースツリー中の要素の内容などからテキストを動的に生成する（computing generated text）には，<xsl:value-of>要素を用いて，"select"属性値にテキストノード化したい文字列式を指定する。この要素

はインスタンス化されると，結果ツリーの中にテキストノードを生成する。リストでは，<xsl:value-of select="dataroot/record1/theme" />のように記述して，XML データから<記事名>要素の内容となるテキストを動的に生成している。これを，XSLT スタイルシートの中で固定して，<xsl:text>デザインの 30 年</xsl:text>のように記述しても，結果は同じである。XML データの変更を受けない部分，たとえば，XML データを変換して HTML ページを生成して出力する場合，各ページに共通する Web サイト名やコーナ名などは，<xsl:text>要素を使ってテキストノードを生成（create a text node）するのも一方法である。

繰り返し処理には，<xsl:for-each>要素を用いる。プログラムコードにおける for～next と同じ役割を果たすが，増分を指定する属性はなく，単純に select 属性値に，繰り返すノード集合を式として指定するだけである。この<xsl:for-each>要素は入れ子にすることができる。

●リスト 3-26 ●リスト 3-24 のソースツリーを●リスト 3-25 の結果ツリーに変換する XSLT スタイルシート

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" indent="yes" />
  <xsl:template match="/">
    <xsl:element name="公開記事">
    <xsl:element name="オンライン記事">
    <xsl:element name="記事名">
      <xsl:value-of select="dataroot/record1/theme" />
    </xsl:element>
    <xsl:element name="目次">
    <xsl:for-each select="dataroot/record1">
      <xsl:element name="タイトル">
        <xsl:attribute name="年代">
          <xsl:value-of select="period" />
        </xsl:attribute>
        <xsl:value-of select="title" />
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
   </xsl:element>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

変換結果を XML 形式で出力する。

結果ツリーの要素名を指定して，要素を生成する。

テキストを生成する。

処理を繰り返す。

属性を生成する。

テンプレートルールの適用

●リスト 3-26 は，<xsl:apply-templates>要素を使って，●リスト 3-27 のように記述することもできる。この要素は，定義済みのテンプレートを適用する（applying template rules）ために用いる。

●**リスト 3-27** リスト 3-26 を<xsl:apply-templates>要素で代替えした例

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" indent="yes" />
  <xsl:template match="/">
    <xsl:element name="公開記事">
      <xsl:apply-templates select="dataroot" />
    </xsl:element>
  </xsl:template>

  <xsl:template match="dataroot">
    <xsl:element name="オンライン記事">
      <xsl:element name="記事名">
        <xsl:value-of select="record1/theme" />
      </xsl:element>
      <xsl:element name="目次">
        <xsl:for-each select="record1">
          <xsl:element name="タイトル">
            <xsl:attribute name="年代">
             <xsl:value-of select="period" />
            </xsl:attribute>
            <xsl:value-of select="title" />
          </xsl:element>
        </xsl:for-each>
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

テンプレートルールを適用してい

3.5.4 使用頻度の高い要素

条件分岐

XSLT にはプログラムライクな要素が用意されている。特に使用頻度が高いのは，条件分岐の機能を持つ要素である。二者択一の条件分岐には，<xsl:if>要素を使い，"test"属性に条件式を指定する。式の評価結果はブール値に変換される。プログラムでいえば，If 文と同じ使い方である。XSLT では，XHTML への変換が可能であるから，表示と非表示データを切り分けたり，適用する XHTML タグを切り分けて表現を変えるといった処理が可能になる。

複数の条件分岐には，<xsl:choose>要素を，<xsl:when>および<xsl:otherwise>要素と組み合わせて使う。プログラムでいえば，Select Case 文と使い方は同じである。

ソート

ID や型番や年齢など，特定の要素をキーとしてソート（sorting）するには，<xsl:apply-templates>要素あるいは<xsl:for-each>要素の子として<xsl:sort>要素を用い，"order"属性値に ascending（昇順）あるいは descending（降順）を指定する。

空白と出力エスケープ

XSLT には空白を制御する要素（whitespace stripping）がある。空白を削除するには xsl:strip-space 要素を，保持するには，xsl:preserve-space 要素を使う。

また，テキストノードを XML 形式で出力すると，&あるいは<は，出力結果が整形式 XML 文書となるようにエスケープされるが，これを無効にするには（output without any escaping），xsl:text 要素および xsl:value-of 要素において，<xsl:value-of select="文字列式" disable-output-escaping="yes"/>のように，"disable-output-escaping"属性値に yes を指定する。

変数バインディング

XSLT には，変数バインディング要素（variable-binding element）がある。<xsl:variable>要素で変数を宣言して，値を代入することができ，<xsl:param>要素ではパラメータを指定することができる。この 2 つの要素は，宣言する場所によってスコープ（scope）が異なる。トップレベル要素として使うと（top-level variable-binding element），グローバル変数（a global variable）扱いとなり，テンプレート内で用いるとローカル変数扱いとなる。変数やパラメータに代入されている値を利用するには，変数名やパラメータ名の頭に$記号を付ける（それぞれの要素の使用方法については「第 6 章参照）。

別のテンプレートにパラメータの値を引き渡すには，<xsl:param>要素に<xsl:with-param>要素を組み合わせて使う。

3.5.5 XML 文書への XSLT スタイルシートの適用

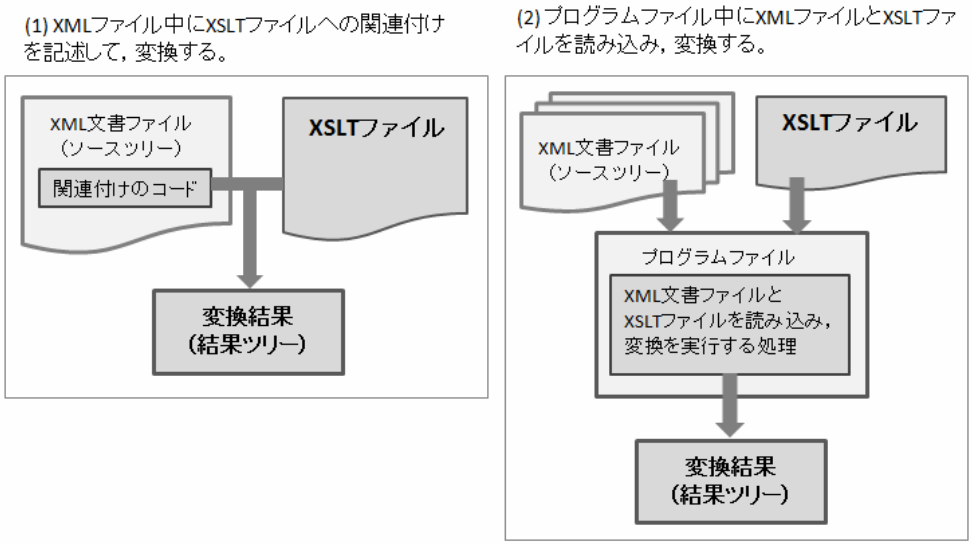
XML 文書との関連付け

XSLT スタイルシートを単に記述して XSLT ファイルとして保存しただけでは，XML 文書を変換することはできない。両者を何らかの方法で関連付け，変換を実行する必要がある。ただし，その方法は，XSLT1.0 仕様書で定められているわけではない。実際の使い方としては，●図 3-11 のように 2 種類ある。

1 つ目の方法は，XML 文書中に直接 XSLT ファイルへの関連付けのコードを記述するというものである（●リスト 3-28）。

もう 1 つの方法は，プログラムで XML 文書ファイルと XSLT ファイルを読み込んで，変換する方法である。これには，設計如何で，さまざまな構成が考えられる（第 5 章参照）。プログラムで XSLT による変換を実行する場合，XML 文書ファイルと XSLT ファイルの読み込み，および変換の実装方法は，利用する開発･実行環境によって異なる。

●図 3-11　XSLT スタイルシートによる変換を実行する方法



●リスト 3-28　XML 文書中に記述する XSLT ファイルへの 1 対 1 の関連付け

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="関連付ける XSLT ファイル名"?>
<ルート要素>～略～</ルート要素>
```

複数文書へのアクセス

XSLT には，XPath1.0 の関数ライブラリ（core XPath function library）に追加された関数（additional functions）がある（●表 3-10）。このうち document 関数を使って，メインのソース文書以外の XML 文書にアクセスすることができる。たとえば，「A.xml」に XSLT ファイル「A.xsl 」への関連付けのコードを記述する。その「A.xsl」の中に，●リスト 3-29 のように記述すると，「B.xml」にアクセスすることができる。

●表 3-10　XSLT の追加の関数

基本構文	機能
document(オブジェクト，ノー集合?)	メインのXML文書以外のXML文書へのアクセスを許可するノード集合関数
key(キーの名前，式の条件を表すオブジェクト)	xsl:key要素と組み合わせて，指定する動作を呼び出す，ノード集合関数
format-number(数値，フォーマットパターン文字列，文字列?)	2番目の引数のフォーマットパターン文字列と，3番目の引数のdecimal-format(あるいはデフォルト)で，1番目の引数を文字列に変換する文字列関数

●リスト 3-29　document 関数を使った複数文書へのアクセス

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" version="1.0" indent="yes" />
<xsl:variable name="変数名" select="document('B.xml')/ルート要素名" />
  <xsl:template match="/">
    ～略～
  </xsl:template>
</xsl:stylesheet>
```

DOM か XSLT か

以上のような既存 XML 文書を利用したツリーの再構築は，前掲の DOM によるツリーの生成でも代替え可能である。そのため，XML の構造やノード数，処理目的によって，任意の開発言語によるプログラムコードの中で DOM を使うか，XSLT を単独で使うかを判断する必要がある。

生成するツリーの構造が定型で深い階層構造を持つものであれば，むしろ XSLT の方が実装は容易である。また，XHTML として出力した結果に CSS を適用する場合は，DOM で抽出したデータを HTML タグに関連付けたり，フレームワークの提供するコントロールにバインドするよりも，XSLT のみで済ませる方が，メンテナンスしやすくリデザインに対応しやすいことも多い。XSLT は，Web アプリケーションにおいて静的な最終結果を出力する場合に用いると効果的である（第 5 章参照）。

さらに，実装した処理の運用期間についても検討する必要がある。DOM そのものは安定した仕様であっても，開発言語によるプログラムの中で使う以上，それら開発言語のバージョンアップに際して，メンテナンスが必要になるのに対し，XSLT は単独で機能するので，バージョンアップの影響を受けにくい。長期運用が前提条件であれば，XSLT を採用した方が，運用期間を通しての開発・運用環境のバージョンアップの影響を受けずに済む。開発実務においては，将来性や拡張性と，顧客側の運用コストのバランスを考えて，採用する仕様を決定するとよい。実務では，最適の使用技術を見極める「判断力」が重要である。

XSLT を採用した場合，実際にどの程度複雑な処理が可能であろうか。マシンのスペックや採用する XSLT プロセッサにも左右されるが，筆者の経験では，ローカルアプリケーションとして実装するのであれば，2 万ノードのソースツリーから，10 回のネストされた繰り返し処理の中で，30 回の条件分岐によって異なる結果ツリーを生成し分ける処理程度は可能であるといってよい。

ただし，それ以上に複雑な変換処理を実装する必要があったり，1 個の XML 文書ファイル中の全ノード数が 10 万以上であったり，処理対象とする XML 文書ファイルのファイルサイズが 100MB を超えるとなれば，筆者ならば DOM でも XSLT でもなく LINQ を採用する。

XSLT は，他の普及しているプログラム言語に比べて，使用頻度は高いとは言えない。

だが，Web アプリケーションにおける HTML 出力に限っては強力な仕様である。今やディスプレイ上の表現には，Silverlight や Flash などが多用されているものの，単純なテキストと静止画像の表示には XHTML が広く用いられている。また，互換の必要性は増えこそすれ減ることはない（互換処理を XSLT のみで実装するかどうかは別として）。XSLT は，今後 3 年間は必要な技術であり続けるので，習得しておくと何かと役には立つであろう。

3.6 LINQ to XML によるクエリ

3.6.1 LINQ to XML の概要

LINQ to XML の役割

前述の XQuery, DOM の項で述べたように, 開発･実行環境として.NET Framework を採用する場合は, LINQ to XML (Language-Integrated Query, 統合言語クエリ) を用いる方がよい。

LINQ は, .NET Framework3.5 より追加された, 言語埋め込み型クエリ技術である。2008 年 12 月に日本語版がリリースされた Visual Studio 2008 から利用可能となっている。LINQ クエリの基本パターンは, RDB(LINQ to SQL), XML(LINQ to XML), Object(LINQ to Object)を問わず同一であるため, データ形式別に異なるクエリ言語を習得する必要はない。

XML と RDB を併用する場合は, 本章冒頭の●図 3-2 に挙げたように, SQL Server データベースの XML 型の列に XML ツリーを格納するのではなく, 併用することができる。たとえば, SQL Server データベースの主キーと, ネイティブ XML 文書ファイル内の任意の要素が対応する構造とし, 両者のデータを結合して処理するといった方法である。SQL Server データに対しても, XML データに対しても, LINQ を用いればよい。

System.Xml.Linq 名前空間に含まれる LINQ to XML のクラスには, データ取得やフィルタリングに加え, 投影, ソート, グループ化, 結合などの強力な機能がある。これらにより, XML データの抽出･検索, XML ツリーの生成･変更･削除･更新, シリアライズといった, XML 案件で必要とされる処理の大半を実装することができる。従来 XQuery, DOM, XSLT による処理の多くは, LINQ ひとつで代替え可能であるので, .NET 系エンジニアの場合は, XPath の基本を理解した後, XQuery と DOM を習得するよりも先に, LINQ to XML を学ぶ方が効率的である。

ただし, XSLT については, LINQ と使い分けた方がよいケースもあるので, LINQ 習得後に学んでおくとよい。

●表 3-11 LINQ でサポートされている基本の句と機能

基本の句	機能
From	データを格納する変数名やデータの集合を指定
From句と組み合わせる句	**機能**
Select	投影(クエリの結果として選択される要素)
Where	フィルタリング(フィルタ条件)
Order By	ソート
Aggregate	クエリ結果を評価(関数併用など)
Join, Group Join	結合
Group By	グループ化
Let	式の評価結果を反復変数に格納
Distinct	重複するクエリを排除
Skip, Take	クエリ結果の一部を指定
Skip While, Take While	

LINQ クエリの基本構文

LINQ to XML の基本的なクエリは，処理対象 XML 文書を読み込んだ後，「データソースの指定→クエリの作成→クエリの実行」の順番で記述する。LINQ でサポートされている句の機能は，前掲●表 3-11 の通りである。各句の構文については，VS2008 ヘルプの「標準クエリ演算子のクエリ式構文」に詳しい。

XML 文書の読み込み

LINQ を用いて処理対象 XML 文書を読み込むには，次のように記述する（Web アプリケーションの場合）。

```
Dim returnValue　As XDocument= XDocument.Load (XML 文書ファイルの URI)
Dim returnValue　As XElement= XElement.Load(XML 文書ファイルの URI)
```

XDocument クラスと XElement クラスは，XML to LINQ 軸の起点となるルートノードの扱い方に応じて使い分ける。

XDocument クラスを用いて読み込んだ XML 文書のルート要素ノードは「ルートノードの子」として扱われる。つまり，ルートノードの子孫といえば，読み込んだ XML 文書のルート要素以下を指す。

XElement クラスを用いた場合は，読み込んだ XML 文書のルート要素ノードが「ルートノード」として扱われる。つまり，ルートノードの子孫といえば，ルート要素自身は含まれず，読み込んだ XML 文書のルート要素の子孫を指す。

実務においては，ルート要素そのものを指定した処理は稀であるため，基本的には XElement クラスを用いると考えてよいだろう。

基本のクエリ

LINQ to XML を用いた，Web アプリケーションにおける，クエリを取得するための基本構文の例は，●リスト 3-30 の通りである。

●リスト 3-30　LINQ を用いた，クエリ取得の基本コード（Web アプリケーションの場合）

```
Option Strict On
Imports System.Linq
Imports System.Xml.Linq

Partial Class _Default
    Inherits System.Web.UI.Page

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
Dim filePath As String = Server.MapPath("./")
Dim xmldoc As XElement = XElement.Load(filePath & "XML ファイル名")
```

```
Dim query As Enumerable(Of 列挙するオブジェクトの型) = From 範囲変数(反復変数) In 対象コレクション Where フィルタリング条件 Select クエリの結果として選択される要素
For Each 反復変数 In 反復処理するコレクション(ここでは query)
  コントロールの ID.Text &= 反復変数.ToString
Next
    End Sub
End Class
```

まず，パスとファイル名を指定して XML 文書を読み込む。先に述べたように，ここでは XElement クラスを用いている。次に，From 句を使って，問い合わせの対象となるコレクションをデータソースに指定し，対象コレクション内の範囲変数を指定する。条件を指定してクエリを返す必要がある場合は，Where 句でフィルタリングし，Select 句でクエリ結果を定義することができる。

LINQ to XML では，クエリを定義するのと同時に実行される即時実行もあるが，基本は遅延実行である。クエリ変数にクエリを定義した後，この例では，For Each ステートメントを用いて，処理を実行している。

LINQ to XML の軸

LINQ to XML では，XPath 同様，軸を使って処理対象ノードを指定する。特に，対象コレクションを指定する際には，軸を多用する。そのため，ノード間の関係については十分に理解しておかなければならない（●表 3-12）。

ただし，軸の種類や解釈は，XPath と若干異なる部分があるので，注意が必要である。LINQ における後進軸は，祖先をたどる場合のみである。前の兄弟は，前進軸である。つまり，前の兄弟をポジションによって指定する場合，XPath であれば，コンテキストノードから逆文書順にたどるが，LINQ では，コンテキストノードの兄コレクションの中の文書順にポジションを指定する。

●表 3-12 LINQ to XML の軸

LINQ to XML				XPath	
機能		基本構文	省略構文	種類	基本構文
コンテキストノードの子	前進軸	XContainer.Elements	.（1 個のピリオド）	前進軸	child
コンテキストノードの子孫	前進軸	XContainer.Discendants | XContainer.DiscendantNodes	...（3 個のピリオド）		descendant
コンテキストノードと，コンテキストノードの子孫	前進軸	XElement.DiscendantAndSelf | XElement.DiscendantNodesAndSelf			descendant-or-self
コンテキストノードに続くすべての兄弟ノード	前進軸	XNode.ElementsAfterSelf | XNode.NodesAfterSelf			following-sibling
文書順でコンテキストノードより後にあり，子孫・属性・名前空間を除くすべてのノード	なし				following
コンテキストノードの属性群	前進軸	XElement.Attribute | XElement.Attributes	@		attribute
コンテキストノード自身	前進軸	XContainer.Nodes			self
コンテキストノードの名前空間	なし				namespace
コンテキストノードの親	前進軸	XObject.Parent	..（ダブルピリオド）	後進軸	parent
コンテキストノードの祖先	後進軸	XNode.Ancestors			ancestor
コンテキストノードと，コンテキストノードの祖先	後進軸	XElement.AncestorsAndSelf			ancestor-or-self
コンテキストノードの前にあるすべての兄弟ノード	前進軸	XNode.ElementsBeforeSelf | XNode.NodesBeforeSelf			preceding-sibling
文書順でコンテキストノードより前にあり，祖先・属性・名前空間を除くすべてのノード	なし				preceding

3.6.2 実際の記述例

XML ツリーの生成処理

次に，実際の記述例をいくつか示す。

DOM を用いた前掲●リスト 3-19 の XML ツリー生成処理（●リスト 3-20）を，LINQ を用いて実装するには，2 つの方法がある。●リスト 3-31 は，宣言型の関数型構築の場合である。また●リスト 3-32 は，XML リテラル使用の場合である。前掲の DOM のコードに比べ，生成･追加の順序が分かりやすい。

●リスト 3-19（再掲載） 生成する XML ツリー

```
<?xml version="1.0"?>
<オンライン記事>
    <記事名 公開日="2008-07-07">デザインの 30 年</記事名>
</オンライン記事>
```

●リスト 3-31 関数型構築の LINQ コード

```
Option Strict On
Imports System.Linq
Imports System.Xml.Linq

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        Dim xmldoc As XDocument = New XDocument(New XDeclaration("1.0", "utf-8",
"yes"), New XElement("オンライン記事", New XElement("記事名", "デザインの 30 年", New
XAttribute("公開日", "2008-07-07"))))
        TextBox1.Text = xmldoc.ToString
    End Sub
End Class
```

●リスト 3-32 XML リテラル使用の LINQ コードの一部

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
  Dim xmldoc As XElement = <オンライン記事><記事名 公開日="2008-07-07">デザインの 30 年
</記事名></オンライン記事>
  TextBox1.Text = xmldoc.ToString
End Sub
```

要素ノードリストの取得

また，DOM を用いた抽出処理を，LINQ で置き換えてみる。●リスト 3-33 は，既出の●リスト 3-1 の XML 文書の中から<タイトル>を抽出する●リスト 3-21 の処理を，LINQ で実装した例である。

●リスト 3-33 <タイトル>要素の内容テキストを抽出する LINQ クエリ

```
Option Strict On
Imports System.Linq
Imports System.Xml.Linq

Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim filePath As String = Server.MapPath("./")
        Dim xmldoc As XElement = XElement.Load(filePath & "App_Data/SampleData_Article.xml")
        Dim query As IEnumerable(Of XElement) = From c In xmldoc.Descendants("タイトル") Select c

        For Each titleNodes In query
            TextBox1.Text &= titleNodes.Value & vbCrLf
        Next
    End Sub
End Class
```

3.6.3 XPath 拡張機能の利用

Extensions クラスの概要

XML の構造によって，XPath 式を用いた方が処理対象ノードを指定しやすい場合は，System.Xml.XPath 名前空間の Extensions クラスを用いることがある。これは，LINQ to XML の XPath 式を評価する拡張メソッドである。ただし，LINQ to XML のクエリを用いる方が，パフォーマンスはよいとされている。

Extensions のメンバには，CreateNavigator，XPathEvaluate，XPathSelectElement，XPathSelectElements がある。次にいくつかの記述例を示す。

XPathSelectElements メソッドの使用例

●リスト 3-34 は，前掲●リスト 3-33 の LINQ クエリを，Extensions を使って置き換えたコードである。XPath 式により要素のコレクションを返す，XPathSelectElements メソッドを用いている。

●リスト 3-34　前掲●リスト 3-33 の LINQ クエリを XPathExtension で置き換えた例

```
Option Strict On
Imports System.Xml.Linq
Imports System.Linq
Imports System.Xml.XPath

Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim filePath As String = Server.MapPath("./")
        Dim xmldoc As XElement = XElement.Load(filePath & "App_Data/SampleData_Article.xml")

        For Each titleNodes In xmldoc.XPathSelectElements("//タイトル")
            TextBox1.Text &= titleNodes.Value & vbCrLf
        Next
    End Sub
End Class
```


XPathEvaluate メソッドの使用例

XPath 式を評価するには，XPathEvaluate メソッドを用いる。これは，取得したノード群をドキュメント順で返すメソッドである。

ノードの種類を問わず，全ノードを順次処理するには，●リスト 3-35 のように記述する。ルート要素を処理するため，処理対象とする XML 文書ファイルは，XDocument.Load で読み込んでいる。

実行結果は，●図 3-12 の通りである。<公開記事>要素ノード，その子である<オンライン記事>要素ノード，<記事名>要素ノード，<記事名>要素の内容であるテキストノード，<記事名>要素の後の兄弟である<目次>要素ノード．．．というように，全てのノードが順次処理されている。

●リスト 3-35　全てのノードを順次取得する LINQ クエリ

```
Option Strict On
Imports System.Xml.Linq
Imports System.Linq
Imports System.Xml.XPath

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
```

```
        Dim filePath As String = Server.MapPath("./")
        Dim xmldoc As XDocument = XDocument.Load(filePath & "App_Data/SampleData_Article.xml")

        Dim pathEpr1 = DirectCast(xmldoc.XPathEvaluate("/descendant::node()"), IEnumerable)
        For Each result As XNode In pathEpr1
            TextBox1.Text &= result.ToString & vbCrLf & vbCrLf
        Next
    End Sub
End Class
```

●図 3-12 リスト 3-35 の LINQ クエリの実行結果

```
<公開記事>
  <オンライン記事>
    <記事名>デザインの30年</記事名>
    <目次>
      <タイトル 年代="1970年代後半">手計算と手作業</タイトル>
      <タイトル 年代="1980年代前半">制作工程の分業</タイトル>
      <タイトル 年代="1980年代後半">ワープロの普及</タイトル>
      <タイトル 年代="1990年代前半">DTPの普及</タイトル>
      <タイトル 年代="1990年代後半">Webサイトの連携</タイトル>
      <タイトル 年代="2000年代前半">情報デザインの始まり</タイトル>
    </目次>
  </オンライン記事>
</公開記事>

<オンライン記事>
  <記事名>デザインの30年</記事名>
  <目次>
    <タイトル 年代="1970年代後半">手計算と手作業</タイトル>
    <タイトル 年代="1980年代前半">制作工程の分業</タイトル>
    <タイトル 年代="1980年代後半">ワープロの普及</タイトル>
    <タイトル 年代="1990年代前半">DTPの普及</タイトル>
    <タイトル 年代="1990年代後半">Webサイトの連携</タイトル>
    <タイトル 年代="2000年代前半">情報デザインの始まり</タイトル>
  </目次>
</オンライン記事>

<記事名>デザインの30年</記事名>

デザインの30年

<目次>
  <タイトル 年代="1970年代後半">手計算と手作業</タイトル>
  <タイトル 年代="1980年代前半">制作工程の分業</タイトル>
  <タイトル 年代="1980年代後半">ワープロの普及</タイトル>
  <タイトル 年代="1990年代前半">DTPの普及</タイトル>
  <タイトル 年代="1990年代後半">Webサイトの連携</タイトル>
  <タイトル 年代="2000年代前半">情報デザインの始まり</タイトル>
</目次>

<タイトル 年代="1970年代後半">手計算と手作業</タイトル>

手計算と手作業

<タイトル 年代="1980年代前半">制作工程の分業</タイトル>

制作工程の分業

<タイトル 年代="1980年代後半">ワープロの普及</タイトル>

ワープロの普及

<タイトル 年代="1990年代前半">DTPの普及</タイトル>

DTPの普及

<タイトル 年代="1990年代後半">Webサイトの連携</タイトル>

Webサイトの連携
```

<公開記事>要素ノード

<オンライン記事>要素ノード

<記事名>要素ノード

<記事名>要素内容のテキストノード

<目次>要素ノード

<タイトル>要素ノード

<タイトル>要素内容のテキストノード

また，全てのテキストノードを取得するには，●リスト 3-36 のように記述する。

●リスト 3-36　全てのテキストノードを取得する LINQ クエリ

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
  Dim filePath As String = Server.MapPath("./")
  Dim xmldoc As XElement = XElement.Load(filePath & _
  "App_Data/SampleData_Article.xml")

  Dim pathEpr1 = DirectCast(xmldoc.XPathEvaluate("/descendant::text()"), _
  IEnumerable).Cast(Of XText)()
    For Each result As XText In pathEpr1
      TextBox1.Text &= result.ToString & vbCrLf
    Next
End Sub
```

全てのノードを取得するクエリ式を指定し，条件分岐によって，要素ノードのみ，あるいはテキストノードのみを取得するには，XObject のノード型を取得する機能を持つ，XNode クラスの NodeType プロパティを用いる。●リスト 3-37 は，テキストノードを取得する例である。この If result.NodeType = System.Xml.XmlNodeType.Text Then ～ End If という条件文は，If result.NodeType = 3 Then ～ End If のように，NodeType に「3」を指定しても同じ結果を得られる。

テキストノードではなく，全ての要素ノードを取得するのであれば，NodeType には「1」を指定する。

●リスト 3-37　ノード型を判別する LINQ クエリ

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
  Dim filePath As String = Server.MapPath("./")
  Dim xmldoc As XDocument = XDocument.Load(filePath & "App_Data/SampleData_Article.xml")
  Dim pathEpr1 = DirectCast(xmldoc.XPathEvaluate("/descendant::node()"), IEnumerable)
    For Each result As XNode In pathEpr1
      If result.NodeType = System.Xml.XmlNodeType.Text Then
        TextBox1.Text &= result.ToString & vbCrLf & vbCrLf
      End If
    Next
End Sub
```

LINQ to XML は強力である。実際，億単位のノード数のネイティブ XML 文書ファイルも処理することができる。もちろんのこと，その機能も膨大であり，本書の限られたページ内で紹介することはできない。オンライン記事や書籍，Visual Studio のヘルプを参照して，使いこなすことをお勧めする。

3.7 XML Schema による定義

3.7.1 XML Schema の概要

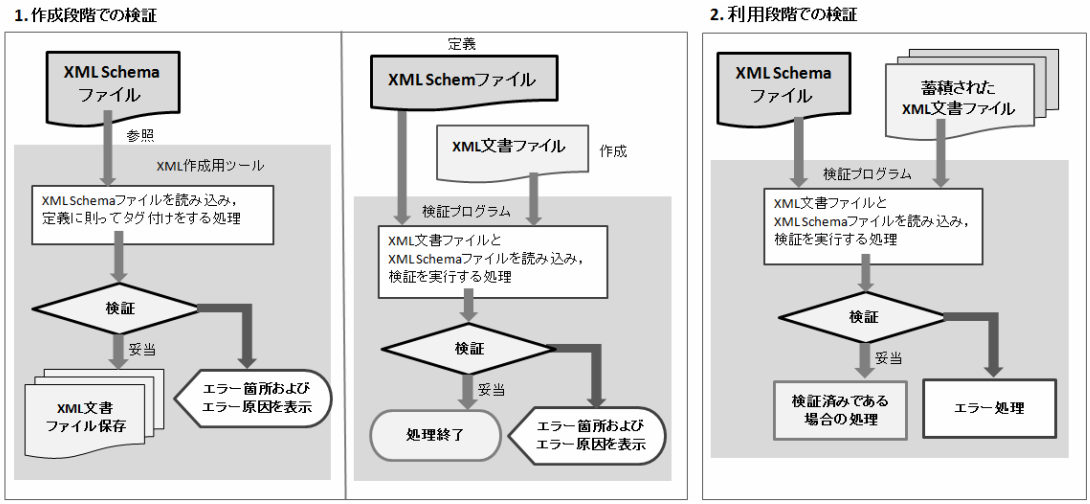
XML Schema の役割

本章では，これまで XPath，XQuery，DOM，XSLT，LINQ to XML といった，XML データ処理のための仕様を見てきた。それらの機能を効率よく用いるには，処理対象とする XML 文書における，メタデータの名前や意味が統一されていなければならない。RDB のデータベース設計においてリレーションシップとフィールド名とデータ型等を定義するように，XML においても，階層構造とメタデータの名前とデータ型等を定義した後，設計にしたがって文書を作成する必要がある。

そこで，設計者は，あらかじめ，階層構造および要素名や属性名，出現順序等の条件を定義する。そして，この定義を文書として作成する。このような XML 文書の構造と条件を定義する仕様に，DTD や Relax，XML Schema がある。このうち，本章では，W3C 仕様であって広く用いられている XML Schema について触れる。XML Schema は，DTD と異なり，データ型や制約も定義でき，名前空間にも対応しているので，細かい条件を定義することができるという利点がある。

XML Schema 文書は，XML 文書の検証に役立つ。XML 文書の作成段階で，XML Schema 文書によって定義された条件を満たしているかどうかを検証すれば，条件を満たさないデータをフィルタリングでき，蓄積される XML 文書の品質を保つことができる。また，XML データを利用する際に，定義された条件を満たしているかどうかを XML Schema 文書を参照して検証すれば，条件を満たさない XML 文書を特定し，誤りの原因を追究することができる。XML Schema 文書は，登録時と，利用時の二重の検証機構として役立つ仕様である（●図 3-13）。

●図 3-13　XML Schema 文書は，二重の検証機構となる。



データ件数が多く頻繁に更新される公共機関の文書や商取引のデータでは，設計に則らない不正なメタデータやデータが 1 件でも混入すると致命的である。仮に，10 万件の XML データの中に，1 件だけ不正なデータがあったとしても，これを処理するアプリケーションは，誤った結果を返したり，処理を中断

するかもしれない。たとえば，一意識別子として処理に用いる予定の<個人 ID>データが，たった一件分でも未登録であったり 2 回以上繰り返して登録されていると，その発見と修正は容易ではない。

しかしながら，100%正確な XML 文書の作成は不可能に近い。オペレータが膨大なデータを 1 から入力したり，Web アプリケーションから不特定多数のユーザによって入力されたデータにタグを自動的に付加して XML 文書を作成する場合，操作ミスやタイプミスは避けられない。既存の RDB データからシリアライズにより XML 文書を生成する場合であっても，RDB から階層構造を持つ XML へのマッピングは容易ではないので，レガシーデータが完全であろうと結果も正確であるとは限らない（そもそも簡単にマッピングできるような RDB であれば，あえて XML 化する必要性は薄い）。ましてや，Web 上に散在する XML データを処理するケースでは，データ作成者を特定することすらできないこともある。XML Schema の採用は，XML 文書の品質維持のために有効であるといえるだろう。

XML Schema で定義する条件は，XML 文書にもとめる精度に応じて判断する。厳密すぎる定義では，XML Schema による計算上は許容されなくても，人の判断では許容範囲であるデータが生じたときに，フィルタリングされてしまう。あるいは XML Schema 文書の修正が必要となる。逆に，ゆるやかな定義では，不正な構造の不正なデータの入り込む余地が拡大する。XML Schema の効果を最大限に引き出すには，定義可能な条件のうち，XML 文書の作成および処理目的を達成するために，最低限必要な条件を絞り込む必要がある。ここでも，設計者の「判断力」が問われる。

XML Schema 文書作成および検証の必要度

XML Schema を使うに際しては，XML Schema 文書を作成すべきかどうか，検証を行うべきかどうかを，決めておく必要がある。設計にしたがって XML 文書を作成する必要はあるが，設計の文書化，文書化方法としての XML Schema の採用，設計書を参照した検証が，必要とはされないケースもあるからだ。

実務における XML 文書の作成と検証においては，主に，次の 6 つの方法がある。

(1) タグ付けの見本となる XML 文書を作成し，これをもとに，同じ構造で別の文書を作成する。XML 文書の検証は行わない。
(2) XML 設計を XML Schema 文書の形で定義する。Microsoft InfoPath などの XML 文書作成ツールの中で，定義済みの XML Schema を参照して，同じ構造で別の文書を作成する。XML 文書の検証は行わないか，ツールに搭載された半自動的な検証機能を利用するにとどめる。
(3) XML 設計を行い，Microsoft Excel などのビジネスソフトで木構造を表現した仕様書を残しておく。XML 文書の検証は行わない。
(4) XML 設計を XML Schema 文書の形で定義し，仕様書として残しておく。但し，XML 文書の検証は行わない。つまり，仕様書の記述言語としてのみ XML Schema を採用する。
(5) XML の構造を XML Schema・DTD・Relax のいずれかを用いて検討するが，最終的には DTD で定義して，XML 文書の検証を行う。
(6) XML 設計を XML Schema 文書の形で定義し，これを利用して，XML 文書の検証を行う。

XML 文書が妥当であるかどうかを問わない場合，つまり整形式でありさえすればよい場合は，(3)あるいは(4)を採用とするとよい。これらは，小規模なネイティブ XML データを扱う案件では，簡易であり有効な方法である。もし，顧客側が XML 文書ファイル作成を担当するのであれば，(3)の方がよいだろう。

妥当な XML 文書とする必要性があれば，(5)あるいは(6)の方法を採用する。

一般に，「XML Schema を使う」といえば，(6)を指す。

しかしながら，XML 技術者ではない一般の人々（顧客）は，これらの 6 つを混同しがちである。作業のコンセンサスを得る際には，十分な説明が必要である。

XML Schema 仕様書の入手先

XML Schema は，W3C の策定した仕様である。3 部から成り，いずれも 2004 年 10 月 28 日に Second Edition が W3C 勧告となっている。本書執筆時点（2008 年 8 月）では，1.1 はワーキングドラフトである。

「XML Schema Part 0: Primer Second Edition」（http://www.w3.org/TR/xmlschema-0/）」は，XML Schema 言語を使ったスキーマの記述方法を定めたものである。

「XML Schema Part 1: Structures Second Edition（http://www.w3.org/TR/xmlschema-1/）」は，スキーマ･コンポーネント（schema component）の構造の記述および，複数のスキーマのインクルードとインポート，および異なる名前空間を持つスキーマ文書の参照方法を規定したものである。

「XML Schema Part 2: Datatypes Second Edition（http://www.w3.org/TR/xmlschema-2/）」は，XML1.0 における要素や属性の型を特定する DTD のスーパーセットを提供する。XML Schema は，第 2 章で触れた DTD より後に策定され，DTD より豊富な機能を持っている。この仕様書では，データの型システムの構文と例，およびビルトインデータ型の継承関係と定義方法が既定されている。

XML Schema の各仕様書は，「XML Technology＞Schema」コーナから入手することができる。

3.7.2 基本構文

ルート要素および要素

XML Schema 文書は，それ自体，XML 1.0 および名前空間を使って記述する。

ルート要素は<xsd:Schema>であり，その中に出現する各要素の接頭辞 xsd:は，通例として，XML Schema の名前空間宣言「xmlns:xsd="http://www.w3.org/2001/XMLSchema"」に結び付けられる。

ルート要素より階層下の要素とテキストの出現順序や出現回数を規定したものを，内容モデル（content model）と呼ぶ。内容モデルは，<element>，<any>と，オプションの<group>，<choice>，<sequence>，<all>といった各要素（後述）を組み合わせて定義する。オプションの要素は，回数を定義する"minOccurs"属性および"maxOccurs"属性を持ち，入れ子にすることができる。

要素と属性の宣言と定義

要素は<element>要素を使って宣言し，属性は<attribute>要素を使って宣言する。要素の内容モデルおよび，属性，属性の値のデータ型は，型定義（type definition）を用いて記述する。

型定義には，単純型定義（simple type definition）と複合型定義（complex type definition）がある（●表3-13）。

●表 3-13　単純型定義と複合型定義

		単純型		複合型
要素	匿名の型定義	<xsd:element name="要素名"> 　<xsd:simpleType> 　　データ型の派生の定義 　</xsd:simpleType> </xsd:element>		<xsd:element name="要素名"> 　<xsd:complexType> 　　複合型定義の内容 　</xsd:complexType> </xsd:element>
	名前付き型定義	宣言	<xsd:element name="要素名" type="型定義の名前" />	<xsd:element name="要素名" type="型定義の修飾名" />
		型定義	<xsd:simpleType name="単純型定義の名前"> 　　データ型の派生の定義 </xsd:simpleType>	<xsd:complexType name="複合型定義の名前"> 　　複合型定義の内容 </xsd:complexType>
属性	匿名の型定義	<xsd:attribute name="属性名"> 　<xsd:simpleType> 　　データ型の派生の定義 　</xsd:simpleType> </xsd:attribute>		
	名前付き型定義	宣言	<xsd:attribute name="属性名" type="型定義の名前" />	
		型定義	<xsd:simpleType name="単純型定義の名前"> 　　データ型の派生の定義 </xsd:simpleType>	

子要素も属性も持たない場合，つまり要素の内容がテキストのみの場合は，単純型定義を用いる。一方，内容に要素あるいは属性がある場合は，複合型定義を用いる。一般的には，宣言の中で型を定義する「匿名の型定義（anonymous type definition）」を用いるが，宣言と新しい型の生成を分けて定義することもできる。

単純型定義は子要素や属性を持たない場合に用いるのであるから，要素に属性を追加して定義する場合は，複合型定義を用いて属性宣言を記述することになる。そして，単純型定義の"base"属性値として指定した型に基づき，新しい複合型を定義する。属性から見た親の要素に子要素がある場合は，属性宣言は最後に記述する。複数の要素に対して同一名称の属性が使われている場合は，属性グループ定義（attribute group definition）を用いることにより，記述を簡略化することができる。

●リスト 3-39 は，●リスト 3-38 の構造の XML 文書を，複合型定義と属性グループ定義を用いて定義した例である。

●リスト 3-38　異なる要素に，名前が同じである"番号"属性および"見出し"属性のある XML 文書

```
<?xml version="1.0"?>
<公開記事 xmlns="http://www.PROJECTKySS.NET/Schema">
  <章 掲載="許可" 番号="1" 見出し="デザインの 30 年">
    <節 番号="1" 見出し="手計算と手作業">1970 年代後半の制作事情</節>
    <節 番号="2" 見出し="制作工程の分業">1980 年代前半の制作事情</節>
    <節 番号="3" 見出し="ワープロの普及">1980 年代後半の制作事情</節>
  </章>
  <章 掲載="保留" 番号="2" 見出し="プログラミングの 30 年">
    <節 番号="1" 見出し="マシンとメディアの進化">1980 年代前半の開発事情</節>
    <節 番号="2" 見出し="パッケージソフトの普及">1980 年代後半の開発事情</節>
  </章>
</公開記事>
```

<章>要素と<節>要素の両方に"番号"，"見出し"属性がある。

●リスト 3-39　複合型定義を用い，属性および属性グループを定義したスキーマ文書の例

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  ①
targetNamespace="http://www.PROJECTKySS.NET/Schema"  ②
xmlns="http://www.PROJECTKySS.NET/Schema" elementFormDefault="qualified">
<xsd:element name="公開記事">
 <xsd:complexType>  ③
 <xsd:sequence>                                  ④
 <xsd:element name="章" maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:sequence>                                 ④
   <xsd:element name="節" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:simpleContent>
      <xsd:extension base="xsd:string">
       <xsd:attributeGroup ref="articleNo" />  ⑤
      </xsd:extension>
     </xsd:simpleContent>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="掲載" type="xsd:string" use="required" />
  <xsd:attributeGroup ref="articleNo" />  ⑥
 </xsd:complexType>
 </xsd:element>
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
 <xsd:attributeGroup name="articleNo">  ⑦
  <xsd:attribute name="番号" type="xsd:integer" use="required" />  ⑧
  <xsd:attribute name="見出し" type="xsd:string" use="required" />  ⑨
</xsd:attributeGroup>
</xsd:schema>
```

①接頭辞 xsd:は W3C スキーマ名前空間に結び付けられる。

②●リスト 3-31 の XML 文書に関連付ける，XML Schema 文書の名前空間を指定している。XML 文書中のデフォルト名前空間としても指定している。

③内容に要素あるいは属性があるので，複合型定義を使っている。

④<章>と<節>は何回出現してもよい。

⑤<節>要素に付属する"番号"属性と"見出し"属性の定義を，articleNo という名前によって参照している。

⑥<章>要素に付属する"番号"属性と"見出し"属性の定義を，articleNo という名前によって参照している。
⑦属性の定義を<xsd:attributeGroup>要素で括り，articleNo という名前を付けている。
⑧"番号"属性は，整数型で，必須である。
⑨"見出し"属性は，文字列型で，必須である。

要素内容（element content）には，複合型（complex types），単純型（simple types），混合内容（mixed content），内容を持たない空の内容（empty content），内容を規定しない型（anyType）がある。作成する XML 文書中の要素内に，子要素とテキストがある場合は（●リスト 3-40），XML Schema では混合内容として定義する（●リスト 3-41）。complexType 要素の"mixed"属性に ture を指定する。デフォルトは false である。

●リスト 3-40　要素内に，子要素とテキストがある XML 文書の部分

```
～略～
<親要素>
  <子要素>要素の内容テキスト</子要素>
   テキスト
</親要素>
～略～
```

●リスト 3-41　混合内容を定義したスキーマ文書の部分

```
～略～
<xsd:element name="親要素" maxOccurs="unbounded">
<xsd:complexType mixed="true">
 <xsd:sequence>
 <xsd:element name="子要素" type="xsd:string" />
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
～略～
```


3.7.3 データ型および出現方法の定義

制約条件の定義例

データ型には，仕様で定められているビルトインデータ型（built-in datatype）とユーザ派生データ型（user-derived datatypes）がある。

たとえば，URI をデータとする属性のある XML 文書（●リスト 3-42）を定義するには，●リスト 3-43 のように記述する。

●リスト 3-42　URI をデータとする属性のある XML 文書

```
<?xml version="1.0"?>
<ルート要素 xmlns="http://www.PROJECTKySS.NET/Schema">
 <親要素 属性="http://www.PROJECTKySS.NET/test/sample.pdf">内容テキスト</親要素>
</ルート要素>
```

●リスト 3-43　URI を定義したスキーマ文書の部分

```
～略～
<xsd:element name="親要素" maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute name="属性" type="xsd:anyURI" use="required" />
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
～略～
```

また，ファセット（facet）を用いて，値の範囲に制約を課すことができる。ファセットには，length，minLength，maxLength，pattern，enumeration，whiteSpace，max Inclusive，max Exclusive，min Inclusive，min Exclusive，total Digits，fraction Digits の 12 個がある。ビルトインデータ型が利用できる制約ファセットの一覧表は，仕様書の「Appendices B Simple Types & their Facets」に掲載されている。

これらのうち，使用頻度が高いのは pattern および enumeration である。pattern 要素の"value"属性値を正規表現により定義すると，値を制約することができる。たとえば属性値が日付であり，その年月日をハイフォンで区切る場合は，データ型に xsd:date を指定する。

また，属性値をスラッシュで区切って，たとえば「公開日="2008/07/07"」のように制約したい場合は，pattern に，\d{4}/\d{2}/\d{2}を指定する（●リスト 3-44）。

enumeration 要素は，ブール型以外の単純型において，重複する値の集合を制限することができるので，選択肢の定義に有効である（●リスト 3-45，リスト 3-46）。

●リスト 3-44　ファセット pattern の使用例

```
～略～
<xsd:attribute name="公開日">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="¥d{4}/¥d{2}/¥d{2}" />
  </xsd:restriction>
 </xsd:simpleType>
</xsd:attribute>～略～
```

●リスト 3-45 "番号"属性値が 1, 2, 3 のいずれかである構造の XML 文書

```
<?xml version="1.0"?>
<公開記事>
  <章 見出し="デザインの 30 年">
    <節 番号="1">手計算と手作業</節>
    <節 番号="2">制作工程の分業</節>
    <節 番号="3">ワープロの普及</節>
  </章>
  <章 見出し="プログラミングの 30 年">
   <節 番号="1">マシンとメディアの進化</節>
    <節 番号="2">パッケージソフトの普及</節>
  </章>
</公開記事>
```

●リスト 3-46 ファセット enumeration を用いて定義した XML スキーマ文書

```
～略～
<xsd:element name="節" maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute name="番号" use="required">
     <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
       <xsd:enumeration value="1" />
       <xsd:enumeration value="2" />
       <xsd:enumeration value="3" />
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
～略～
```

出現順序，回数，初期値の定義例

順序ある出現を定義するには，sequence 要素を使う。出現する内容の選択肢を指定（いずれか一つが出現）する場合は，choice 要素を使う。出現順序が順不同の場合は，all 要素を使う。●リスト 3-47 と●リスト 3-48 は，親要素が繰り返す中に，その順番で必ず出現する子要素が 2 個と，それに続く出現可能要素のうち，<出現可能要素 1>あるいは<出現可能要素 2>の，いずれかが出現する XML 文書である。

出現可能要素の内容としては，それぞれ正の整数，文字列を想定している。この定義例は，●リスト3-49である。

●リスト3-47 正の整数を内容とする<出現可能要素1>がある XML 文書。

```
<?xml version="1.0"?>
<ルート要素 xmlns="http://www.PROJECTKySS.NET/Schema">
  <親要素>
    <長男要素>内容テキスト1</長男要素>
    <二男要素>内容テキスト2</二男要素>
    <出現可能要素1>3</出現可能要素1>
  </親要素>
</ルート要素>
```

●リスト3-48 文字列を内容とする<出現可能要素2>がある XML 文書。

```
<?xml version="1.0"?>
<ルート要素 xmlns="http://www.PROJECTKySS.NET/Schema">
  <親要素>
    <長男要素>内容テキスト3</長男要素>
    <二男要素>内容テキスト4</二男要素>
    <出現可能要素2>内容テキスト5</出現可能要素2>
  </親要素>
</ルート要素>
```

●リスト3-49 choice 要素を用いたスキーマ文書の一部分

```
～略～
<xsd:element name="親要素" maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:sequence><xsd:element name="長男要素" type="xsd:string" />
   <xsd:element name="二男要素" type="xsd:string" />
   <xsd:choice>
     <xsd:element name="出現可能要素1" type="xsd:positiveInteger" />
     <xsd:element name="出現可能要素2" type="xsd:string" />
   </xsd:choice>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>～略～
```

出現回数を制約（occurrence constraints）することもできる。最小回数はminOccurs属性で，最大回数はmaxOccurs属性で定義する。値は正の整数値で，デフォルトは1である。無制限の場合はunbounded を指定する。●リスト3-50は，親要素が3回繰り返し，三男要素が出現してもしなくてもよい場合の定義例である。

●リスト 3-50　回数を定義するスキーマ文書の一部分

```
～略～
<xsd:element name="親要素" maxOccurs="3">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="長男要素" type="xsd:string" />
   <xsd:element name="二男要素" type="xsd:string" />
   <xsd:element name="三男要素" type="xsd:string" minOccurs="0" />
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
～略～
```

要素および属性の初期値は，element 要素あるいは attribute 要素に対して，"default"属性を指定する。"fixed"属性値では固定値を指定することができる。属性が必須であること指定するには，"use"属性値に required を指定する。

以上，次章以降を読み進めるために最低限必要な，XML および XML 周辺仕様についての基礎知識について述べた（※4）。皆さんは，これらの知識をもとに，すぐにでも実際の設計について知りたいかもしれない。だが，知識は，正しい設計を保証するが，良い設計を約束するものではない。

実際の設計に移る前に，次章では，XML 誕生の背景・現状・展望と課題を知ることによって，XML 設計という仕事を掘り下げていく。

※4　本書は，W3C やベンダの技術仕様書の，ダイジェスト版でも補足解説でもない。本書は，決して仕様書の代わりにはならない。したがって，設計時には必ず，必要な仕様書にあたるよう努めてほしい。どの仕様も奥が深いので，完全な理解は難しいが（筆者自身，読む度に気付かされることがあり，まだ理解できていないことが少なからずある），一歩ずつ，理解できそうな部分から学んでいく必要がある。

第 4 章　XML 設計の予備知識

本章では，実際の設計について学ぶ前に，XML 誕生の背景・現状・展望と課題を明らかにし，XML の本質に迫る。

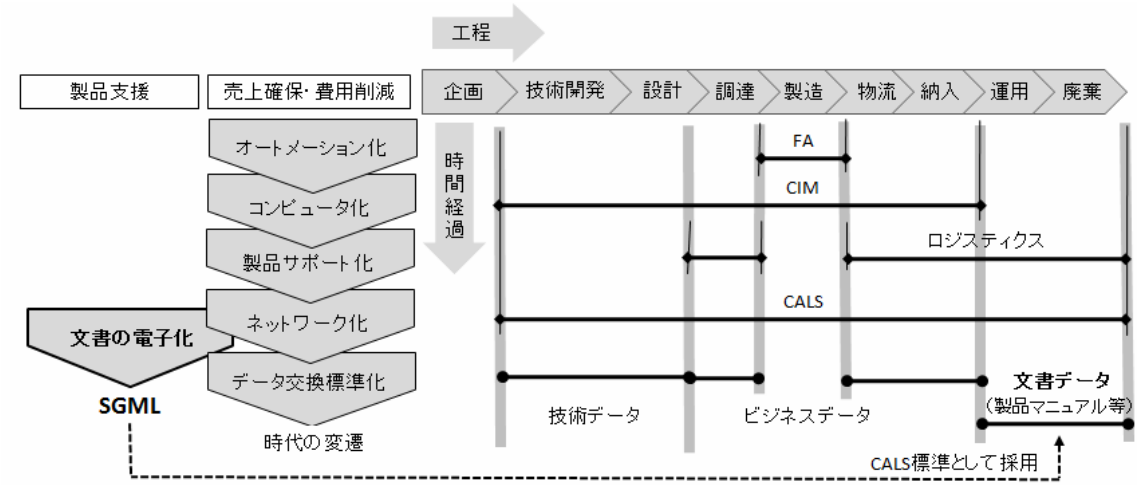
4.1 社会の中の XML～誕生の背景～

4.1.1 ロジスティクス，CALS，SGML

XML 設計は，あくまでも社会に貢献する仕事のひとつである。技術仕様についての知識を身に付けるだけでなく，社会の中での XML 設計の位置付けについて知っておかなければ，仕様に則った誤りのない設計はできたとしても，社会で役に立つ設計にはならない。

そこで，まず，XML 登場以前の，我が国における社会背景から見ていこう（●図 4-1）。そこには，生販物（生産・販売・物流）における「データ交換」と，文書作成における「文書の電子化」という，2 つの流れがあった。

●図 4-1 XML 登場までの背景。ロジスティクス，CALS，SGML



1980 年代初頭，パソコンとフロッピーディスクが普及すると，製造工程を持つ企業は，次々と合理化に取り組み始めた。顧客のニーズが細分化し始めることを察知した企業は，多品種少量生産のために，販売数・在庫数を記録して調整する生産管理に着手した。「FA（Factory Automation，生産工程の自動化）」は，合理化の合言葉となった。

80 年代半ばになると，FA は，CIM（Computer Integrated Manufacturing，コンピュータ統合生産，開発・製造・販売工程の合理化）へと拡大する。大規模な生産ラインを持たない企業でも，設計・製造・管理の一貫合理化が始まった。設計工程のレガシーデータを製造工程に渡して加工用データとして利用したり，設計レガシーデータと管理用詳細データを連携して，受発注や帳票管理に流用するというものである。データを流用することによって，再入力の軽減と，タイプミスによる精度劣化の防止が期待できるようになった。

そして，80 年代後半からは，各工程間で，INS 回線を使ってデータを受け渡すようになった。入出力機器やアプリケーションが次々と開発され，文字列・数値・ラスタデータおよびベクトルデータの画像や図面の混在データベース化が進んだ。当時の主流はカード型データベースであり，RDB 同様，フィールド名でメタデータを表す，フラットな構造であった。

それら設計や管理用のデータは，アプリケーション固有の形式でやりとりされていた。しかし，それでは

設計業者と製造業者が異なるアプリケーションを導入していると，データを再利用することができない。また，業界共通の部品データを資産として活用することもできない。ユーザは，業界標準データ交換フォーマットのサポートを，ベンダに期待するようになった。

一方，文書については，1980年代半ばまで，印刷媒体の制作自体が手作業であった。DTPが一般化したのは 1980 年代後半であり，文書の電子化は，ゆっくりと進んでいた。電子化が最も必要とされたのは，マニュアルである。マニュアルには，取扱説明書やユーザーズ･マニュアルだけでなく，保守や修理に使用されるサービス･マニュアルがある。たとえば電子機器であれば，分解組立図･回路図･実体配線図･部品リストなど，1 モデルだけでも大部数になる。印刷すればさらに嵩が増すものを，製品輸出先へ輸送していたのである。小型の電子機器であればまだしも，複雑なマニュアルが必要となる大型機器や設備において，文書のペーパレス化が待ち望まれたのは当然であろう。

そこへ，CALS（Computer-aided Acquisition & Logistics Support）が登場した。CALS は，Logistics Support という通り，ロジスティクス（物資･人員･設備の効率的な調達･輸送･管理）を目的とするものである。IT を用いたデータ共有とデータ互換により，製品ライフサイクルの合理化を実現するものとして広まっていった。90年代になると，設計･製造業向けシステムを提供する各ベンダが，CALS標準の元になる業界標準フォーマットのサポートを進め始めた。それはたとえば，CAD データであれば IGES（Initial Graphics Exchange Specification，初期グラフィックス交換仕様）であった。システムを導入する企業にとって，CALS標準入出力フォーマットのサポートは，検討事項の一つとなった。

CALSでは，標準の文書電子化フォーマットとして，SGML（Standard Generalized Markup Language，汎用マークアップ言語）が採用された。SGML は，DTD で定義したタグを用いて文書構造を記述するものである。95年前後からは，米国業界標準に対応したマニュアル類のSGML化が，自動車や航空機業界で進む状況を報じる記事が目立ち始めた。なお，国内では95年に，マニュアルの内容と表現に影響を及ぼす，PL法（Product Liability，製造物責任法）が施行されている。

4.1.2 HTML，XML，XHTML

国内でSGMLが話題になり始めた1995年頃から，インターネット接続サービス業者が増え始めた。そして，HTML（HyperText Markup Language）により記述されたWebページが作成されるようになった。

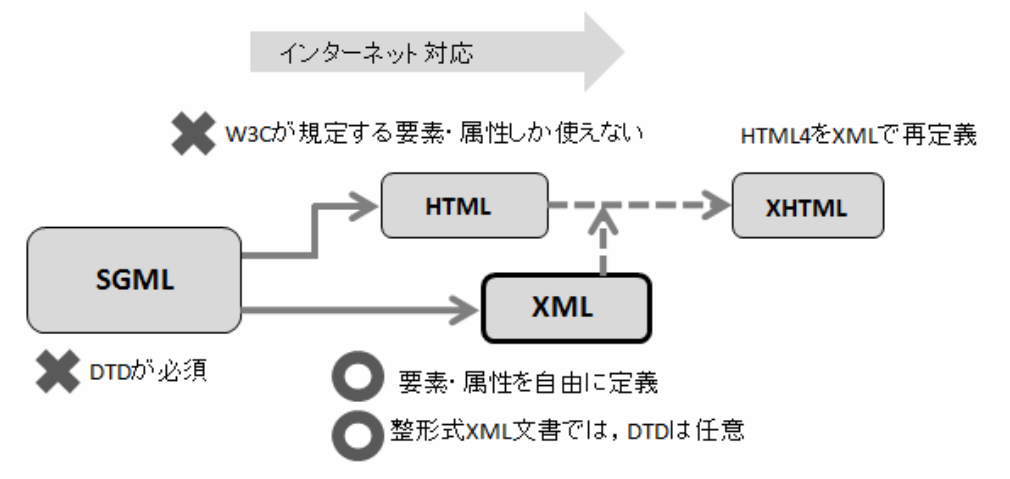
HTMLは，SGMLをルーツとする，World Wide Webを発行するためのマークアップ言語である。HTMLでは，W3C が DTD で規定した要素および属性しか使うことができない。意味の異なるデータにも，同じメタデータを付けざるをえない。たとえば，順序付けリストであれば，<社員リスト>であろうと<商品リスト>であろうと，<ol>要素と<li>要素を使うしかない。これでは，コンピュータは，メタデータによってデータの意味を区別して処理することができない。

そこで，独自タグを扱うことのできる仕様が待ち望まれるようになった。そして策定されたのが，XML である。XMLは，SGMLのサブセットであり，決してHTMLの後継ではない。XMLは，HTMLのようにWeb上で提供され，受信され，処理される，汎用的な SGML を目指して策定されたものである。SGML 文書は，SGML宣言＋前書き（DTD）＋文書本体（インスタンス）から成り，DTDは必須であった。しかし，イン

ターネット対応とするには，簡易で扱いやすい仕様が必要とされる。XML は，インターネット対応であり，第 2 章で解説したように，DTD は必須ではないというシンプル･イズ･ベストを体現するような仕様である。

XML 登場後に発表され，現在 Web ページの記述言語として標準的に用いられている XHTML は，HTML4 を XML1.0 で再定義した仕様である（●図 4-2）。一言でいえば，整形式 XML の文法に則って，HTML のタグを用いる仕様である。

●図 4-2 SGML，HTML，XML，XHTML の関係



4.2 社会の中の XML ~ 普及と活用 ~

4.2.1 社会基盤を支える XML

世界的に見れば，CALS の登場が 1985 年，SGML が国際規格となったのは 1986 年であって，CALS と SGML は，ほぼ同じ歴史を持っていると言われている。CALS 標準の「データ交換」と，SGML による「文書の電子化」が，それぞれ併行して発展してきた到達点に，XML があることになる。

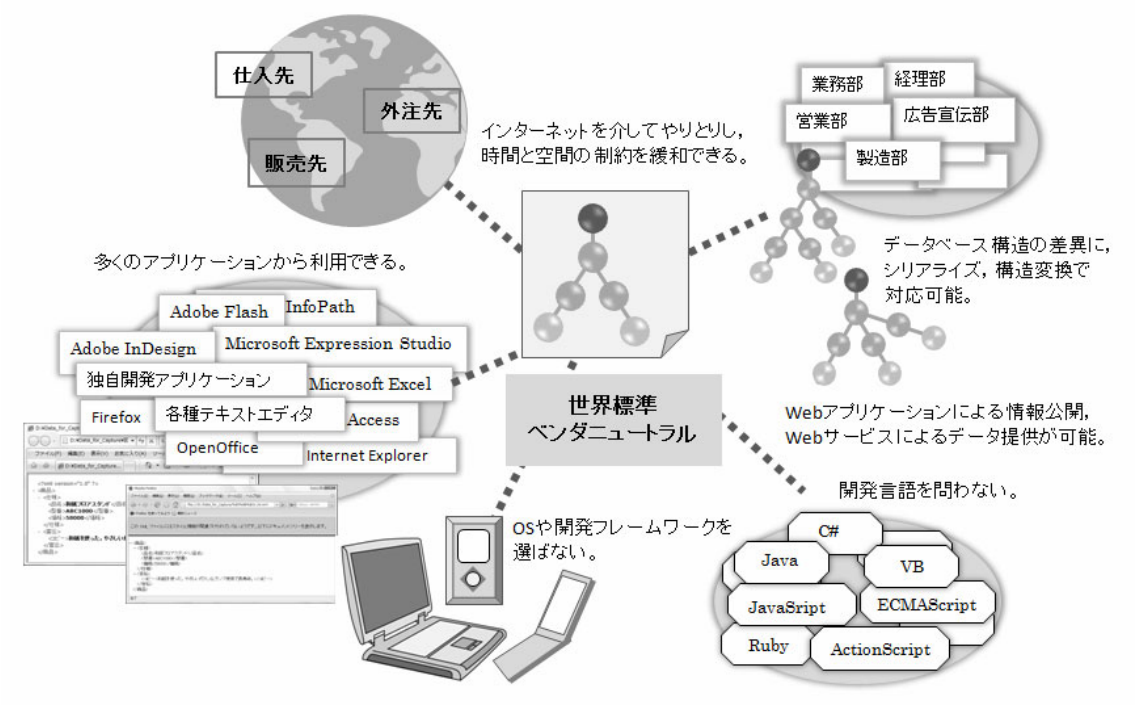
SGML は，電子化による長期蓄積と輸送コストの削減により，文書の利用期間と利用範囲を拡大した。インターネットは，利用時間と利用空間の制約を外した。そして，インターネット対応として SGML から生まれた XML は，両者のメリットを受け継ぎ，さらに利用時間と利用空間の範囲を拡大した。

現在の経済活動は，インターネットを抜きにしては語れない。販売先や OEM 供給先が拡大したり，生産拠点を移転するケースが増えたり，ユーザの居住地が拡大する一方で，ジャスト･イン･タイムがもとめられている。帳票類やマニュアルなどの XML データをインターネットでやりとりすることにより，より効果的なロジスティクスを見込むことができる。XML の誕生は，時代の要請である。

XML は，世界標準であってベンダニュートラルな仕様である。そのため，文化やハードウェアが異なっても利用できる。OS や開発フレームワークや開発言語を問わず，仕入先や外注先や販売先の各業務に適した処理アプリケーション開発に利用することができる。複数の企業の調達･業務･経理といった部門の業務プロセスの違いから，平生の業務で用いるデータベース構造が異なったとしても，構造を変換

することによって流用することができる。さらに，Web アプリケーションや RIA によってデータを公開したり，アプリケーション同士で XML データをやりとりし合うこともできる（●図 4-3）。XML の活用により，資源と労力と費用の合理化を期待することができるのである。

●図 4-3 XML は，CALS と SGML のメリットを受け継ぎ，利用時間と利用空間を拡大した。



XML は，電子カルテ，マニュアル，地図情報案内，建設 CALS，電子商取引などから拡がり始め，いまや，ニュース原稿から，健康診断情報まで，幅広く採用されている。Microsoft Office のようなエンド･ユーザの多いビジネス･アプリケーションでも，XML 形式がサポートされている。さらに，XML は，汎用マークアップ言語であるので，W3C，ベンダおよび各業界団体や学術機関では，XML ベースの言語が策定されている（※1）。いまや XML はシステムやアプリケーションのバックグラウンドで動く，社会基盤を支える仕様となっている。

XML が，1998 年 2 月の W3C 勧告からわずか 10 年で急速に普及したのは，以上のような XML の特長が，社会のニーズに合致したからであろう。

なお，ここに記した XML 誕生の背景については，筆者が記憶をたどりつつ経験を書きとめたものである。SGML から XML へと移行した構造化文書専門のテクニカルライタや，RDB を手始めにして XML も手掛けるデータベース･エンジニアでは，また異なる見方となるであろう。

※1：たとえば，数式を表現する MathML（Mathematical Markup Language），ニュース管理の標準フォーマットである NewsML がある。

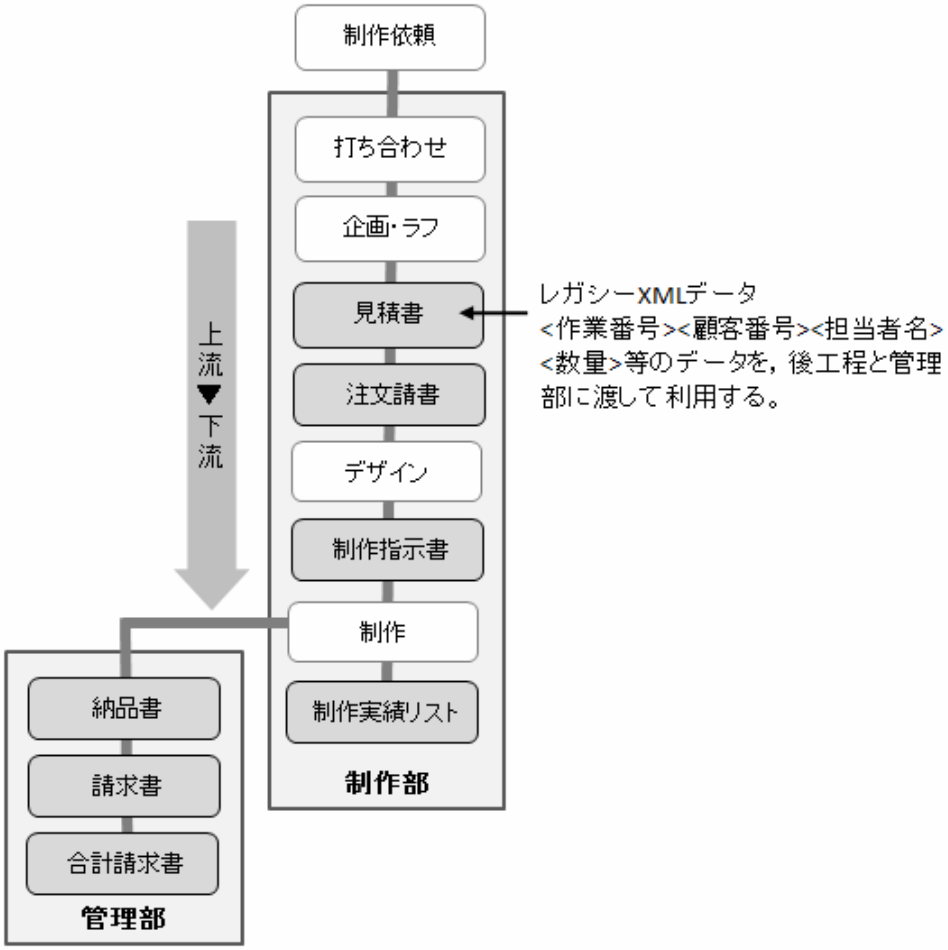
4.2.2 ワンソース・マルチユース

木構造型のマルチユース

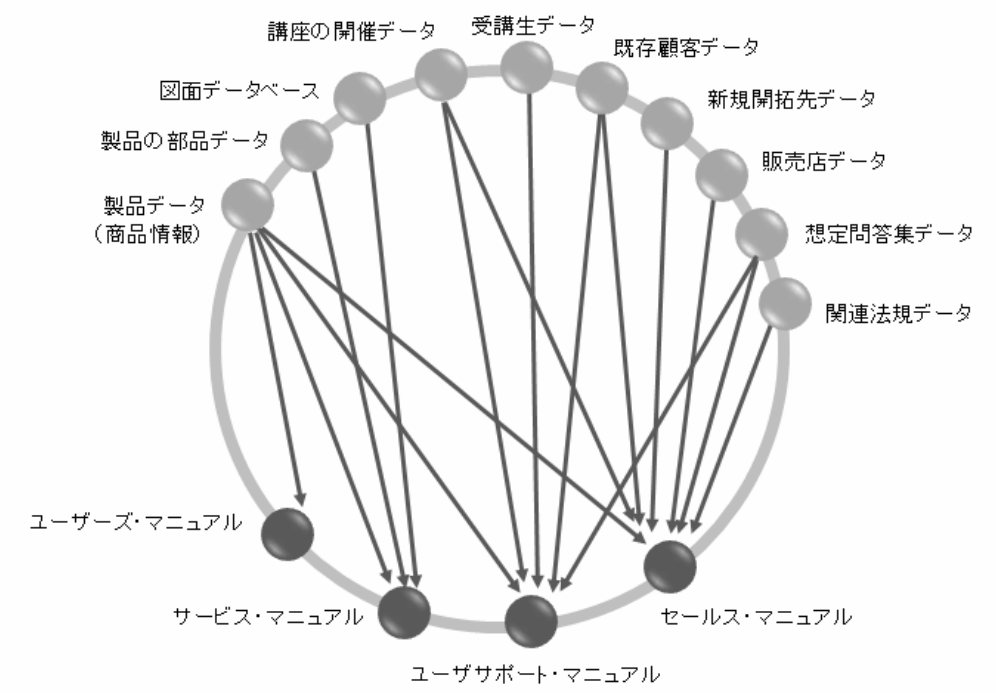
XML データの基本の活用方法は，レガシーXML データを上流工程から下流工程へ渡していく，木構造型のワンソース･マルチユースである。レガシーXML データは，後工程や別部門で変換や編集によって利用される。そのデータの動きにもシーケンスがある。設計データを製造工程まで利用したり，仕入れ管理･在庫管理･売上管理を連動させるといった方法が，これにあたる。●図 4-4 は，1 点モノの制作において，<作業番号><顧客番号><担当者名><数量>といったレガシーXML データを後工程や別部署で流用する，極めて簡単な例である。

また，木の根と葉の上下を逆転させた，バオバブの木のような，利用方法も考えられる。複数の異なるレガシーXML データを，1 個の目的に利用する形のマルチユースである。●図 4-5 に，各種帳票データや図面データを，マニュアル類に利用する例を示す。図中に，天地逆転した木のあることが見て取れる。

●図 4-4 見積書から合計請求書発行までデータを流用する最も簡単な例。



●図 4-5 複数の異なるレガシーXML データを，それぞれ 1 個の文書に利用する例。



ネットワーク型のマルチユース

XML 文書そのものは木構造であるが，XML はインターネット対応の仕様であるから，その活用範囲はネットワークのように広がることもある。ネットワーク型のマルチユースでは，一企業内のみならず，異分野・異業種の取引先やユーザが，レガシーXML データを活用していく。そして，複数の目的で変換や編集により利用した XML の部分木を，さらにフィードバックして再利用する。木構造型のマルチユースと異なり，ネットワーク型のマルチユースには一方向のシーケンスはなく，あらゆる方向にデータの利用範囲が拡大していく。

一例として，商品データを記録した 1 個の XML 文書ファイル中の，<品名>と<型番>データの利用範囲が拡大していく様子を示す（●図 4-6）。

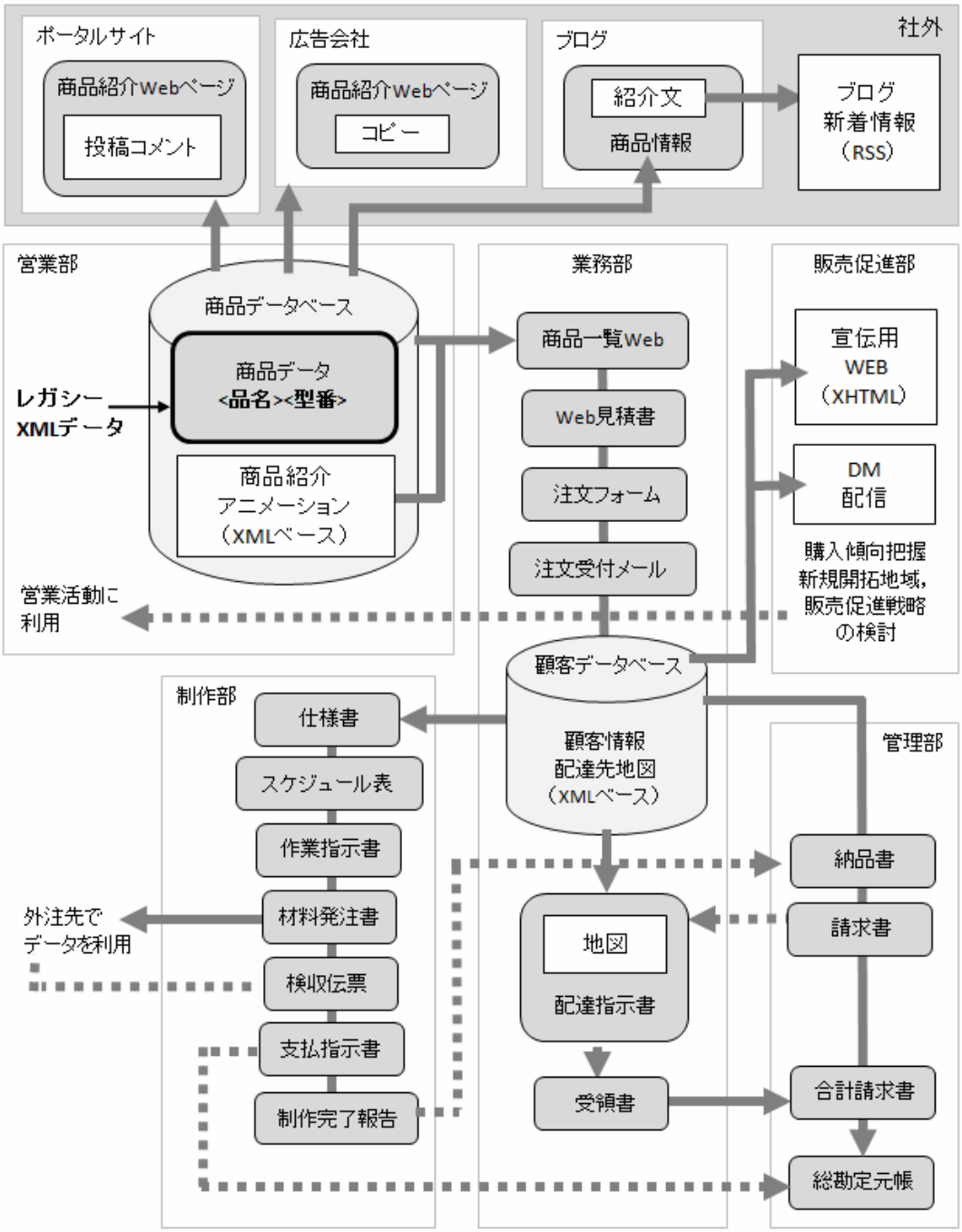
まず，<品名><型番>データは，商品一覧の案内ページで，商品データの表示と検索に利用される。その際，商品紹介アニメーションも XML ベースの仕様で構築することが可能である。商品データと絵や音のデータといった，一見無関係に見える文書データ同士であっても，それが XML 仕様に則って作成された文書であれば，プログラムによって連携させることは比較的容易である（※2）。

注文された商品の<品名><型番>データは，作業指示書や外注用の伝票類に利用することができる。納品の際には，配達先の詳細な地図や現地写真データの URI を参照可能な顧客データベースを用いて，配達指示書を作成する。これはモバイル・アプリケーションとして実装することも可能であろう。それらのデータは，購入傾向の把握，ダイレクトメールの配信，新規開拓地域，販売促進戦略の検討に利用することができる。

レガシー商品データを管理している企業の外側からも，データの利用は可能である。XML データは，異なるサーバに蓄積されていたとしても利用することができる。商品販売のポータルサイトや広告制作会社が，商品データを利用して宣伝用の Web ページを作成したり，Web フォームからポストされた商品

に対するコメントのデータを，検証したうえで XML 文書化して蓄積することも考えられる。専属契約ブロガーが商品データの紹介記事を書き，記事の更新情報が XML 形式の RSS で配信されることもあるだろう。

●図 4-6 レガシー商品データが利用範囲を拡大していく例。



※2：ベクターグラフィックスやアニメーションを記述する XML 形式の言語として，Microsoft 社の独自仕様である XAML（ザムル，Extensible Application Markup Language）がある。これを用いて，Silverlight アプリケーションを開発することができる。

Topic XML の 10 の目標

XML 誕生の背景を見てきたところで，XML の設計目標を確認しておこう。XML 仕様書の冒頭「1.1 Origin and Goals」には，次のような 10 個の設計目標（The design goals for XML）が掲げられている。1.と 2.が時間空間の制約を取り払い，4.と 6.がコンピュータによる処理とヒトによる理解をもとめていることを，見て取ることができる。3.では，母胎となった SGML との互換性がうたわれている。

XML 文書は，ハードウェアの中にとどまらず，ネットワークを介して国境を越え，部門・企業・業種を超える。さらに，長期間蓄積した暁でも利用することができる。XML は，who と when と where を変える。「誰が」———ヒトとコンピュータの両方が，「いつ」———時間にとらわれることなく，「どこで」———場所にとらわれることなく，使うことができるのだ。

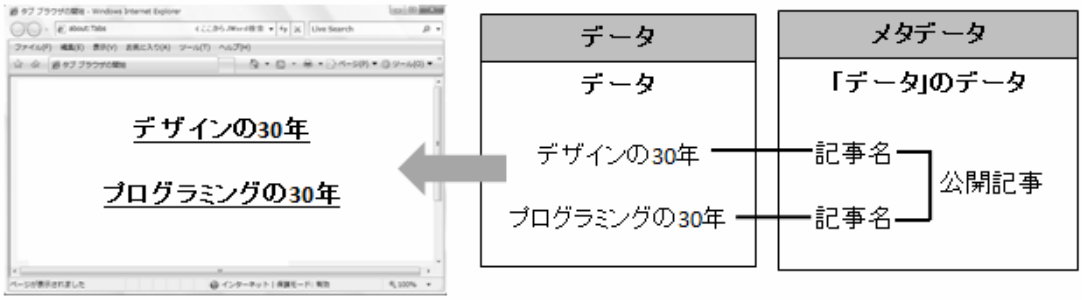
1.XML shall be straightforwardly usable over the Internet.
2.XML shall support a wide variety of applications.
3.XML shall be compatible with SGML.
4.It shall be easy to write programs which process XML documents.
5.The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6.XML documents should be human-legible and reasonably clear.
7.The XML design should be prepared quickly.
8.The design of XML shall be formal and concise.
9.XML documents shall be easy to create.
10.Terseness in XML markup is of minimal importance.
(W3C 仕様書より)

4.3 社会の中の XML ~役割と課題~

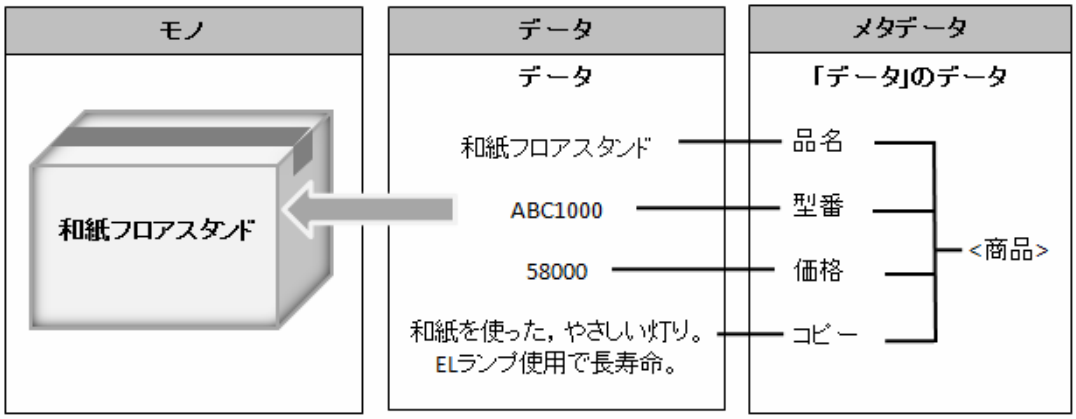
4.3.1 製品ライフサイクルとデータ・ライフサイクル

オンライン記事や公開文書などのように XML 文書そのものが最終製品である場合や，XML をデータ交換フォーマット目的にのみ用いる場合は，前章までの解説に用いたサンプル文書（●図 4-7）のように，データとメタデータだけで完結する。一方，ヒト・モノ・カネといったマクロな物体の物理的な動きと連動して用いられる場合は，●図 4-8 のように，「物質を表す」データがあり，そのうえに「データの意味を表す」メタデータがあるという形になる。現実には，データのみで完結するのではなく，ヒト・モノ・カネの関わる案件の方が多いと思われる。

●図 4-7（第一章●図 1-19 再掲載）データと，データを表すメタデータ。



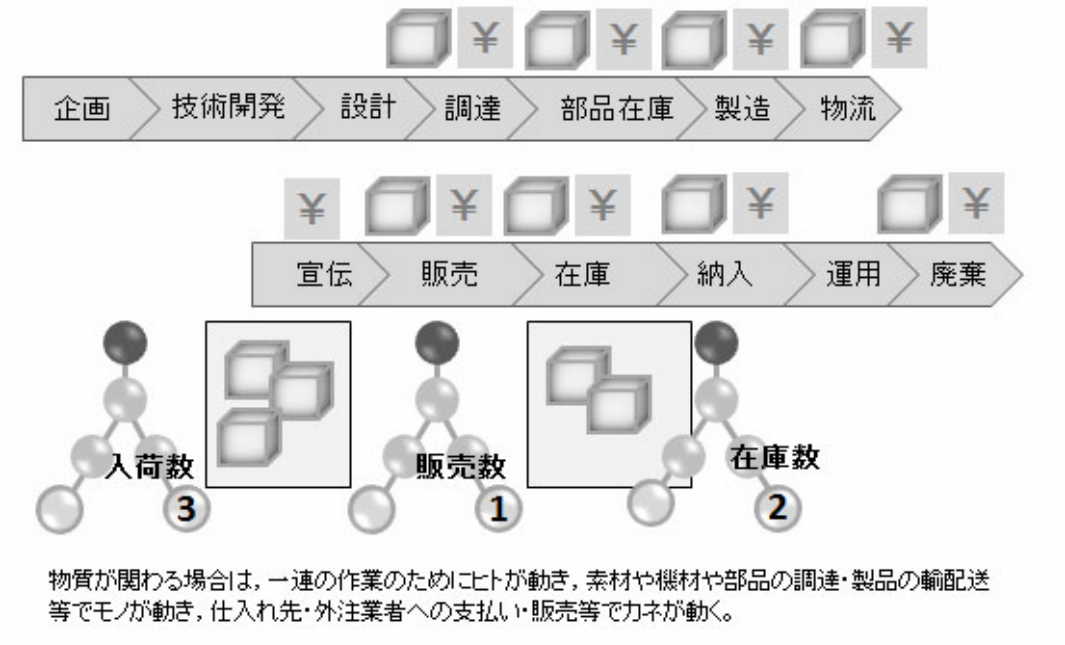
●図 4-8 物質を表すデータと，データを表すメタデータ。



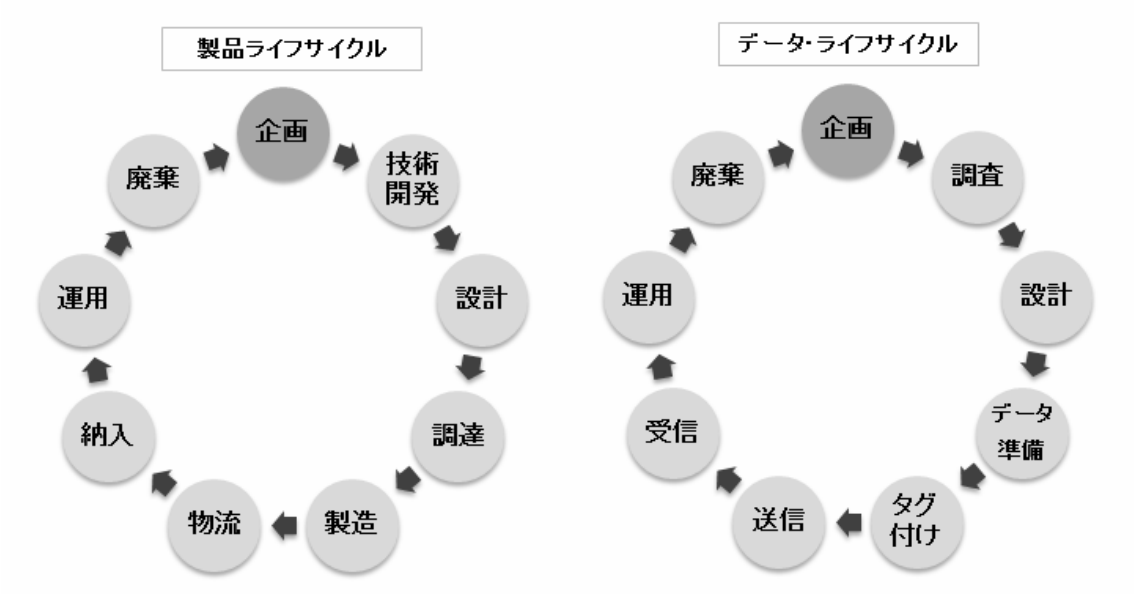
物質が関わるケースでは，XML データは，実世界での物理的変化に伴い，逐次変化する。たとえば製品が入荷されたり，販売されると，<入荷数>や<在庫数>のデータが変わる（●図 4-9）。データおよびメタデータは編集・追加・削除される。

製品にライフサイクルがあるように，データにもライフサイクルがある。ロジスティクスの考え方は，物資・人員・設備だけではなく，電気・磁気といった信号にも応用することができる。XML データの効率的な設計・生成・運用は，データ・ライフサイクルの合理化を実現するものとなる（●図 4-10）XML は，メタデータの付加されたデータのロジスティクスを実現する可能性を秘めているのである。

●図 4-9 XML データは，フィジカルな動きと連動して，逐次変化する。



●図 4-10 製品ライフサイクル（生販物のロジスティクス）と，データ・ライフサイクル（信号のロジスティクス）



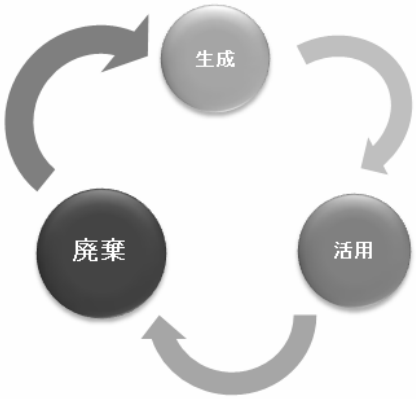
4.3.2 CALS の教訓

前述のとおり，製品にライフサイクルがあるように，データにもライフサイクルがある。調達・輸送・管理の対象が，物資・人員・設備ではなく，電気や磁気といった信号であっても，その利点と課題は同じである。

CALS 自体は，製品ライフサイクルの中でも，物質を供給した後の保守・管理や廃棄・撤去を重視するものであった。だが，実際の経済活動では，利益につながる製造・販売の方が，後工程より重視されがちになっている。その偏ったプライオリティは，安全性や環境面の問題を引き起こしかねない危険性を秘めている。

これはデータ・ライフサイクルついても同様である。XML 文書ファイルの生成と活用を，文書やノードの管理と廃棄よりも重視するとどうなるか。作りっぱなしと安易な連携により，使わず，管理できず，削除もできないデータが蓄積され続けることになる。

多くの場合，生成・活用のみ重視した設計作業の方が簡単であり，低コスト短時間で済む。だが，時間を要してでもライフサイクル全体をとらえた構造を考える方が，長期的に見れば賢明である。我々の子孫の世代に利益をもたらすのは，廃棄を重視し，なお且つ，生成・活用・廃棄の三位一体となった設計である（●図 4-11）。XML 文書を作成する前の設計段階から，生成だけではなく，廃棄についても考えておくべきである。



●図 4-11 廃棄までも重視した，生成・活用・廃棄の三位一体となった設計が必要である。

廃棄を重視しなければならない背景には，次章で述べるデータの整合性という技術上の問題だけでなく，データ爆発の問題もある（一般的には「情報爆発」と言われるが，本書では，そのうち信号として扱われる情報についてのみ述べる）。

いまや，蓄積されるデータも流通するデータも，急激に増えている。ブロードバンドの普及や広帯域化，RDID（無線 IC タグ）や電子マネーの普及，ストレージや各種メディアや携帯端末の進化。企業や商店，官公庁では，電子メールから各種書類まで膨大なデータが作成されている。ビジネスデータだけではない。P to P（Personality to Personality）時代に突入し，テキストやビデオや楽曲などのコンテンツを誰でも発信できるようになった。情報の提供者と活用者の境界線が失われ，情報の活用者が情報の提供側にもなる（CGM，Consumer Generated Media）。存在を認めよと叫ぶパーソナリティ文化は，いかなる社会システムをもっても制御できず，瞬く間にゼッタバイト級のデータがあふれることになる。

それらのデータの何割かは，既に XML あるいは XML による言語で記述されているか，もしくは，今後記述されうる可能性がある。XML は，メタデータをセットとするテキストデータであるだけに，データサイズが大きくなる。サイズダウンを可能にするポスト XML の仕様も登場してはいるが，設計者側がサイズダウンに期待して甘んじているだけでは，技術開発とデータ爆発は，薬剤と薬剤耐性の関係になる。

問題は，データの増加だけではない。データが増えるということは，蓄積するためのハコや通信網が必要になるということである。ハードウェアや周辺部品製造のための電力や輸送コストも必要になる。モノを作れば，環境基準を下回るにせよ廃棄物は出る。データの増加は，モノの増加を抜きにしては語れないのである。

我が国の XML の父・村田真氏著の「XML 入門」（1998 年初版）によれば，『CALS と関連づけて SGML や XML を語る人は米国にはほとんどいない』とのことである。逆にいえば，このことこそが，我が国の強みではなかろうか。我々は，CALS の教訓を XML 設計に生かし，データ・ライフサイクルの中の運用管理・廃棄を重視した設計を心がけることができる。

技術者は具現化したがり，科学者は実証したがる。だが，作る前，実験する前に，作られたモノや実験結果の，社会への波及効果や，後世に対する責任も考えるべきである。

これからの社会でもとめられるのは，「how fast, how many」ではなく，「how beautiful」だ。使わなくなったファイルや活用できないジャンク・データが全世界のサーバに点在する様子をありありとイメージし，それを「美しくない」と思う美意識があればこそ，美しい設計にするにはどうすればよいかと工夫を凝らすようになる。「技術哲学立国」は，IT 業界発展の一つの鍵ではないだろうか。

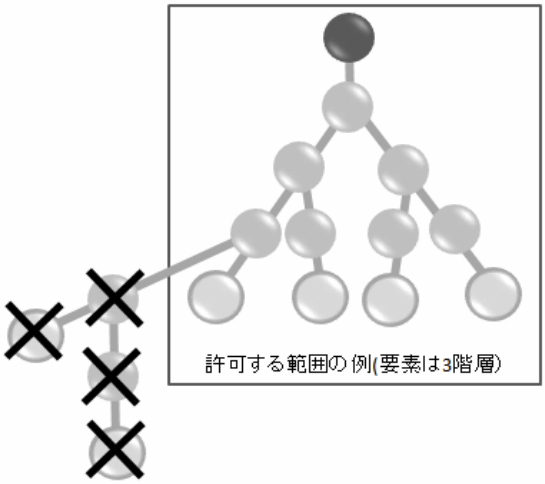
ここでいう美意識とは，我々の暮らすこの世界での美でしかないかもしれないが，それでもとりあえず，我々はこの世界に生きているのであるから，この世界の美の基準に倣うとしようではないか。

4.3.3 データ爆発の抑制

設計とメンテナンスによる抑制

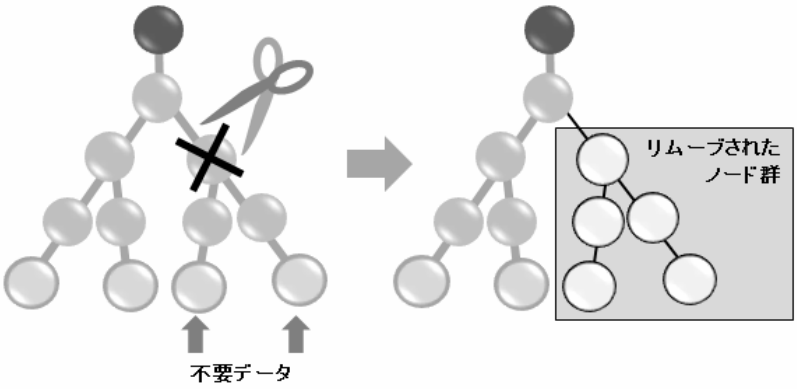
さいわい XML は木構造であるので，次の 2 つの方法によって，多少なりともデータ爆発を抑制することができる。

1 つは，あらかじめ，枝葉を伸ばす範囲（木の最大階層）を設定しておく方法である（●図 4-12）。スキーマ定義によって繰り返し回数を制限したり，アプリケーションによって設定した階層以上のデータを追加させないようにするのである。



●図 4-12 木の枝葉を伸ばす範囲を決めて，ノード爆発を防ぐ。

もう 1 つは，迂回路がないという木の特長を生かし，データが追加される枝を 1 本に絞り込む設計にしておくのである。木構造では，枝をたどる経路が 1 種類に限られるため，1 個のノードをリムーブするだけで，それ以降の経路を一気に断つことができる（●図 4-13）。n 個のデータを削除するために，n 個のデータが付属する n 本の枝を操作するよりも，n 個のデータを子として持つ 1 本の枝を操作すれば済む構造が望ましい。データ爆発を未然に防ぐための基本的な方法は，計画的な植樹（設計）と適切な剪定（メンテナンス）である。



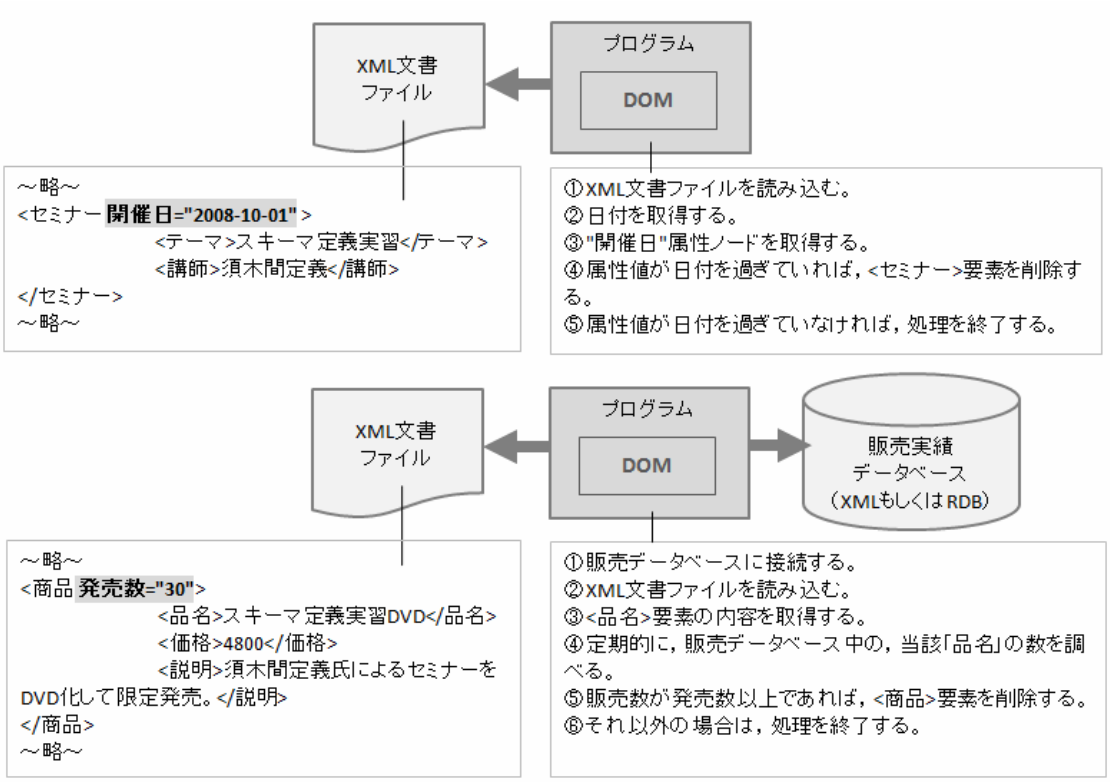
●図 4-13 迂回路がない木構造の特長を生かして，最小の処理で経路を断てるようにする。

メタデータによる廃棄の判断

ノードをリムーブするには，要素名や属性名を，ゴミを分別する際の「プラスティック」や「アルミ缶」マークのように目印として扱い，ノードを分別処理する。廃棄対象となるデータが含まれるノードを特定して，削除を実行するのである。

作成後に更新されることのないデータ，たとえば催事データを例にとれば，開催日を属性値として付加しておくことにより，終了翌日に自動的にリムーブする処理を実装することができる。限定発売の商品データであれば，販売数を属性値として付加しておくことにより，在庫がなくなり次第，商品データをリムーブすることもできる（●図 4-14）。

●図 4-14 廃棄の可否を判断できる属性を追加しておくと，自動的に処理できる。



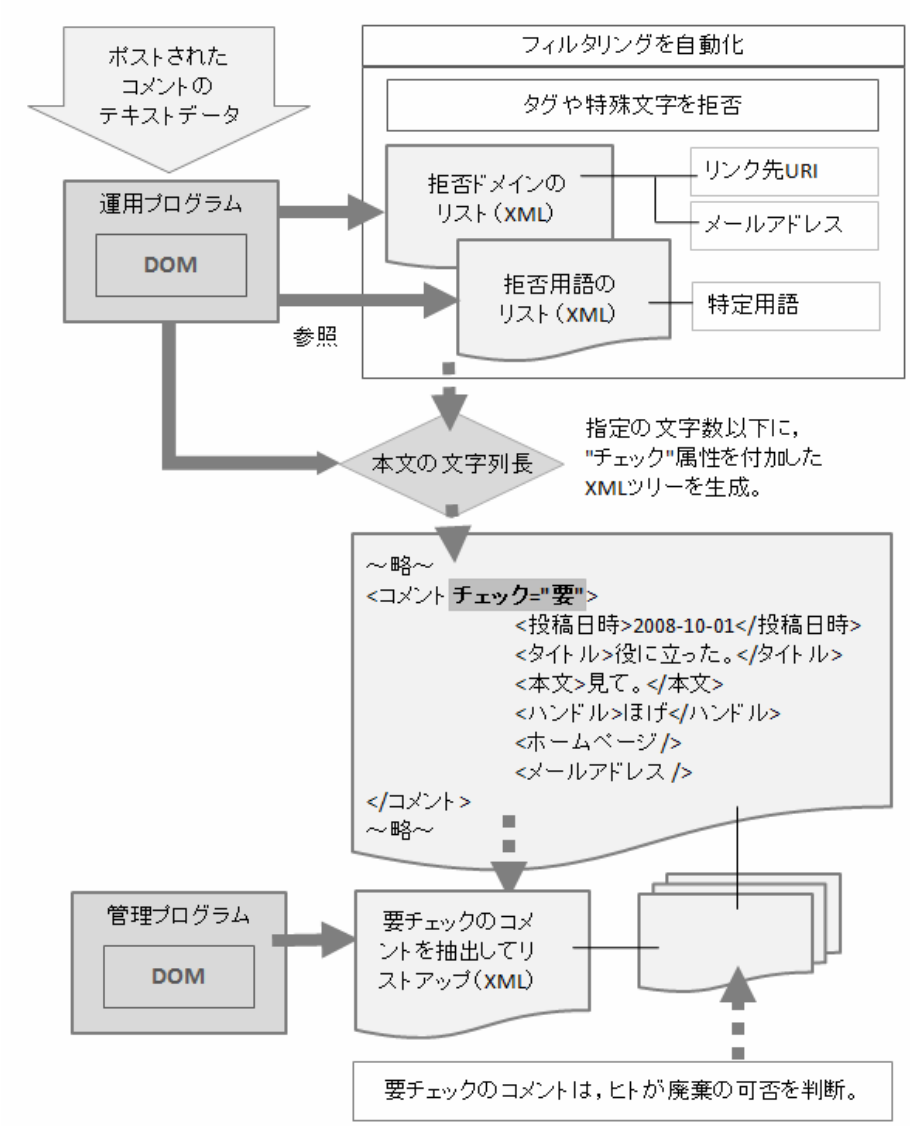
一方，随時更新されるデータでは，ノードをリムーブする前に，廃棄の可否をコンピュータに計算させ，さらにコンピュータで判断できないデータについては，ヒトが判断しなければならない場合がある。

たとえば，Web 上に一般公開している企業ブログのコメント欄を例にとると，有益なフィードバックだけでなく，テーマとは異なるコメントがポストされることがある。「ヒトが内容を見るまで有益か無益か分からない」データが蓄積されていく。

コメントに特定のリンク先 URI やメールアドレス，特定の言葉が含まれているといった条件の明確なデータならば，廃棄処理を自動化することはできる。しかしながら，条件を指定できるものばかりではなく，廃棄基準をくぐりぬけるデータが出てくる。そこで，一定字数以下のコメントデータを自動的に削除するとしよう。すると，一定文字数以下であっても，含蓄のある漢字だけの数語のコメントがフィルタリングされてしまう。内容の有益性を区別する自動化処理は困難である。いずれはボトムアップ型プログラムによって高精度の判断が可能になるが，現時点では，人力は不可欠といえるだろう。

そこで，自動化処理による廃棄基準をくぐりぬけた，一定字数以下のデータに対して，ヒトによる判断を必要とすることを表す属性を追加しておくという方法が考えられる。定期的にこの属性値を持つデータのみを抽出して表示し，ヒトが廃棄の有無を判断して，削除を実行できるようにするのである（●図4-15）。現時点では，こういった削除条件が不明確な処理については，コンピュータにより計算できない部分を，ヒトの判断で補うのが妥当であろう。

●図 4-15 コンピュータにより計算できない部分を，ヒトの判断で補える設計にする。



ただし，この方法では，正しい日本語の長文でありながら，内容が適切ではないコメントを防ぐことはできない。また，これは，データの管理者という判断を担うヒトがいてこそ可能になる。

Web 上で流通する管理者不在の文書となると，大きな課題がある。不特定多数の文書作成者「全員」が，運用・廃棄の意義を理解し，実践することは不可能である。その試みが実現すれば，Web は言葉の最終処分場ではなく，知の集積（※3）になるのであろうが，これは，教育だけでは解決しえない問題である。

もし，情報の発信者の全員が，良心があり正しい文法を身に付けて豊富な語彙を持っていたとしても，事実を認識して理解する機構にトラブルが発生すると，非事実であるデータを蓄積して発信する可能性が捨てきれない（「非事実」≠「嘘」）。たとえば，デスマーチで何晩も徹夜すれば，比較的誰でも，ふとぼ

んやりして話を聞き間違えてしまうことがあるように，知の集積を実現するための最大のハードルは，ヒトの脳の認知の問題かもしれない。

いかなる技術も，それを利用するユーザ次第で，メリットの生じることも，デメリットの生じることもある。XML 設計者が，良い設計を極めようとするならば，XML 文書および XML データ処理アプリケーションを利用するユーザの認知の問題を避けて通ることはできない。

※3：知のデータベース化を目的とする W3C 仕様に，「セマンティック・ウェブ（Semantic Web）」がある。

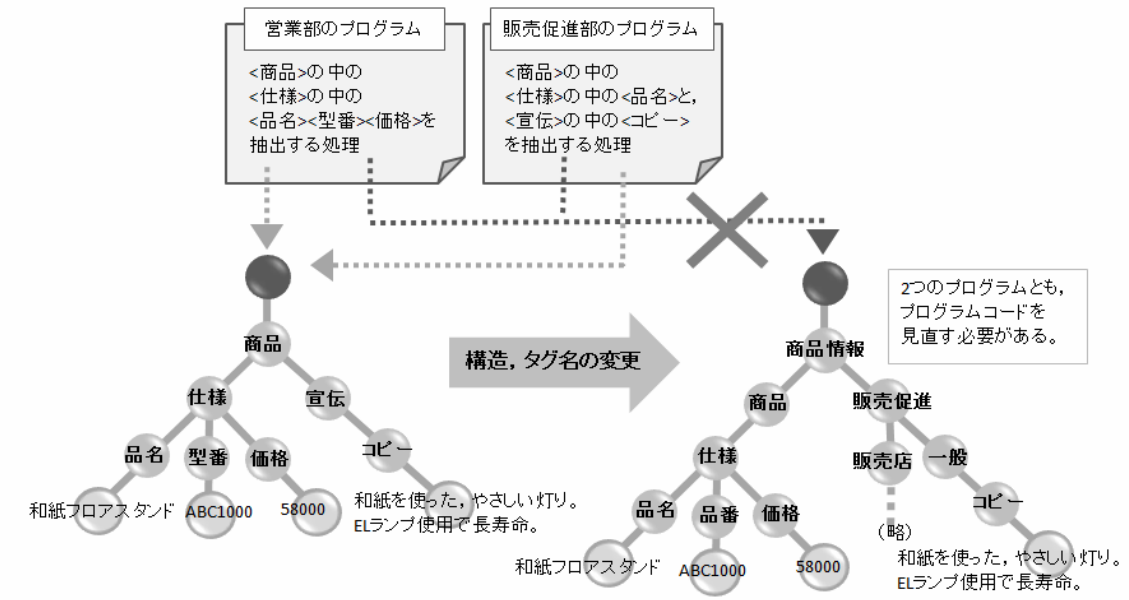
4.4 設計者にもとめられるスキル

4.4.1 最重要スキルは予測力と判断力

サステナビリティの問題～蓄積期間と範囲の想定～

XML データの用途や連携方法が多岐にわたり，長期間・広い範囲で利用されると，XML 文書の存在する環境は変化する。木構造は変化に対して柔軟であるから，末尾に枝葉を付けて成長するぶんには，大きな影響はない。だが，木が柔軟であるからといって，木のデータを処理するアプリケーションも，変更に柔軟に対応できるとは限らない。木の最下層にいたるまでの中途の階層で，構造やタグ名が変更されると，アプリケーションの側に影響が及ぶことがある。特に，複数のアプリケーションから同じ木を利用していると，その影響は深刻である（●図 4-16）。正常に動作しているプログラムコードの見直しは，リスクが大きい。

●図 4-16 利用中の木構造の中途階層の変更は，プログラムに影響を及ぼすことがある。



そのため，実務レベルの設計では，運用上避けられない変更を，できるだけ吸収できるような構造を考えておかなければならない。木の構造も，それを処理するアプリケーションのプログラムコードも，運用期間中の変更を免れる，サステナブルなものを目指すべきである。

それには，設計に着手する前に，対象データが利用される時間と空間の範囲を予測する必要がある。

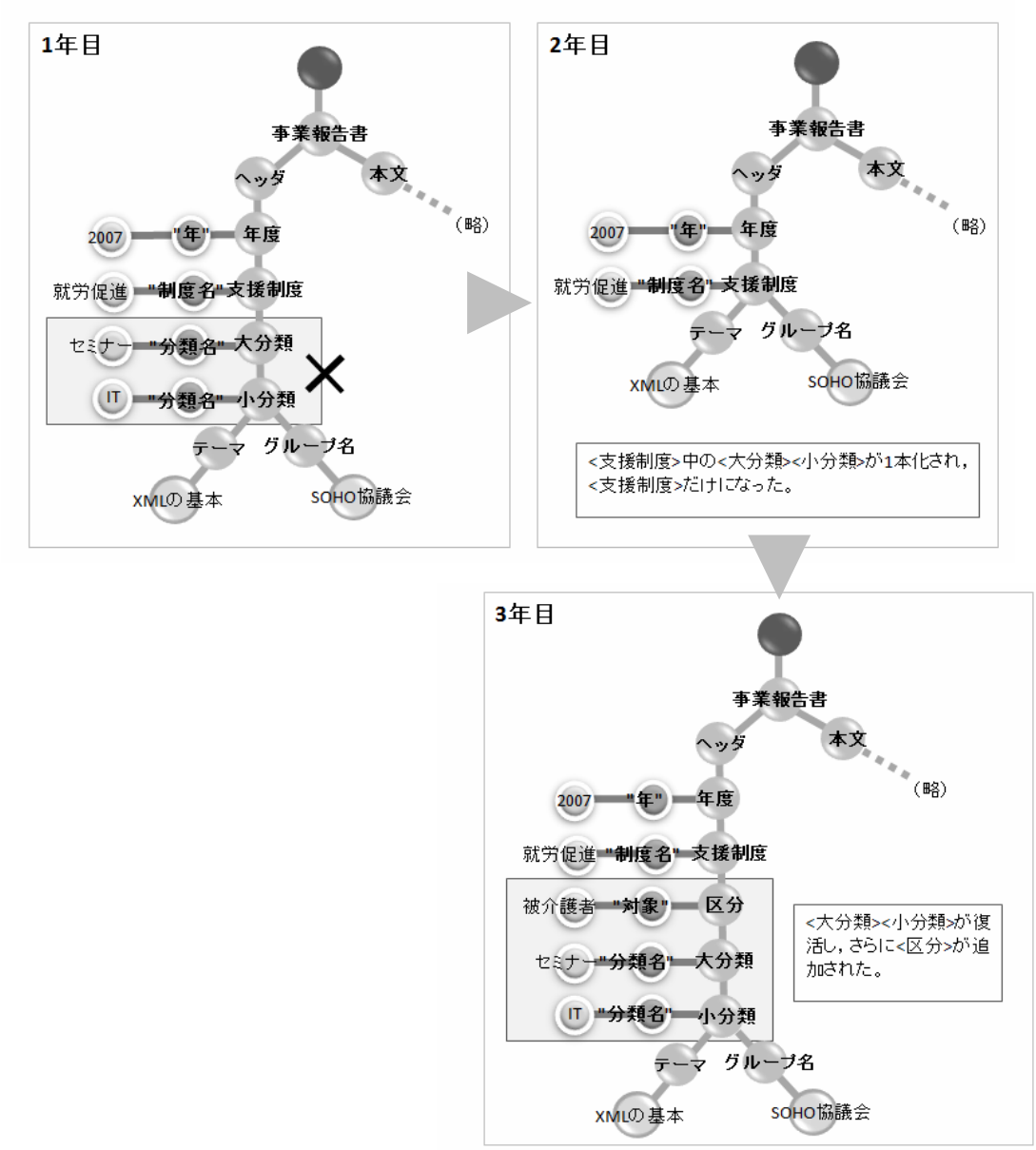
未来の任意の時点で，現在作成している XML 文書を参照したり，アーカイブのデータを追加して利用することが想定される場合は，「現在→未来」という一方通行ではなく，「未来→現在→過去」という方向からも，設計が破たんする恐れがないか検討してみるとよい。利用期間と範囲の予測に基づいて，最適の構造を判断しなければならない。

また，もし，運用期間後，XML 文書が廃棄されることなく任意のハードウェアの中に存在し続けるのであれば，考えられる限り，再利用に耐えうる構造を心がけておくべきである。

社会情勢の問題～時代の空気を読む～

XML 文書は，長期間利用されるほど，社会変化に遭遇する回数も多くなる。かならずといっていいほど遭遇するのは，政治経済の動きや，法律の制定や廃止といった社会システムの変化である。

●図 4-17 社会情勢が設計に影響を及ぼす例



比較的よくある例で説明しよう。ここでは，技術ベンチャー・グループが，数年間，公的機関の一部助成を得て，新素材の用途開発を行い，開発報告書を XML 文書化して提出するものとする。

開発が順調に進み，1 年目の報告書を提出した。ところが，2 年目，予算縮小のあおりを受けて，制度自体が縮小され，比較的上位の 3 階層が 1 本化された。3 年目，景気が回復して大判振る舞いとなり，助成対象事業者が増え，分類の細分化が必要になった。このように，技術仕様には無関係な社会システム上の変化が，階層の追加や削除，要素名の変更に影響を及ぼすことがある（●図 4-17）。そればかりか，1 本の木を複数に分割する必要性が生じたり，まったく異なる設計の木と結合する必要が生じることもある。公的な意味合いを持つ木には，海外事情や景気や世論，投票結果までもが間接的な影響を及ぼすことがある。民間の木であっても，予算や異動などの事情もあろう。

社会情勢の変化の予測は難しいが，予測しなければ，長期運用の困難な木を設計してしまいかねない。

哲学的問題～最小単位を定義する～

もうひとつ，全く技術には無関係に見えて，その実 XML 設計に重大な影を落とす事項を挙げておこう。

それは，ヒトに直結するデータである。このデータは，非常に社会問題に晒されやすい。企業の吸収合併や提携によるヒエラルキーの変化は所属先の情報に大いに影響を与えるし，家族の構造も（健康保険は未だ世帯単位ではあるが），フラットな社会に対応するかのように，家長制のピラミッドから個人単位へと変わりつつある。

そこで，個人単位の木を作るとなると，最小単位の定義という問題が発生する。一意な最小単位が「一人のヒト」ではなく，さらに細分化した定義の異なる「我」となれば，木の構造も変わる（●図 4-18）。

一寸考えてみてほしい。妊産婦の場合，本人が最小単位だろうか，胎児も最小単位のひとつだろうか？胎児を最小単位と捉えるならば，何週目からヒトとしての最小単位とするのが生物学的に，あるいは「社会的に」妥当だろうか？

脳死者は，（発する信号のすべてを他者や機器が受信することはできないにせよ）どの時点まで自身の存在を認識しており，「存在」たりえるのか？また，社会的に生きている「社会的存在」と定義付けられるのか（社会的存在＝存在，ではない）（※4）。移植される前の臓器の情報は，どの時点までがドナーに属するものなのか？

複数のパーソナリティを持つ者は，1 つのパーソナリティが最小単位なのか，脳の中の特定の部分が最小単位なのか？それとも 1 つの脳が最小単位なのか？

遺伝子情報は，個人の属性に過ぎないのか，個人の実在そのものなのか？

脳の特性が影響した社会的行為は個人の意志では制御不可能なものと考え，別の木の情報として記録するのが妥当であろうか？それとも，脳は個人の身体の一部であるからという理由で，個人の子要素して記録するのが妥当であろうか？

現在の医療情報は，基本的に，患者が最小単位である。だが，はたして今後も，その構造で情報を表現することはできるだろうか？

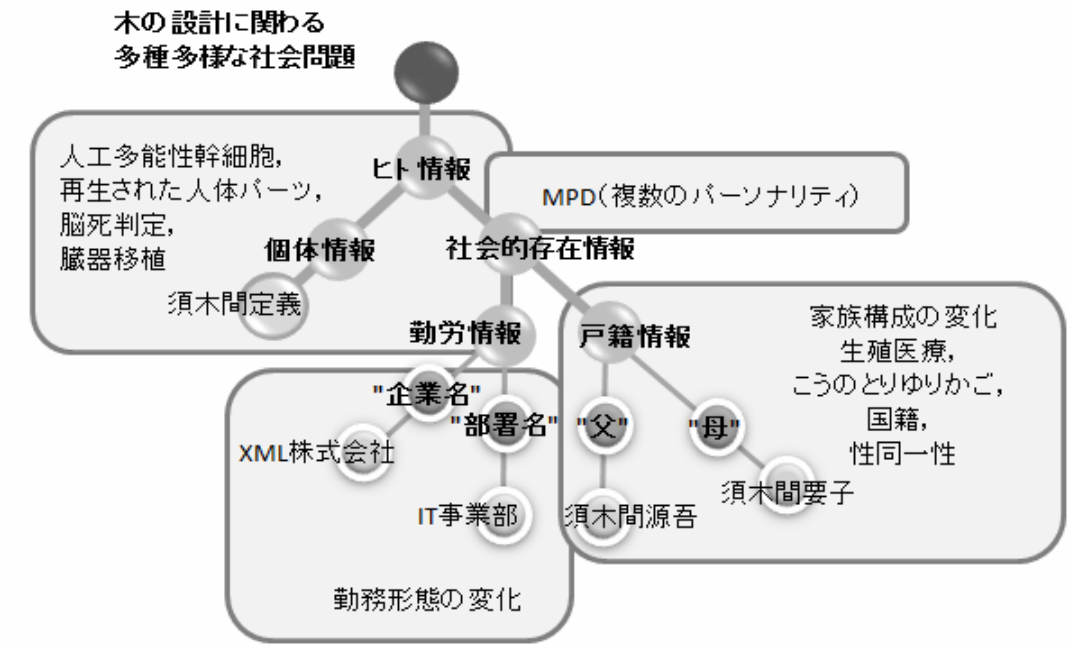
iPS 細胞，代理出産，脳死，臓器移植といった「一人のヒト」ノードを揺るがす問題は増え続ける一方である。社会的な妥当性と生物学的，医学的な妥当性は必ずしも一致しない。個体差もある。人それぞれの立場や哲学の問題もある。存在に対する根源的な問いかけを差し置いての，とりあえずの社会的解決を導く必要性のある議論に，確たる一つの正解などない。

とはいえ，医療と法律は，XML を先んじて採用した分野ではあるが，未開拓の分野でもある。電子カルテの IT 化は開業医の導入負担の問題などから，遅遅として進まないと聞く。また，MML が浸透しているわけでもない。RDB で管理しているケースもある。つまり，これからの業務である。

こういった個人情報を記録した XML 文書と，将来的に何らかの形で連携する木の設計を，皆さんが手がける，あるいは何らかの形で設計に関わる可能性は，決して低いとはいえない。

これから構築する木を，10 年先にも生き続けるものとするには 10 年先を，20 年先にも生き続けるものにするには 20 年先を予測して，構造を考えなければならないのは自明の理である。そして，XML が長期蓄積に適している仕様であるだけに，10 年，20 年，といった長期運用案件のニーズは少なくないのである。

●図 4-18 一人のヒトの最小単位が変われば，木の構造も変わらざるをえない。



※この木はあくまで例であって，実務では，"企業名"ではなく，変更が生じない企業コードを用いるべきである。なぜなら，企業や商店の屋号は，一地域内では重複しないが，他の地域では重複する可能性があるからである。

こういった，存在とは何かという問い，一意な存在と世界の境界，世界の成立のような問題は，技術者には無関係であって，哲学者や物理学者や生物学者に任せておけばよいと思われるかもしれないが，決してそんなことはない。

我々は，平生の生活の中で，自分と他者，自分と周りの世界，生者と死者を，明確に区別できているように勘違いしているが，実のところ，たしかなことなど何ひとつない。我々を構成する物質はいかにして存在することとなったのか？意識の座はどこなのか？情報機器は存在の発するすべての情報を取得できるわけではなく，我々は，自分自身のことさえ，何も分かってはいないし，共通の答えを持ち合わせているわけでもない。

しかしながら，根本的な問題について無知であっても，設計者は，頼まれれば，木構造を考えなければならない。一意なデータを必要とするシステムを構築しようとすれば，何を一意とするかという問題を避けて通ることができない。

30 年前に我の定義の細分化について語っても，笑い話にすらならなかったが，30 年後の今は問題となり始めている。ここに書いていることは荒唐無稽な話ではなく，確実に設計上の課題となる。XML 設計の本であるにも関わらず，第 1 章の冒頭を，存在と時空の話で始めたのは，そのような理由からである。

これからの設計者は，「一意な存在とは何か？」という問題から，決して逃れることはできない。

※4：本書は 2008 年秋に執筆されたが，その後，臓器移植法案が可決され，「社会的な存在」とは何かという問いに対する国家の答えが定まってしまったようである。

4.4.2 ヒトが設計することの意義

コンピュータとヒトの，予測・判断方法は異なる

本書で XML を学ぶきっかけを得て学習を重ね，XML 設計者となったとしても，3 年後には，かなりの設計業務がパターン化され，自動化されるようになるだろう。設計パターンの公開データベースができ，類似ケースの構造設計を省力化できるようになる。また，コンピュータに，新しいパターンを逆提案してもらうこともできるようになる。

だからといって，少なくともあと30年，（XMLに限らず）データ構造設計という仕事が「完全に」自動化されることはないといっていい。その間に，構造設計の学び方を学ぶ（メタ学習する）ボトムアップ型プログラムが次々発表されたとしても，である。なぜならば，コンピュータが，ヒトと同じ仕方で予測したり，予測に基づいて判断することは，不可能だからである。

現在のコンピュータは，「時間の流れ」を前提とした仕事しかできない。「既に与えられた」条件にもとづく一方向の計算はできるが，「未来に決定されている」条件にもとづく計算はできない。たとえていうならば，選ぼうとする道が落石で通行止めになっている可能性を，天候や地理的条件から計算することはできるが，上空に視点を移し，そこから行く手にある光景を俯瞰して選ぶべき道を判断する，という芸当はできない。現在稼働中のコンピュータが，未来の自分自身（コンピュータ）が計算することになっている木構造を，論理を積み重ねて推論するのではなく，論理に縁らない方法で察知することはできない。

決定されている条件とは，見知らぬ多数の人々の言動の積み重ねの末に選ばれている道である。コンピュータは，これを感知することができない。仮に，科学技術の進化によって，会議などの場の空気を作り出している任意の物質をセンシングできるようになったとしても，コンピュータはそのデータを基準とした判断しかできない。時代の空気を読み，社会システムの変化を，ヒトが感じるようには感じることはできない。

そして，コンピュータは，ヒトの脳を含む身体性に基づく予測ができない。ヒトの意識や倫理感が絡む問題を理解した上で，「一人の意識あるヒトの生命に関わる切実さ」をもって判断をくだすことはできない。

全体像を俯瞰するヒトの能力を生かそう

コンピュータは，この時空における自らの物理的ポジションの情報を取得したり記憶することはできるだろうが，自らの存在そのものを問うことができるだろうか？

また，物理的存在のあるポジションと，空間におけるポジションを，切り離して想像したり，知覚する能力を，獲得することができるだろうか？

そういった能力を持ち合わせなければ，いかに優れた処理能力を持つコンピュータであっても，ヒトのように，「概念（コト）」を取得することはできないのではないか？ヒトは「時間は存在しない」ことを想像し，実感し，理解することができる。だが，コンピュータにそれが可能だろうか？

時間のある世界で動くコンピュータが，概念の世界の情報を取得することができるとは，にわかには考え難い。少なくともここ 30 年で，コンピュータが時間を克服するとは考えられない。コンピュータにとっては，時間はたしかにあるものであり，未来の選択肢とは，時間の軸の先に転がっている「事象（モノ）」を表す情報であるだろう。

語弊はあるかもしれないが，概念をクラス，事象をインスタンスに例えると，ヒトが「クラスを知ればインスタンスを生成する」ことができるのに対し，コンピュータは「生成されたインスタンスしか知覚できない」。

進化したコンピュータは，芸術作品を創作することもできるだろうが，依然として，ヒトとコンピュータを隔てるものがある。時空におけるポジションの知覚，存在の認知，事象ではなく概念の取得，といったことについて，何かしらヒトとコンピュータには根本的な違いがあるのではないか。そして，その違いは，予測という作業に大きく関わっているように思われる。ヒトは，「時間の流れ」を前提とせず，全体像を感知することができる。コンピュータよりも，適切な選択肢に「気付く」可能性を秘めている。

その可能性を追求することが，設計の品質を高めるのではないだろうか。

こういった，いわゆる「考えても結論があるわけでなし，何の役に立つのか分からないようなこと」は，多忙な日常生活の中では，あまり意識しないかもしれない。だが，設計の仕事を突き詰めるならば，設計者は，既に決定されている木を観て，構造を考えた方がよいし，そうした姿勢を目指すべきであろう。

4.4.3 XML 設計とは何か？

XML 設計者の価値は，良くも悪くも，コンピュータとは異なる方法で予測を行い，なにがしかの判断をすることにある。XML 設計を行う上で最も重要なスキルは，理解の速度でも，記憶力でもない。ヒトにしかできない，予測と判断である（もちろん，筆者のように記憶力が悪いよりも，良い方がいいに決まっている！）。

適切な予測と判断に基づく木は，データのゴミを生み出さず，変化を吸収しやすい構造になる。XML 文書の蓄積利用期間が終了したとき，一切の問題が発生せず，目的を達成できた設計が，結果的に良い設計である。予測と判断という問題が絡む以上，木構造の設計に【良い設計はある】が，【唯一の正しい設計はない】。

XML の本質とは，端的に言うならば，「データに最もふさわしい意味と構造」である。
そして，XML 設計の実践とは，「的確な予測と判断に基づいて，その構造をデザインすること」である。

XML 設計者を目指すならば，技術仕様を極めることにのみ，達成感を持ってはならない。予測力と判断力を磨かなければ，コンピュータによる計算よりも不適切な構造を設計しかねないことを肝に銘じたうえで，実務に取り組むべきである。

設計に取り組む心がまえができたところで，次章では，ネイティブ XML 文書の基本的な設計方法について見ていこう。

Coffee Break「未来のリサイクル業」

未来のある時代。我々の後進の会話。

「20 世紀は，すばらしいデータ形式が花開いた時代だった。」
「ああ，そうだ！XML が登場し，そしてポスト XML として，あまたの仕様が現れた。」
「どれも，すばらしいものだった。だが，いまや，その姿かたちもない。」
「データは残っているのにな。」

「仕様策定者たちも，開発者たちも，無念だったろうな。処理不能なジャンク･データの爆発に敗れたのだ。」
「そう，彼らは，皆それなりの仕事をしたさ。でも，彼らの力だけではどうにもならないことがあった。設計者がまっとうな設計を行い，実装者が誠意をもってバグをつぶし，運用管理者が睡眠時間を削って目を凝らしていたとしても，データを発信する側の全員が，データを正確に分類し，意味付けするのでなければ，ゴミは指数的に増えるのだ！」
「そう，そして，世界の中心がなくなり，全員が発信源になった。動画，音楽，自分の部屋の一歩外に出れば，誰かに撮られている，そんな世界になった。」
「だから，オピニオン･リーダーたちは，誰もが正確に意味付けしたデータを発信すべきだと訴え始めた。だが，オピニオン･リーダーたちは，ひとつ大切なことを見落としていた。」
「ああ，彼らは，人の認知の差を，小さく見積もりすぎていた。誰でも，努力すれば，さほど大差なく，事実を正しく受容し，認識できると思っていたのだろうなあ。まぁ，誰でも自分や周りの人間を基準に，一般的ってことを考えてしまうけれども。オレだって，自分や仲間や家族を一般的だと思いがちだし，人のことなんて言える立場じゃないんだが。」

「あはは，オレだって，そうだ。それに，もうひとつ，大切なことが見過ごされてきたんじゃないかな。AI はずいぶん発達したが，人間の心は，まだコンピュータで置き換えることが出来ていない。コンピュータは，人が判断するようには判断できず，ふるえるような言葉の重なりをムダなデータとして処理している。詩のこころが徐々に失われ始めているような気がするよ。」

「うん，ひょっとしたら詩や哲学のいくらかは，数々の後天的環境リスクの偶発的な組み合わせに晒され，傷つき，放置された脳から始まるのかもしれないな。詩人の完全なクローンが，詩人になるとは限らない。リスクなく育てれば詩人にならず，リスクを与えすぎると繊細なだけに潰れてしまうかもしれない。だが，ヒトと違ってコンピュータは病老死苦を排除する。手加減を知らない。リスクに晒されれば，傷ついた CPU を元通りに修復しようとする。完全を目指す努力が，逆に調和を損ねる結果につながるなんて，先達は考えなかったのかもしれないなあ。」

「そうだな。もっとも，だからこそ，我々は，こんな仕事にありつけているのだが。さて，もう一仕事するか。このゴミの山から，使えそうなデータを見つけなくては。あと 100TB 探したら，休憩としよう。」

第 5 章　XML 設計の基本

本章では，XML 文書の基本的な設計方法を指南する。

5.1 設計の準備～企画・ 仕様調査～

5.1.1 設計の基本工程

「XML 設計」は，狭義には，データ･ライフサイクル中の一工程である。広義には，データ･ライフサイクル中の「企画」から「タグ付け」までを含む工程である（●図 5-1）。

「良い設計」を目指すには，XML1.0 仕様に則った「正しい設計」だけでは十分でない。前章で述べたように，運用期間中に構造の見直しが生じないよう，変化への耐性を高めなければならない。また，不要なノードの追加や文書の蓄積を制御し，データ爆発を防がなければならない。目指すべきは，「XML 文書の蓄積期間が終了したとき，一切の問題が発生せず，目的を達成できた設計」である。

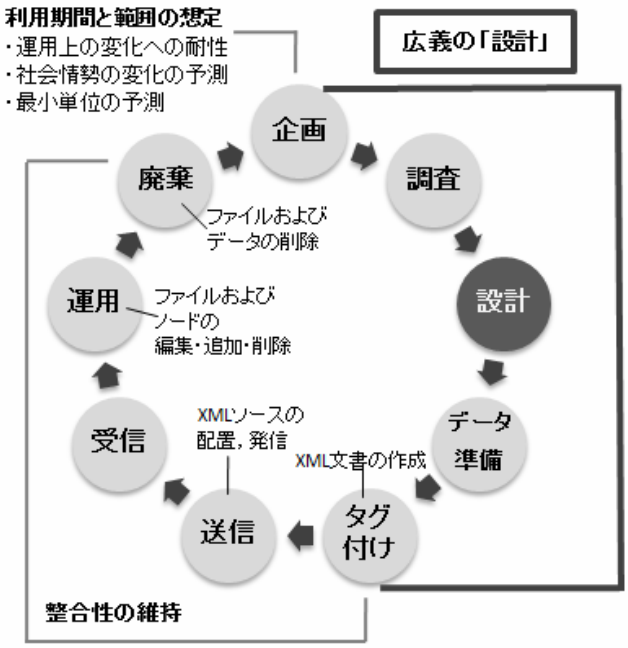
それには，XML データを「静的なテキスト」というよりもむしろ，「アプリケーションから利用される動的なデータ」として捉えて，設計にのぞむとよい。

では，具体的には，どのように作業を進めればよいのか。

●図 5-1 中の広義の設計における基本的な作業を，●表 5-1 に示す。これは，最も基本的なネイティブ XML 文書―――名前空間は使っていないか，使っていたとしてもデフォルト名前空間のみの，整形式であって妥当でなくてもよい XML 文書―――の設計工程である（**※1**）。

※1：本書では触れないが，実務では，予算･工期･人員の制約がある。XML 処理アプリケーション開発とメンテナンス，OS や開発フレームワークやミドルウェアの導入やバージョンアップ，ホスティングなどの費用も，工程に影響することがある。

●図 5-1 ネイティブ XML 文書の基本的な設計工程



●表 5-1 ●図 5-1 中の広義の設計における作業

企画	企画書作成
	処理アプリケーションの動作要件
	ミドルウエア導入の判断
	作成・処理のための仕様の決定
調査	設計用データの準備
	既存スキーマの調査
	設計のルール作り
	XML文書の作成方法決定
設計	木構造の設計
	文書内における整合性の確認
	文書ファイル群の構成の検討
	ユーザインタフェースの設定と適用
	複数ファイル間における整合性の確認
	タグ名の確定
	仕様書の作成
データ準備	実データの準備
タグ付け	XML文書ファイル生成用アプリケーションの開発
	XML文書ファイルの作成

5.1.2 設計の準備～企画～

企画書作成

●表 5-1 の通り，XML 設計の第一歩は，企画の立案である。決して，いきなりタグ付けを考えたり，XML ソースを記述してはならない。段取り八分が重要であり，企画段階を踏むことなく「良い設計」はできない。

企画作業は，案件が独自開発ではなく受託開発ならば，ヒアリングのうえで要件定義を行うことから始める。顧客業務の背景（社会的背景，文化的背景，経営・業務課題）を調査したうえで打ち合わせに臨み，要望を引き出して傾聴する。そして，「対象」「範囲」「目的」「目標」を具体化していくのである（●表 5-2）。

●表 5-2 ネイティブ XML 文書設計の「企画」工程

企画	
工程	検討・決定事項
企画書作成	利用対象（ユーザ層，スキル）
	運用範囲（時間，空間）
	XML化の目的（RDBと比較しての優位性，XML化による利点と問題点）
	期待効果（活用方法，安定性，拡張性，達成目標）
処理アプリケーションの動作要件	使用するXML仕様のバージョン
	アプリケーションの種類（ローカルアプリケーション，Webアプリケーション，Webサービス，モバイルアプリケーション，AIRやSilverlightなどによるRIA，ガジェット等）
	ハードウエアの動作要件（CPU，メモリ，OS，フレームワーク，それぞれのバージョン）
	ユーザ側の環境（パソコン，携帯端末，PDA等。OS，画面解像度，回線，ブラウザとそのバージョン，プラグインとそのバージョン）
	XML文書ファイルを蓄積する環境（サーバOS，フレームワーク）
ミドルウエア導入の判断	セキュリティ，耐障害性，排他制御，インデックス作成，処理対象文書のサイズなどから検討
作成・処理のための仕様の決定	XML仕様のバージョン
	XML周辺仕様とそのバージョン

初めて企画に取り組むのであれば，まずは 5W1H の確認から始めてみるとよい。すなわち，誰が(who)，どのようなデータを(what)，いつ(when)，どこで(where)，なぜ(why)，どのように(how)，利用するのかを理解するのである。このうち，「誰が(who)」については，とくに念入りに確認する。設計者から見てユーザといえば「顧客(アプリケーションの運営管理者)」だが，顧客から見れば，それはエンドユーザ(一般ユーザ)や，顧客の取引先(顧客が運営管理するアプリケーションを利用して宣伝や販売を行っている者)かもしれない。わずかな言葉の解釈の違いが，理解の違いを生むこともある。

処理アプリケーションの動作要件

XML設計と，XMLデータ処理アプリケーション開発は，セット業務であることが少なくない。実際，両者を別件として扱うよりも，アプリケーションによるデータの動きをイメージしながらXML設計を行う方が，破たんを防ぐことができる。文書設計のみを請けたのでない限り，設計前に，アプリケーションの要件を詰めておく方がよい。

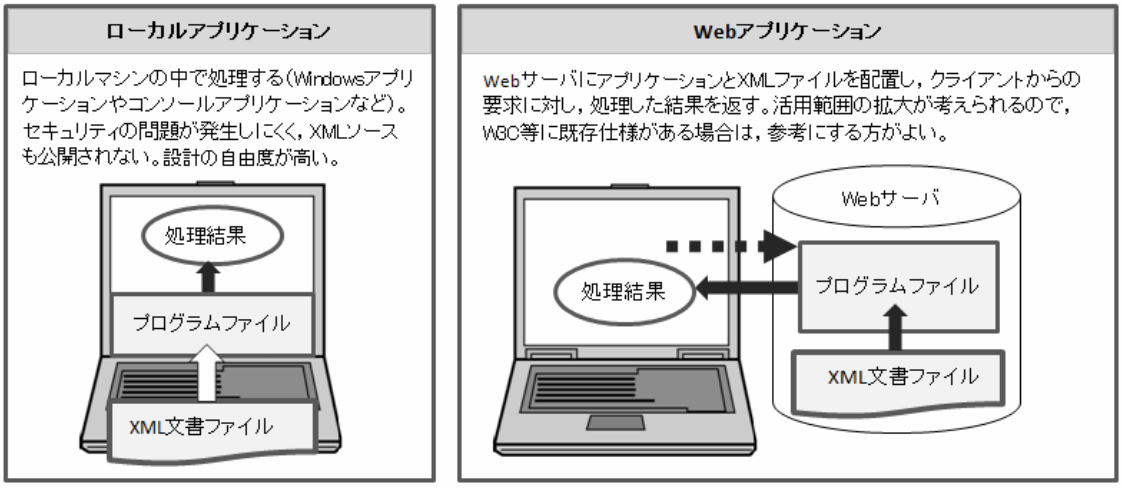
まず，実装希望機能から，アプリケーションの種類を検討する(●図 5-2)。

もし，既存のローカルアプリケーションと連携するWebアプリケーションを開発するのであれば，RDBのレガシーデータを利用して XML 文書を作成することになるかもしれない。すると，タグ名は RDB のフィールド名に準じることになるかもしれない。また，Web サービスを利用するのであれば，新規設計というよりもむしろ，既存 XML ファイルの再利用になる。

また，文字エンコードの認識，リソースの解放，データ処理のために用いる技術，といった，アプリケーションの開発フレームワークに依存する条件もある。

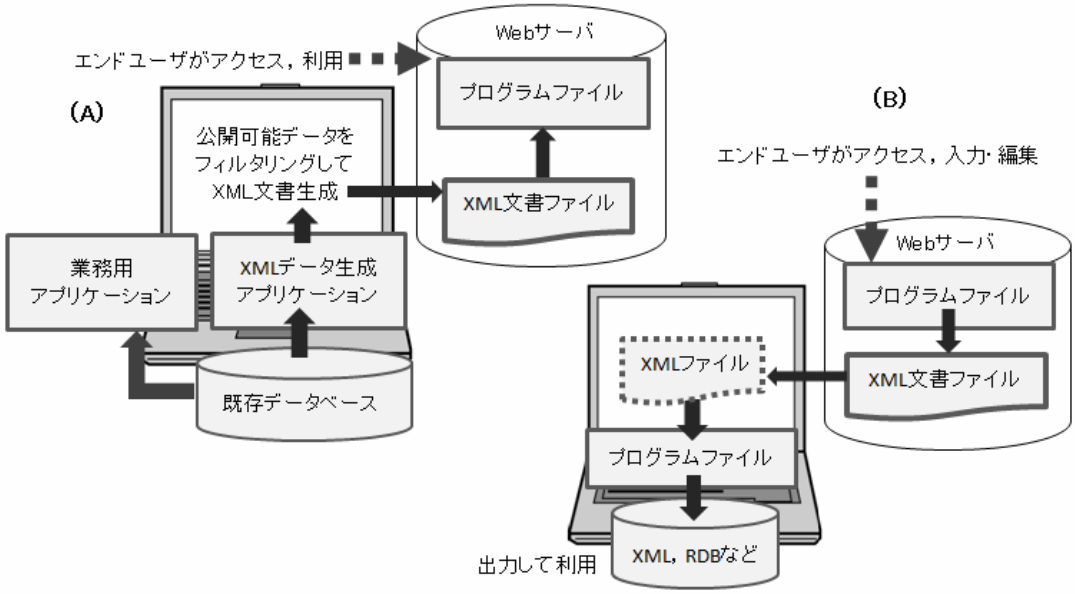
アプリケーションの機能が固まれば，ハードウェアの動作要件に合意を得る。それとともに，ユーザ側の環境も確認する。実行環境によっては，メモリの問題から，ノード数を抑えた方がよいこともある。もっとも，ハードウェアも進化するので，設計開始時点ではなく，運用開始時点での，データ量，見込まれるデータの増減を考慮したのでよい。

●図 5-2 XML データ処理アプリケーションの種類を検討する



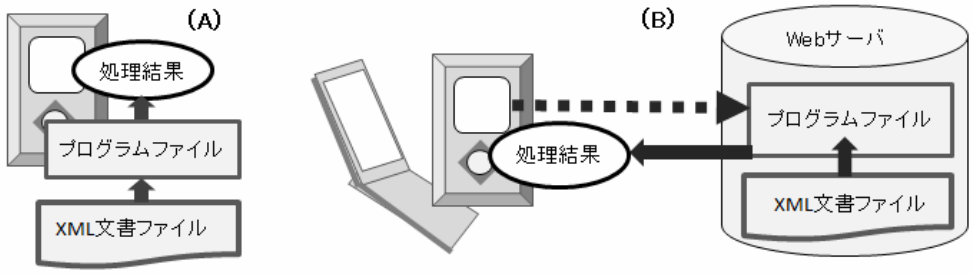
ローカルアプリケーションと連携するWebアプリケーション

(A) 既存ローカルアプリケーションで利用中の（主に）RDBのデータから，公開可能なデータを抽出してXML文書ファイルを作成する。タグ名は既存のフィールド名に準じた方がよい。
(B) Webアプリケーションから入力されたデータをもとにXML文書ファイルを作成し，ローカルアプリケーションで利用する場合は，最低限でも開発フレームワークの検証コントロールなどを利用したデータの検証が必要である。設計時には，データについても定義しておく（後述，★表5-6参照）。



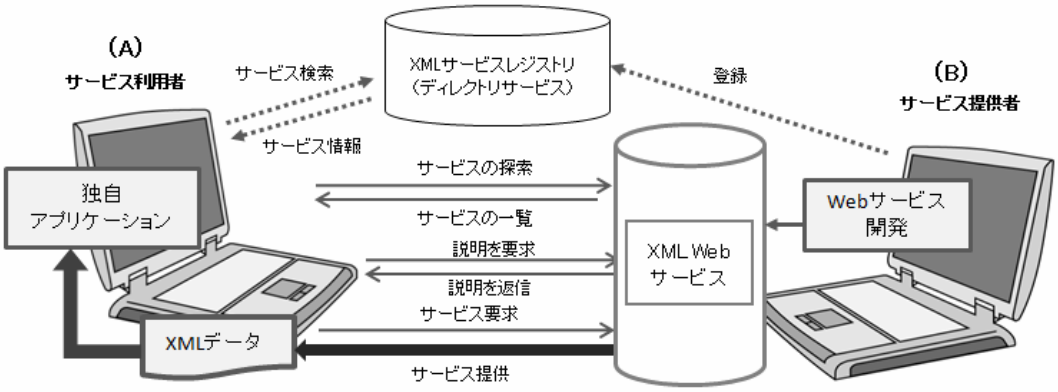
モバイルアプリケーション（PDA，携帯電話）

(A) 携帯端末内に配置したアプリケーションでXMLデータを
処理する場合は，携帯端末側のメモリの制約から，処理対象ノード数と階層の目安を設けて，その範囲内で設計する。
(B) ケータイ用Webアプリケーションで処理するXML文書の設計は，パソコン用Webアプリケーションの設計に準じる。表示スタイルの制約はXSLTスタイルシートの切り替えで対応すればよい。対応機種によっては 文字コードの問題に注意する必要がある。



Webサービス

(A) Webサービスで返されたデータを利用する場合や，複数のXML Webサービスを利用したアプリケーションを開発する場合は，返されたXML文書の設計を踏襲するかカスタマイズして利用する。
(B) Webサービスの提供者側となる場合は，Webアプリケーションの設計に準じる。タグ名は 英字の方がよい。



現在では，Microsoft Silverlight など RIA (Rich Intenet Application) や，SaaS (Software as a Service) も検討する必要があるだろう。Silverlight では，LINQ to XML を用いて XML 文書ファイルを処理することができる。

ミドルウェア導入の判断

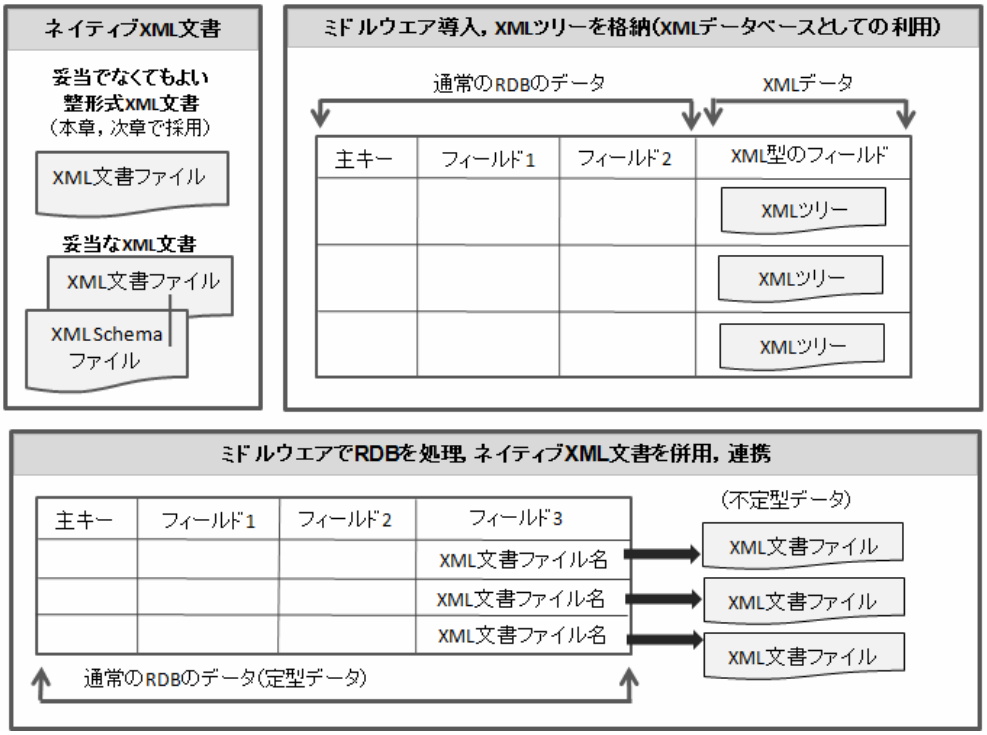
動作要件が固まれば，XML 文書ファイルを蓄積するハードウェアの環境も明らかになる。この段階で，ミドルウェア (XML データベース) の導入必要性について検討し，RDB，ネイティブ XML，XML データベースの採用方法を決定するのである。もちろん，RDB とネイティブ XML を併用する方法もある (●図 5-3)。

ネイティブ XML は，各種アプリケーションの設定ファイルとして，インターネット上で流通するデータとして，ローカルアプリケーション固有の文書保存形式として，広く用いられており，文書の共用や再利，連携をはかりやすい。また，単なるテキストファイルであるから，ハードウェアやフレームワーク，ミドルウェアなどの進化に伴う影響をダイレクトに受けることがなく，初期投資や維持費を抑えられるという実務上の利点がある。

しかしながら，セキュリティ，耐障害性，排他制御 (※2) の面からは，妥当な XML 文書を設計して，ミドルウェアに格納する方がよい。あるいは，そういった問題のつきまとうデータは RDB とし，問題のないデータのみ XML 化して併用するとよい (第 3 章●図 3-2 下図を参照)。

ミドルウェアを採用するとしても，ミドルウェアの XML 列に格納する XML インスタンスの設計は不可欠である。そのため，当然のことながら，ネイティブ XML 文書設計の知識や経験は，必要となる。

●図 5-3 ミドルウェアの導入について検討する



※2：いまA社の材料の在庫データが，n個であるとする。業務部員が，在庫n個を確認して，100個を購入し，在庫データに100個を追加する。ところが，その間に，制作部員が，在庫 n 個を確認して，10 個を使い，データを更新する。すると，在庫は(n-10)となる。ところが，その後で，業務部員の追加した個数が上書きされると，在庫は，(n＋100)個となってしまう。そこで，業務部員が，在庫 n 個を確認して 100 個を追加するという一連の手続き（トランザクション）が完了するまで，制作部員の更新を排他制御する。これにより，正しい在庫数である(n＋90)個となる。

作成・処理のための仕様の決定

XMLデータ処理の動作環境が明らかになったところで，採用するXML仕様と，XML周辺仕様およびそのバージョンを決定しておく。動作環境によって仕様のサポート状況が異なるので，処理アプリケーションの要件とミドルウェア導入の可否について決定した後，検討するとよい。

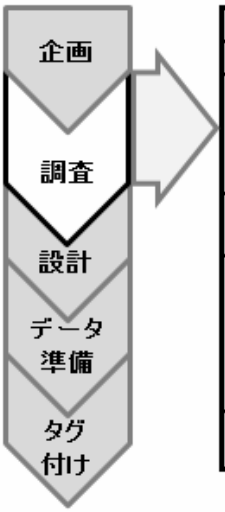
5.1.3 設計の準備～調査～

設計用データの準備

企画のアウトラインが見えてきた段階で，設計前の仕様調査に着手する（●表5-3）。

そのためには，まず，設計用のサンプル・データを準備しよう。もし，実データが存在しないならば，ダミーデータを作成しよう。可能な限り，運用期間中の追加分も見込んでおきたい。不十分なデータに基づいて作業を進めると，「情報の流れ」が寸断されたり，後からのデータの追加により「変化への耐性」に問題の生じる恐れがある。XML 設計の第一の関門は，このデータの準備であるといっても過言ではない。

●表5-3 ネイティブXML文書設計の「仕様調査」工程



調査	
工程	作業
設計用データの準備	サンプル・データ準備，なければダミーデータを作成
	データの内容，および更新方法，最終的な予想ノード数，データの機密性の確認
	データの再利用の可能性，利用頻度の確認
既存スキーマの調査	W3C仕様の調査，各業界団体の仕様の調査
	既存スキーマの利用，カスタマイズ，新規定義の決定
設計のルール作り	検証の必要性の有無，検証方法を決定
	名前空間URIと接頭辞の決定
	文字コードと名前付けルールを決定
	ルート要素名を決定
	暫定的なメタデータ名を仮定義
XML文書の作成方法決定	レガシーXMLデータの作成方法を決定

サンプル・データは，XML設計を依頼した”XMLについての知識を持たない”顧客から提供されることが少なくない。だからこそ，XML 設計者は，不足しているデータに気付く程度の，顧客の業種に関する業務知識を持つことが望ましい。

顧客からデータを収集するには，言葉を引き出して多様なケースを想定し，問題点を洗い出さなければならない。XML技術には無関係であるが，傾聴の技術が設計者には必須である。

顧客側担当者が技術者ならば，言語，文字コード，外字や異体字の有無，数式，地図，図面，画像（イラスト，写真，フィルムなど）といったデータの内容，および更新方法，最終的な予想ノード数，データの機密性について確認しておく。既存データがあれば，データ形式，現在の管理方法，データの再利用の可能性，利用頻度についても質問しておく。参加団体で共用，取引先や支社などで利用，他部署で利用，他目的で利用するかどうかといったことは，それが社外の設計者であれば知りえないこともある。

既存スキーマの調査

データの内容が見えてきたところで，利用可能な既存スキーマがあるかどうかを調査する。まず，W3C仕様を調査し，次に，各業界団体の仕様を調査する（**※3**）。それらはXML Schemaではなく，DTDや，Relaxで定義されていることもある。また，ツリーを図化したものが公開されていることもある。

既存スキーマは，必ずしも検証用として用いるために，調査するのではない。妥当ではなくてもよい整形式XML文書の設計においても，構造やタグ名を検討するための有力な資料となる。既に業界標準仕様があるならば，全部あるいは一部を利用する方が，新たに構造を定義するよりも，データ交換やデータ共有，XML 文書蓄積による効果を見込むことができる。また，業界用語をタグ名に置き換える際の検討作業が軽減される。

もし，設計用サンプル・データの中に，既存スキーマでは定義されていないデータがあるならば，既存スキーマの共通部分のみを利用し，不足分は検討すればよいのである。

新規設計ではなく，リメイク案件の場合は，既存スキーマが存在することがある。顧客側がスキーマファイルなどの構造設計仕様書を持ち合わせていないケースでは，XML 文書から木構造を推論する必要が生じる。設計仕様書が存在し，設計を修正する必要がある場合は，既に蓄積されている XML 文書を最大限活用できるよう，構造やタグ名を可能な限り生かす方向で考えたいものである。

※3：一例として，ニュース記事であれば，社団法人日本新聞協会によって「NewsML」が，Web 上のリソースであれば The Dublin Core Metadata Initiative によって「Dublin Core」が 公開されている。

設計のルール作り

既存スキーマの調査が済んだら，設計のルールを決めていく。

まず，検証の必要性の有無と，検証方法を決定する（第3章「XML Schema 文書作成および検証の必要度」参照）。

整形式文書であるかどうかを検証するだけであれば，ブラウザ搭載の XML プロセッサや Mifes for Windowsのようなエディタを用いれば十分に可能である（第3章「●図3-12」参照）。フォームから入力されたデータをもとに XML 文書を生成する場合は，開発フレームワークの検証機能を利用して，入力データを検証することもできる。

次に，XML 文書の利用範囲が社内の一部署にとどまらない場合は，名前空間 URI と接頭辞を決定しておく。

さらに，文字コードと名前付けルールを決定する（●表 5-4）。これは，プロジェクト名やプログラムファイル名の表記と統一するのが無難であろう。比較的多いのは Pascal 形式であるが，XHTML の Web ページと連携させたり，簡単なタグ名ばかりであれば全て小文字にすることもある。

●表 5-4 名前付けルール

ルール	説明	例
Pascal形式	3文字以上の識別子について，1文字目と，それに続く各単語の最初の文字を大文字にする。 ただし，続く単語がない場合は，1文字目のみ大文字とする。例：<Items>	<OrderDate >
Camel形式	1文字目は小文字で，それに続く各単語の最初の文字を大文字にする。	<orderDate>
大文字	2文字以内のものについて，すべて大文字にする。	<NO>
XHTMLに合わせる	すべて小文字にする。	<title>

タグ名をかな漢字タグと英字タグのどちらにするかも，この段階で決定しておく。

かな漢字を使った方がよいケースは，次の通りである。

1 つは，入門者が XML 設計を行うケースである。英字タグ，たとえば，name や no といったタグ名を用いると，プログラムの変数や，XPath や XSLT の式や関数とタグを混同する恐れがあるからである（本書で日本語タグを採用しているのには，同様の意図もある）。次に，顧客自ら，XML データを利用するケースである。催事の参加申込者やアンケートの集計と情報活用など，顧客が XML データを確認したい場合は，かな漢字の方が分かりやすく利用してもらいやすい。

英字タグを使った方がよいケースは，次の通りである。

1 つは，海外拠点があったり，海外との取引が考えられる場合。たとえば，海外向けの Web サイトや輸出入製品のマニュアルなどである。また，日本語以外のデータを扱う可能性がある場合。たとえば，専門分野の研究論文や，科学や医学など専門用語をタグ名とする必要がある場合も，英字タグの方がよいだろう。英語版のタグ付けツールで XML 文書を作成する場合も，英字タグを採用する。

もっとも，それ以前に，かな漢字にする必然性が「全く」ないのであれば，XML が世界標準の仕様であるという意味で，英字の方が望ましいといえるだろう。

ここで，今定めた名前付けルールに従って，文書の意味を的確に表すルート要素名を決定しておこう。そして，設計用サンプル・データを，性質とレベルを見極めて大まかに分類し，各データに対応する暫定的なタグ名を付けておく。この暫定的なタグ名を用いて，以降の設計作業を進めていく。

XML 文書の作成方法の決定

設計のルールができたら，レガシーXML データの作成方法を決めておく。作成方法は，主に，次の 6 つである。。

(1) テキストエディタで作成する。

(2) 既存の RDB のデータをプログラムによってシリアライズする。あるいは既存 XML データの構造を変換する。

(3) Microsoft Office のような XML 対応ビジネスソフトで作成されたデータを，XML 形式でエクスポート

して，プログラムから処理しやすい構造に変換する。
(4) Microsoft InfoPath のようなタグ付けソフトウェアを使って作成する。
(5) フォームから入力されたデータをもとに，プログラムによって XML 文書を生成する。
(6) Web サービスや Web アプリケーションで提供されている XML データを流用する。

このうち，(3)の，Microsoft Office のようなビジネスソフトからレガシーXML データを作成するケースは少なくない。ここでいうレガシーXML データとは，前の第 4 章で述べたところの「ワンソース」に該当するものである。

以上で，設計作業のための準備が整った。ここで，狭義の設計の実務に移る前に，人的資源の問題について触れておく。

設計者が XML 文書作成までをあ担当するケースは多くはない。設計者は，オペレータに作成作業を依頼するにあたり，次の 2 つに配慮すべきである。

1 つは，オペレータに XML 化の目的とメタデータの意味を伝えておくことである。専門用語や業界用語をタグ名として用い，その仕様書をオペレータに渡しただけでは，正確なタグ付けはできない。XML 文書作成は単なる文字入力ではない。

もう一つは，オペレータの職務経験を見極めたうえで依頼することである。たとえば，HTML コーディングをできる Web デザイナなら，XML 文書作成を一任しても問題はないかといえば，そうではない。ブラウザの解釈が寛大だった古いバージョンの HTML に慣れている者は，属性値のダブルクォートを省略したり，大文字小文字を混同して用いるかもしれない。

いくら設計能力に秀でた者であっても，他の作業担当者との連携を図らなければ，良い XML 文書以前に，正しい XML 文書を作成することはできないだろう。XML データ処理アプリケーションの動作確認時にエラーが生じ，長時間プログラムコードをチェックした挙句，原因は XML データの作成ミスであった，ということが起こりうる。構造の不備やデータ型の定義ミスは比較的発見しやすいが，スペルミスやタグの閉じ忘れなどの発見は難しい。ミスは，それが単純であればあるほど気付きにくい。データのコピー＆ペーストのミスにより，誤った検索結果が返されることもある。XML データの不備に最も影響を及ぼすのは，XML 技術自体には無関係な，ヒトの職務能力の問題である。

企画内容の再確認

さて，次節の設計の実作業に移る前に，企画内容を再確認しておこう。受託開発であれば，顧客と共に再確認することを強くお勧めしておく。

なぜなら，XML 化の成功を妨げる最大の要因は（これは XML 化に限ったことではないが），設計着手後に生じる「あれもこれも症候群」だからである。これは筆者の造語だが，あの処理も実装したい，このデータも追加したい，といった要望を，発信し続ける症状である。「あれもこれも症候群」は，発想力の裏返しであり，提案力と等価ではない。提案力とは，多数のわき出たアイデアの中から，成功につながるものを予測して絞り込み，それ以外を捨て去る判断力と決断力のことである。筆者のささやかな経験からいえば，顧客が企画の意義を理解しているプロジェクトは成功し，企画を軽視するプロジェクトは十中八九頓挫する。

顧客ではなく，設計者が「あれもこれも症候群」に罹ることもある。すると，一元的なスキーマ定義に固執するという症状を見せる。

XML 化に複数の目的があり，別構造の XML 文書が必要となる場合は，オリジナルの XML 文書から構造の異なる XML 文書を生成して使い分ける方が現実的である。ところが，この症候群に罹ると，多数の目的があるにもかかわらず，1 個の XML 文書で完結させようとする。1 本の木にすべてを詰め込むと，構造の見通しは悪化し，データが何度も重複する。それは，多次元を無理やり二次元にマッピングするようなものであって，人間を紙の上にぺしゃんこに貼り付けたら身動きがとれなくなってしまうように，データの身動きもとれなくなるのである。

設計者は，シンプルな設計を心がけるべきだが，シンプルな設計とは，必ずしも 1 個の XML 文書で済ませることではない。

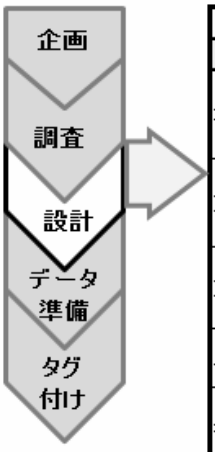
5.2 木構造の設計

5.2.1 ボトムアップ型設計

ボトムアップ型設計の進め方

さていよいよ，狭義の設計，つまり設計の実作業に移ろう。基本工程は，●表 5-5 の通りである。まず，本節では，単一の文書ファイル―――すべてのデータを 1 個の XML 文書ファイルに記録したもの―――の設計について解説する。

●表 5-5 設計工程の各作業



設計	
工程	作業内容
木構造の設計	ボトムアップ型で構造を検討
	ヒトによる理解とコンピュータによる計算のバランス
	技術以外の視点による構造の再確認
文書内における整合性の確認	ノードの編集・追加・削除と整合性の確認
	不整合を抑止する処理のルール作り
	ノード間の関係と処理実行のタイミングの確認
文書ファイル群の構成の検討	(複数ファイル構成を検討する場合)データファイルの構造
	(複数ファイル構成を検討する場合)ファイルの構成。管理ファイル使用の有無と，データファイルとの関係
ユーザインタフェースの設定と適用	設定ファイル使用の有無
	設定ファイルの用い方
複数ファイル間における整合性の確認	ファイルの変更，生成・追加，削除に伴う整合性の確認
	CRUDのタイミングの確認
	単一ファイルか複数文書ファイルか，ファイル構成を決定
タグ名の確定	タグ名の重複可能性の確認
	要素内容および属性値のルール作り
仕様書の作成	データ定義仕様書，サンプルXML文書ファイル，XML文書ファイルの作成方法，データ更新方法，設計までの経緯と決定理由

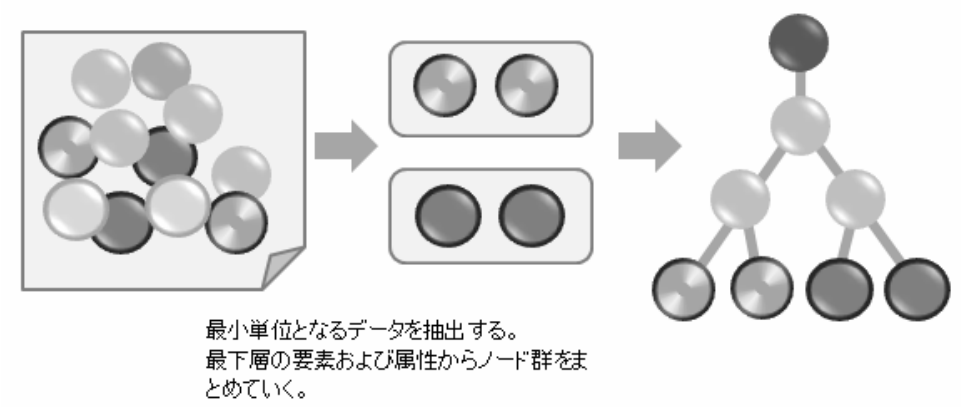
前章で述べたように，XML 文書の設計では，廃棄しやすい木構造を心がけなければならない。それには，トップダウンよりはむしろ，リーフノードからボトムアップで構造を考える方がよい。

手順としては，まず，設計用サンプル・データの中で，最小単位となるデータを 1 個特定する。次に，そのデータと同じレベルで同じ性質のデータを抽出する。そして，最下層の要素あるいは属性から，ル

ートへ遡るように，同じレベルであると思われるノード群をまとめていく。この作業を，ルート要素ノードにいたるまで，繰り返すのである。

ボトムアップでノード群をまとめるには，情報の流れをイメージして行う。第 1 章で述べたように，データの中に「過去→未来」「外→内」「高→低」のような変化を探し，一連のノード群を流れに沿って並べ，「シーケンス」を意識ししながら作業を進めていくのである（※4）（●図 5-4）。

●図 5-4 ボトムアップで構造を考える



この一連の作業は，暫定的な要素名や属性名で行う。正式なタグ名は，構造が確定した後で確定したのでかまわない。また，要素では出現順序が問われるが，属性では問われない。XML データ処理上の特別な事情がない限り，属性の順序は，構造と属性名が確定した後で検討したのでよい（※5）。

※4：地域別や職業別といった分類や，番号順・五十音・アルファベット順は，第 1 章で述べたように「商品情報の，カテゴリの中に複数の商品があり，各商品の中に型番や色やサイズや単価がある」といった，「外→内のシーケンス」として考えるとよい。

※5：顧客，XML 設計者，XML データの入力オペレータ，アプリケーション開発者，といった複数の担当者が協力して構造を検討する場合は，（データの量にもよるが）準備したデータをテキスト化して厚手の紙に印刷し，最小単位ごとにデータを（文字通りカッターで）切り分けて，会議テーブルの上に並べて考えていくとよい。原始的な方法ではあるが，ボードに手描きしたり，ツールを使って画面上で修正を繰り返すよりも，全体を見渡せるので効率がよい。IT の仕事は全て情報機器を使って行うべきだという考えにとらわれない方がよい。

ボトムアップ型設計の利点

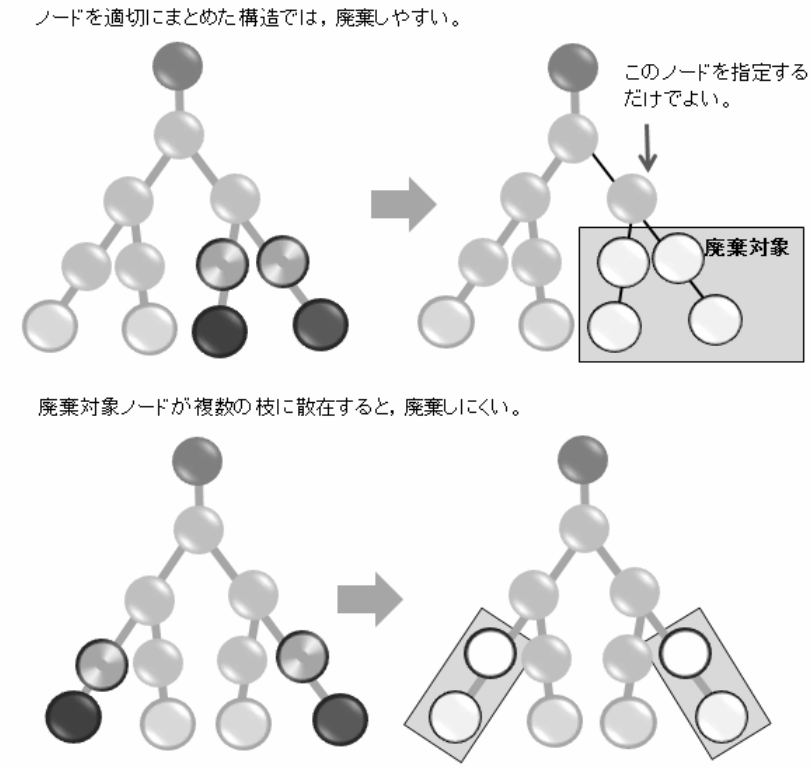
ボトムアップ型の設計方法には，2 つの利点がある。

1 つは，最下層からまとめることにより，廃棄対象ノードが複数の枝に散在することを，設計の早期段階で発見することができることである（●図 5-5）。これにより，n 個のデータを削除するために，n 個のデータが付属する n 本の枝を操作しなければならない可能性が低くなる。すなわち，廃棄の際には，n 個のデータを子孫として持つ最低限の本数の枝を，操作すればよい可能性が高まる。

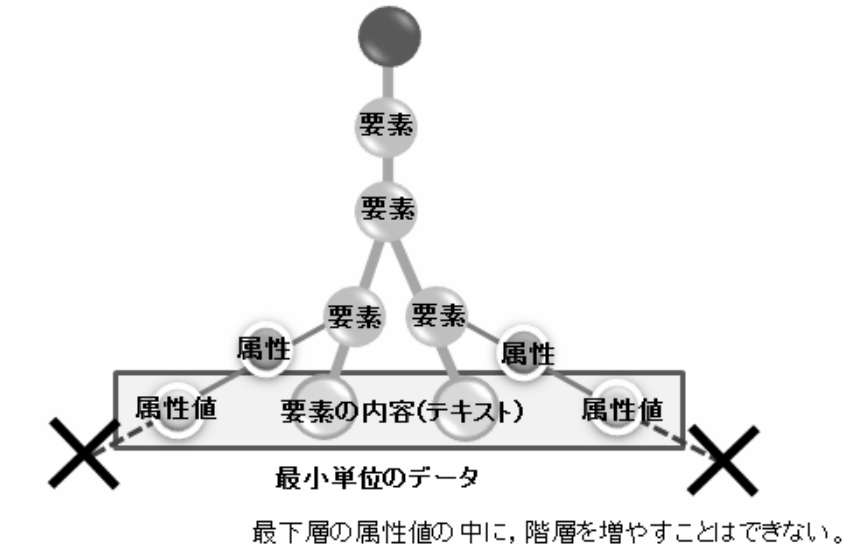
もう 1 つは，最小単位のデータを，さらに細分化するような事態を回避しやすいということである。

最小単位のデータは，木の最下層の要素の内容か，木の最下層の要素に従属する属性がある場合は，その値となるテキストである（●図 5-6）。もし，運用期間中に細分化するデータが予測されるならば，要素として設計しておくとよい。

●図 5-5 廃棄しやすい設計は，ボトムアップ型で行う



●図 5-6 最小単位のデータは最下層の要素の内容か，従属する属性値に割り当てられる。



ただし，すべてのケースでボトムアップ型を採用「しなければならない」わけではない。構造の見通しが立ち，最大階層が自ずと決まるケースについては，トップダウン型で設計すればよい。

たとえば営業予定表などで，年・月・日・時間・分までのタグは必要であるが，秒やミリセコンドまでは不要であることが自明，といったケースである。また，長期間印刷媒体や PDF として制作しているマニュアルなどの文書類では，既に構造が確定しているものもある。もし，文書本体が「管理情報」と「本文」から成っており，「本文」内の構造が「章」「節」「項」で，最下層が「項」内の「リスト」や「リスト項目」であるといった構成が定着しているのであれば，これを XML 化したところで，<リスト項目>より深い階層が生じることはないであろう。

要素と属性の選択基準

最小単位を確定するには，要素と属性の使い分け方について，理解しておかなければならない。たとえば，次のようなケースについて考えてみよう。●リスト 5-1 は，要素のみで表現された木の一部分である。一方，属性を用いると，●リスト 5-2 のようになる。

●リスト 5-1 要素のみで表現された木

```
<?xml version="1.0"?>
<講師>
  <所属>XML 株式会社</所属>
  <氏名>須木間　定義</氏名><氏名>江楠　絵夢</氏名>
</講師>
```

●リスト 5-2 属性を使った木。

```
<?xml version="1.0"?>
<講師 所属="XML 株式会社">
  <氏名>須木間　定義</氏名><氏名>江楠　絵夢</氏名>
</講師>
```

ここで，●リスト 5-2 の属性を用いた構造を採用したとしよう。

では，もし，"所属"属性に対して，部署や，ルビの追加が生じたら，どうすればよいだろうか。また，もし，XML 文書の活用時にソート処理を行うことになっていたら，どのようにルビを扱えばよいだろうか。アルファベット順にソートする場合はアルファベット，五十音順にソートする場合は「かな」をデータとすればよいが，両方が必要であれば，"ruby"と"ふりがな"は，どのように定義するのか。"所属"は属性であるから，これ以上細分化することができない。

もし，こういった事態を予測することが難しいのであれば，要素のみの設計の方が安全なのである。

要素の内容の分割

最小単位の検討において，RDB ならずとも XML 設計においても発生する，ごく一般的な問題がある。<氏名>を<姓><名>に分割するかどうか，<住所>を<都道府県名><市区町村><町名番地>とするか，あるいは<住所 1><住所 2><ビル名>とするか，といったことである。携帯番号，シリアル番号など，何らかの区切りが存在するデータを扱うなら，検討が必要である。

すでに RDB のデータが存在し，これを利用して XML 文書を作成するのであれば，RDB に合わせることも考えられるが，1 からの設計では，複数に分割する方が無難である。子要素あるいは別個の属性として列記しておく方が，一連のデータを一要素の内容としておき，プログラムや XSLT スタイルシートによって必要部分を抽出するよりも，処理は容易である。

もっとも，顧客側に海外拠点開設が予定されており，海外の電話番号，住所，人名データといった連絡先データを記録することになるであろう場合は，顧客側と相談しながらケース・バイ・ケースで決めるしかないだろう。

5.2.2 ヒトによる理解とコンピュータによる計算のバランス

次に，アプリケーションによって XML データを処理する場合の，処理効率について検討する。

●リスト 5-3 に，ひとつの例を示す。これは，コンピュータには処理しやすい，階層は浅く，短いタグ名を付けた文書である（説明のために，極端な要素名や属性名としている。また，表やリストの中は細分化していない）。しかしながら，ヒトが，このソースから，木構造を一瞬で思い描くことができるだろうか。（日本語のわかる）ヒトにとっては，●リスト 5-4 の方が理解しやすいに違いない。●リスト 5-3 を見て，即座に，それが●リスト 5-4 と同じ目的で設計された構造であるとは，気付きにくいのではないだろうか。

●リスト 5-3　コンピュータによる計算のみを重視した構造。

```
<?xml version="1.0"?>
<root>
  <info date="2009-01-01" uri="report/2009/xml01.html" title="XML の現状と課題" c="XML 株式会社" n="須木間定義" />
  <data chap="1" sec="1" div="1">
    <p>1 段落目</p>
    <ul>項目</ul>
    <table>表 1</table>
    <img>図 1</img>
  <data>
</root>
```

●リスト 5-4　●リスト 5-3 をヒトが理解しやすい形にした構造。

```
<?xml version="1.0"?>
<公開報告書>
  <ヘッダ>
    <標題>XML の現状と課題</標題>
    <公開日>2009 年 1 月 1 日</公開日>
    <報告書 URI>report/2009/xml01.html</報告書 URI>
    <執筆者>
      <勤務先>XML 株式会社</勤務先>
      <氏名>須木間定義</氏名>
    </執筆者>
  </ヘッダ>
  <本文>
    <章>
      <見出し>第 1 章</見出し>
      <節>
        <見出し>第 1 節</見出し>
        <項>
```

```
            <見出し>第 1 項</見出し>
            <段落>1 段落目</段落>
            <箇条書き>項目</箇条書き>
            <表>表 1</表>
            <図>図 1</図>
          </項>
        </節>
      </章>
    </本文>
</公開報告書>
```

●リスト 5-4 は，日本語タグで冗長ではあるが，メタデータの役割は果たしており，ヒトにとって理解しやすい。そこで，●リスト 5-3 と●リスト 5-4 の両者の利点を合わせると，一例として，●リスト 5-5 のような構造が考えられる。

この構造には，●リスト 5-3 や●リスト 5-4 と異なり，書誌情報と内容の親となる要素（<report>）がある。つまり，運用期間中の変更耐性を考え，安全策として，あらかじめ，中途階層を追加しているのである。これにより，報告書が結果的に大部数になった場合の分割や，共著による分割に耐える構造になる。複数文書の書誌情報のみのリストファイル作成が必要となった際にも，変更を吸収しやすいのである。

●リスト 5-5　ヒトによる理解とコンピュータによる計算の両方を考慮した構造の一例。

```
<?xml version="1.0"?>
<dataroot>
  <report>
    <head>
      <title uri="report/2009/xml01.html" date="2009-01-01">XML の現状と課題</title>
      <author company="XML 株式会社">須木間定義</author>
    </head>
    <contents>
      <chapter header="第 1 章">
        <section header="第 1 節">
          <div header="第 1 項">
            <para>1 段落目</para>
            <list>項目</list>
            <table>表 1</table>
            <image>図 1</image>
          </div>
        </section>
      </chapter>
    </contents>
  </report>
</dataroot>
```

ヒトによる理解とコンピュータによる計算効率のバランスをとるには，XML データ処理アプリケーションの開発者の立場に立って，構造を検討するとよい。なぜなら，開発者は自分が理解しやすいタグ名を付け，プログラムしやすい構造を考えるからである。結果的に，バランスのよい設計に落ち着くことになる。

XML1.0 勧告から数年間は，コンピュータによる処理速度を優先せざるをえない面もあったが，ハードウェアは今このときも進化している。処理効率を第一に考える必要は薄くなってきている。あえて短いタグ名を付ける必要はないし，無理やりフラットでシンプルな構造にする必要もない。フラットな構造は，階層が深く不定形の構造に比べ，単純に兄ノードから弟ノードの方向へと走査していけばよいので処理は速い。だが，それならば RDB でよく，データの活用範囲を拡げないのであれば，あえて XML 化する必要性があるとはいえないであろう。

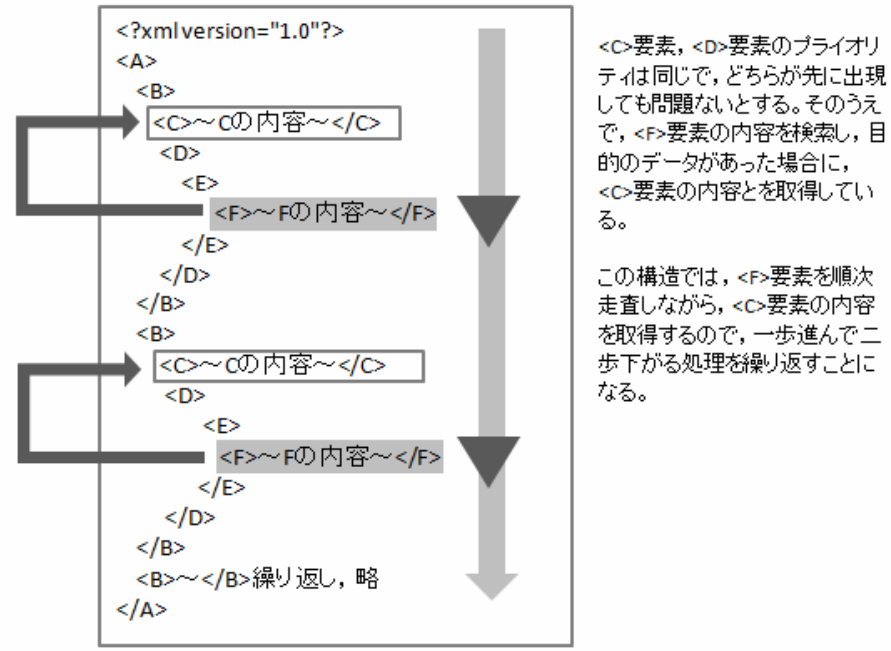
コンピュータに計算できてもヒトが理解しにくい構造にすべきではないのである。「XML の 10 の目標」に掲げられた，次の 2 点を肝に銘じておこう。

6. XML documents should be human-legible and reasonably clear.
10. Terseness in XML markup is of minimal importance.

ただし，ヒトが理解しやすければ，処理効率を全く考慮しなくてもよいというわけではない。

カレント・ノードから，兄の階層へさかのぼったり，親や祖先の階層にさかのぼってノードを特定するような，いわゆる「一歩下がって二歩進む」ような処理が必要となる構造は，XML 化の意味を損なわない限り，避ける方がよい（●図 5-7）。

●図 5-7 走査効率の面から，「一歩下がって二歩進む」構造は避ける



複数のスタッフで構造を検討する場合，参加者が開発者でなければ，あるいは開発者であってもXMLが専門外であれば，むしろヒトは理解できるが，コンピュータで処理しにくい，つまりTips & Tricksを要求するプログラミングが必要となる構造を善しとしてしまうことがある。そういった考えは，提案として受け止め，やみくもに迎合するのではなく，良い部分のみ採用したいものである。

5.2.3 技術以外の視点による微調整

以上の作業によって，荒削りではあるが，木構造の形が見えてきたはずである。

さて，ここで，すこし技術以外の話をしておかなければならない。変化への耐性を高めるためには，ビジネス展開，ヴィジュアル・デザイン，ユーザ・インタフェース，広告宣伝までを視野に入れて，木構造の微調整を行わなければならない。

だが，もし，打ち合わせ相手が将来展開を知りえない立場の担当者であれば，結果的に依頼漏れが発生するかもしれない。

きわめて簡単な例を 2 つ，あげてみよう。

たとえば，商品である飲料品の内容を表すために，<内容量>要素と<価格>要素が必要であるとする。設計者は，<内容量 単位="ml">要素を設定し，そのデータは，180，350，500，1000 といった数値であると考える。また，<価格>データも数値であり，昇順や降順のソート，小計の算出もできると考える。

ところが，作業をすすめるうちに，<内容量>に，1 ケース（1 本 180ml×6 本入り）といった文字列データが出現する。また，社内の管理用に用いる販売用の商品データと，Web サイト上で公開する広告データが異なり，後者の<価格>には「会員特価」といった文字列データを記録する必要が生じる。

これらを数値として集計するのであれば，<内容量 単位="ケース" 内容量="1 本 180ml×6 本入り">1</内容量>，<価格 割引="会員特価">720</価格>といった構造も考えられる。

また，たとえば，顧客が―――もうすこし生活に密着したデータを使おう―――袋詰めした干ししいたけと大豆を，販売しているとする。では，袋詰め商品だけでなく，即席だし用の「粉末」を発売することになったら，どうするのか。だしをとるための容器や，保存容器も扱い始めたらどうするのか。「アレルギー」の項目が必要となったり，さらに法改正により原材料表示方法が変わったり，アレルギーの要因が細分化された場合はどうするのか。しいたけそのものではなく，しいたけの原木の産地や化学物質まで記載するようになったらどうするのか。ならば，設計段階から，包装や容器やアレルギーや原木などを表す要素を追加しておいた方がよいということになるだろう。

これらのことに設計途中の現段階で気が付けば，企画段階にさかのぼって見直さなくとも，多少の要素の入れ替えや追加によって調整することができる。しかし，設計が完了して，XML データ処理アプリケーション開発が開始された後で気付いたのでは，プログラマに負荷をかけるか工期の見直しが生じるという意味で，手遅れとなる。

そういった意味からも，XML 設計者は，2〜3 の得意な業種の知識と経験を得た上で，設計を行うことが望ましい。

以上，大枠の設計ができたら，設計用サンプル・データを用いて，見本となるサンプル XML 文書を作成してみよう。そして， XML Schema ファイル作成の有無や検証の有無に関わらず，●表 5-6 の事項について詰めておこう。この検討の過程および決定事項は，仕様書の一部として記録しておくようにする。

●表 5-6 大枠の設計ができた段階での確認事項

タグ名	暫定的なメタデータ名(この時点での要素名，属性名の最有力候補)
構造	出現順序
	繰り返し
	最低出現回数，最高出現回数
	空要素を許可するかどうか
データ	データ型
	値の下限，上限
	データ長
	デフォルト値
	選択肢の値
	処理のためのキーとするメタデータ

5.3 文書内における整合性の確認

5.3.1 ノードの編集・追加・削除と整合性

物理的な動きとデータの不整合

見本となるサンプル XML 文書を作成しても，これで木構造が完成したわけではない。ここで作業を終えてしまったのでは，机上の設計に終わってしまう。机上の設計とは，文法上の間違いはないが，実務では通用しない設計のことである。

前掲●表 5-1 を，もう一度見てほしい。まだ多くの作業が残っている。今度は，XML データが活用された場合の整合性について，確認する必要がある。つまり，XML データの Edit･Add･Delete（編集･追加･削除）によって，不整合の生じる恐れがあるならば，不整合を排除できる構造に変更するか，あるいは，構造を変更することなく，データの編集･追加･削除におけるルールを決めて明文化しておくのである。なぜなら，XML データを編集･追加･削除した結果，「現実には生じていないデータ」が，「木構造データでは許可されてしまう」事態が，しばしば生じるからである。

特に，XML データが，ヒト･モノというマクロな物体の物理的な動きと連動するケースでは（前章「4.3 社会の中の XML～役割と課題～」参照），不整合が生じる前の時間に遡ってやり直すことができないので，注意が必要である。

モノの所在と，データの不整合

具体的に説明しよう。一意な一個の商品（※6）が，同じ時間に同じ場所にある―――これは，当たり前である。6 時に製造者の倉庫に存在する商品は，6 時に製造者の倉庫に存在している。また，一商品が，違う時間に，同じ場所に再び存在することもある。たとえば 6 時までは製造者の倉庫に存在し，それを出荷して，8 時から 17 時までは販売店に存在したが，売れ残ったために引き取られ，20 時には製造者の倉庫に存在することもある。

だが，一意な一個の商品が，ある特定の同じ時間に，違う場所に存在することはない。たとえば，同じ 7 時に，同一の商品が，製造者の倉庫と販売店の両方に存在することはない。ところが，物理的な動きを正しく表現できる設計でなければ，●リスト 5-6 のように，品番もシリアル番号も同じ，つまり完全に同一の商品が，同一時刻に 2 社に納品されたことになってしまう。

この例では，もし，A 社大阪支店に納品予定だったが，高松支店に変更になったのであれば，<納品

先>要素を追加したうえで<納品先>要素の内容テキストを編集するのではなく，<納品先>要素を追加することなく，"出荷時刻"属性値と，<納品先>要素の内容テキストを編集すべきである（●リスト 5-7）。つまり，変更履歴が不要であれば，<納品先>要素は，1 回しか出現しないものとして設計しておくべきである（※7）。

●リスト 5-6 一意な商品が，重複して存在する例。

```
<?xml version="1.0"?>
<出荷報告書 品番="ABC" シリアル番号="001">
<納品先 出荷時刻="2008/12/01T07:00">A 社大阪支店</納品先>
<納品先 出荷時刻="2008/12/01T07:00">A 社高松支店</納品先>
</出荷報告書>
```

完全に同一の商品が，同一時刻に別の場所に納品されたことになっている。

●リスト 5-7 要素を追加するのではなく，データのみ編集している。

```
<?xml version="1.0"?>
<出荷報告書 品番="ABC" シリアル番号="001">
  <納品先 出荷時刻="2008/12/01T07:00">A 社高松支店</納品先>
</出荷報告書>
```

※6：ここでいう任意の一商品とは，2 部の新聞のように別のモノでありながらシリアル番号を持たないために交換可能で「別のモノではあるが同一のデータとして表現されるモノ」ではなく，「完全に一意なモノ」である。

※7：XML Schema でいえば，●リスト 5-7 で，<納品先>要素の出現回数を設定するには，"maxOccurs"属性を使う。

モノの状態と，データの不整合

また，一商品が，異なる状態で，同じ時間に，同じ場所に存在することもない。●リスト 5-8 では，未開封の商品と，開封済みの商品が重なっている。このケースでは，<商品 状態="未開封｜開封済み">といったように，編集する属性値は 1 個にする方がよい（※8）。プログラムで処理する場合でも，1 個の属性値を更新するだけで済む（●リスト 5-9）。

●リスト 5-8 一意な商品が，異なる状態で，同じ時間に，同じ場所に重なって存在する例。

```
<?xml version="1.0"?>
<商品 品番="ABC" シリアル番号="001" 未開封="YES" 開封済み="YES">
  ～商品の詳細データ～
</商品>
```

完全に同一の商品が，未開封であり，開封済みである。

●リスト 5-9　属性の用い方によって，重複を避ける方法。

```
<?xml version="1.0"?>
<商品 品番="ABC" シリアル番号="001" 状態="開封済み">
  ～商品の詳細データ～
</商品>
```

※8：XML Schema でいえば，●リスト 5-9 で，属性値が「未開封」あるいは「開封済み」のいずれかであるようにするには，<xsd:enumeration>を使う（第 3 章，●リスト 3-46 参照）。

ヒトやカネの動きと，データの不整合

このような不整合は，ヒトの動きに連動するデータでも起こりうる。

同じヒトが，同じ時間，違う場所に存在することは，あってはならない。たとえば，同じ 10 時に，同じ一人の人物が，A 医院と B 医院で健康診断を受けていることはない。また，同じヒトが，異なる状態で，同じ時間に存在することも起こりえない。一人の人物が，メタボリックな状態と，スリムな状態で，同じ日時に健康診断を受けることはない。

ヒト・モノは物理的な動きとして捉えやすいが，カネについては，札や硬貨，あるいは対価となる商品といったモノではなく，データが動いているように錯覚しがちである。しかしながら，同じ 10 時に，A 氏から外注先の B 氏に支払う予定の 100 万円が，A 氏から B 氏へ支払われた状態と，まだ支払われていない状態で存在することはない。また，同じ 10 時に，支払い予定の 100 万円が，A 氏の手元と，B 氏の口座の両方に存在することもないのである。

そのような，「現実には生じていない」誤ったデータが大量に蓄積された後で，不備に気付く事態は避けなければならない。

5.3.2 不整合を抑止するルール作り

要素・属性・テキストの処理

以上のことからも分かるように，本章冒頭で述べたとおり，XML 文書は「静的なテキスト」というよりもむしろ，「アプリケーションから利用される動的なデータ」として捉えた方がよいのである。設計者は，サンプル XML 文書中のノードの動きを，イメージしながら，設計を進めなければならない（※9）。

なに，不整合が生じたならば，DOM を使ってプログラムを記述して修正すれば済むではないか，などと気軽に考えてはならない。そのような単純な問題ではない。ひとくちに，要素や属性やテキストの編集・追加・削除といっても，複数の処理が考えられるのである（●表 5-7）。

どのような処理を，どのようなタイミングで実行するのか―――XML 設計者と，XML データを処理するアプリケーションの開発者との間で，解釈が異なることのないよう，ルールを作っておく必要がある。

※9：設計した XML 文書を活用する場合だけでなく，フォームから入力されたデータをもとに XML 文書を生成するケースでも，編集・削除・追加が生じるという点では，同様に，整合性について考慮すべきである。

●表 5-7 要素･属性･テキストの編集･追加･削除

処理	操作対象	処理内容
編集	要素	順序を入れ替える。
		階層を入れ替える。
		要素名を変更する。
	テキスト	要素の内容テキストを変更する。
	属性	従属先の要素を変更する。
		属性名を変更する。
		属性値を変更する。
追加	要素	要素を末尾に追加する。
		要素を任意の階層の任意の位置に追加する。
	テキスト	空要素にテキストを追加する。
	属性	属性を任意の要素に追加する。
		値のない属性に値を追加する。
削除	要素	要素を削除する。要素の内容テキストや属性や子孫要素がある場合は，それらも削除する。
		要素を削除する。要素の内容テキストや属性がある場合は，それらも削除する。子孫要素は削除しない。
	テキスト	要素の内容のみ削除する。つまり空要素にする。
	属性	属性を削除する。
		属性値を削除する。属性はあるが値がない状態にする。

ノードの編集と整合性

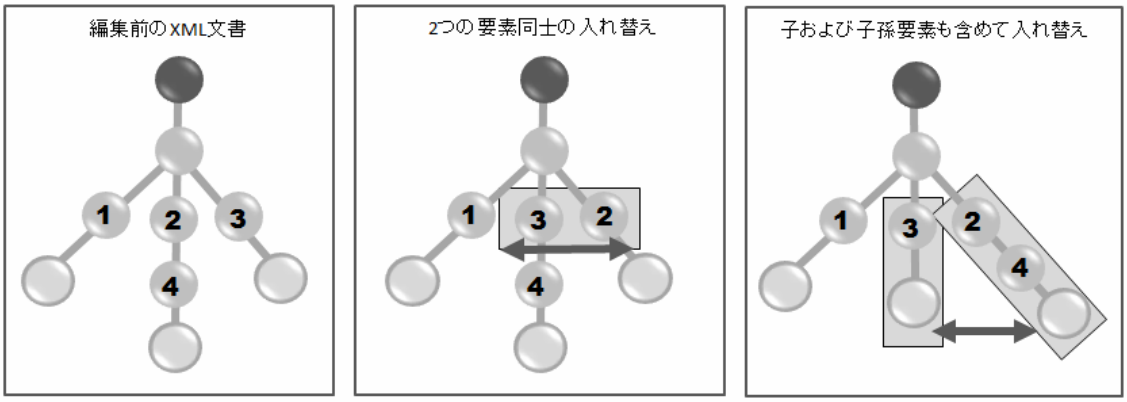
●表 5-7 に挙げたとおり，要素の編集の基本的な方法には，3 種類ある。

1つ目は，要素の順序を入れ替える処理である。2つの要素同士を入れ替えることもあれば，複数の要素を任意の順番に順次入れ替えることもある。これには単純に任意の要素同士を入れ替える場合と，子および子孫要素も含めて入れ替える場合がある（●図 5-8）。

2 つ目は階層を入れ替える処理である。親や祖先の階層，あるいは子や子孫の階層にある任意の要素と入れ替える。

3 つ目は，要素名を変更する処理である。名前空間を使っている場合は，接頭辞はそのままでローカル名のみ変更するのか，名前空間および接頭辞も変更するのかも検討する。

●図 5-8 要素の順序の入れ替え



要素の内容テキストを編集することもある。最も頻度が高いのは，要素名はそのままで，データのみ上書きする処理である。また，既存の要素の内容テキストの任意の文字列位置に，データを挿入することもある。たとえば任意の要素の内容テキストの末尾に，単位を付加するなどである。

属性の編集も考えられる。1 つは，従属先の要素を変更するというものである。たとえば，<節>要素に従属していた"小見出し"属性を，1 階層上の<章>要素の属性に変更する，といったものである。2 つ目

は，属性名を変更する処理である。そして，3 つ目は，属性値を変更する処理である。頻度は低いが，属性値の任意の文字列位置に，データを挿入するケースも考えられなくはない。

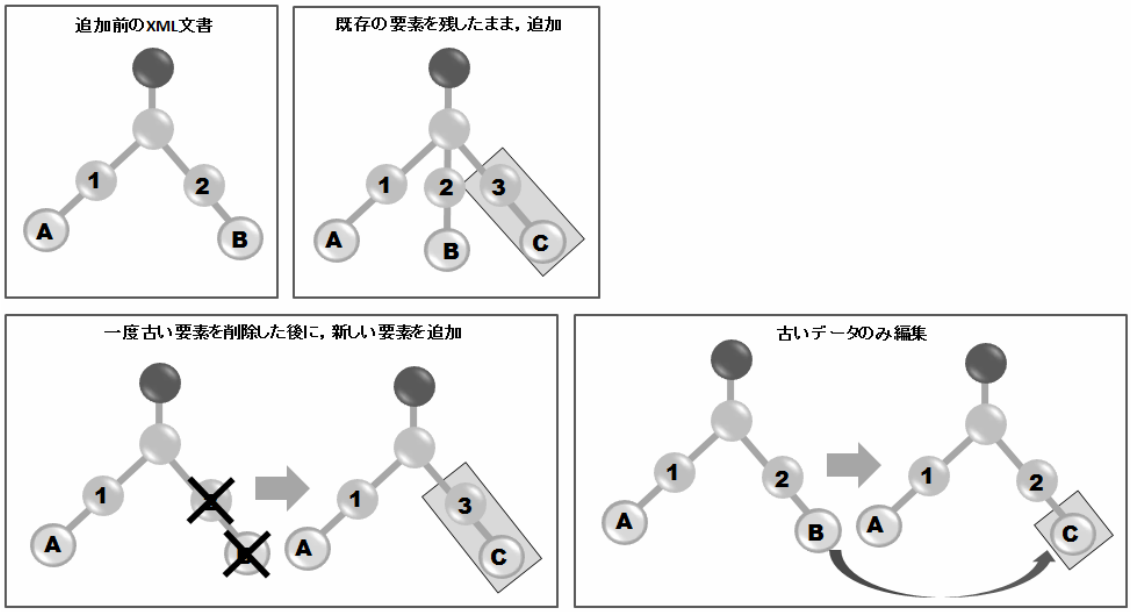
ノードの追加と整合性

ノードの追加には，注意が必要である。単純に追加しただけでは，設計の意図から外れた構造になる恐れがある。

要素ならば，兄弟要素の末尾に追加するのか，あるいは任意の階層の任意の位置を特定して追加するのか，ルールを決めておこう。テキストは，空要素かどうかを判別して，内容がなければ，追加することもある。属性を，任意の要素に追加することもあれば，値のない属性に，属性値を追加することもある。

変更履歴が必要となるケースでは，既存の要素を残したまま，同じ名前の新しい要素を追加する。たとえば仕様の変更履歴であれば，新旧両方の要素に，要素生成日時を値とする属性も同時に追加することによって，履歴が残るようにする方法が考えられる。逆に，変更履歴が不要であれば，古い要素を削除して新しい要素を追加したり，データのみ編集する方法も考えられる（●図 5-9）。

●図 5-9 追加にあたって既存の要素を残すかどうかを決めておく



ノードの削除と整合性

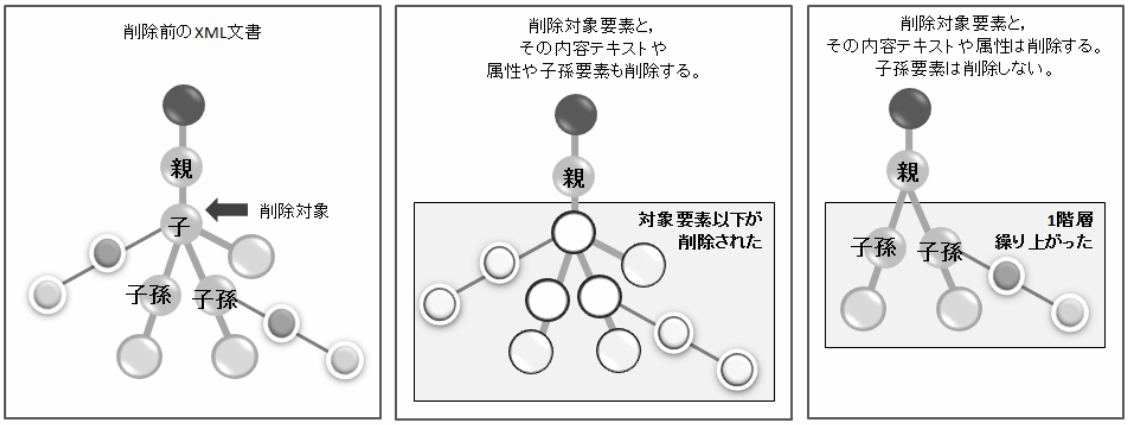
ノードの削除には，細心の注意が必要である。意図して消すことはなくとも，誤って消すことは容易だからである。削除の制限事項は，設計段階から考えておかなければならない。

処理対象が要素ノードである場合，要素ノードを削除すれば，その子孫も削除するのが一般的である。一方で，要素の内容テキストや属性がある場合はそれらも削除するが，子孫要素は削除しないという方法もある（●図 5-10）。後者では，1 階層の繰り上がりが発生する。

逆に，整合性を維持するために，子孫がある場合は，その親の削除を許可しないこともある。

通常，アプリケーション開発者は，「ノードを削除する場合は，その子孫ごと削除する」と捉えるので，仕様書には，具体的な指示を明記しておく必要がある。

●図 5-10 削除対象要素の子孫を削除する場合と，削除しない場合



処理対象がテキストである場合は，テキストを削除する，つまり要素の内容のみ削除して空要素にすることがある。

属性の削除では，属性そのものを削除する場合と，属性値のみ削除する場合の 2 通りが考えられる。

5.3.3 ノード間の関係と処理実行のタイミング

編集･追加･削除の方法だけでなく，それらを実行するタイミングも重要である。最も分かりやすい例をあげれば，要素自体を削除してしまった後で，その要素名やデータを変更することはできない。

実際のソースコードで見ておこう。移動を分かりやすくするため，社員の異動を表したデータを例にとる。●リスト 5-10 は，企業内のヒエラルキーを表した構造である。人事異動に伴って単純に<氏名>要素を追加すると，同一人物が重複する結果になる。このリストでは，「須木間定義」氏が重複している。

●リスト 5-10 社員の所属部署が変更された例。古い要素を残したまま新しい要素を追加している。

```
<?xml version="1.0"?>
<社員名簿 企業名="XML 株式会社">
  <従業員 部署名="IT 事業部">
    <氏名>江楠絵夢</氏名>
    <氏名 異動="2008-03-31">須木間定義</氏名>
  </従業員>
  <従業員 部署名="システム部">
    <氏名 着任="2008-04-01">須木間定義</氏名>
    </従業員>
</社員名簿>
```

異動と転属の両方のデータが記録されている。

●リスト 5-10 の構造では，異動や入退社が多ければ，データは際限なく膨らんでしまう。そこで，●リスト 5-11 のように，古い要素を削除して新しい要素を追加すると，重複は避けられるが，今度は異動元の所属部署を知る術が失われる。

●リスト 5-11　古い要素を除き，新しい要素を追加して更新した例。

```
<?xml version="1.0"?>
<社員名簿　企業名="XML 株式会社">
  <従業員　部署名="IT 事業部">
    <氏名>江楠絵夢</氏名>
  </従業員>
  <従業員　部署名="システム部">
    <氏名　着任="2008-04-01">須木間定義</氏名>
  </従業員>
</社員名簿>
```

「須木間定義」氏の異動元のデータが削除されている。

では，次の●リスト 5-12 のように，レガシーXML データを社員ごとに分割した，複数ファイル構成を採用するとどうなるか。系列企業や支社の XML 文書ファイルが個別に管理されていると，社員が退社した後で事務手続きが生じた際に，データを追跡することが困難になる。

●リスト 5-12　社員の所属部署が変更された例。企業内ヒエラルキーによる構造。

・関西支社で管理しているデータ

```
<?xml version="1.0"?>
<社員名簿　企業名="XML 株式会社"　支社名="関西支社">
  <従業員　部署名="IT 事業部">
    <氏名>江楠絵夢</氏名>
  </従業員>
</社員名簿>
```

2008 年 3 月 31 日付けで，IT 事業部から異動した「須木間定義」氏のデータが削除されている。

2008 年 4 月 1 日に異動してきた「須木間定義」氏の，以前の所属部署データが不明である。

・四国支社で管理しているデータ

```
<?xml version="1.0"?>
<社員名簿　企業名="XML 株式会社"　支社名="四国支社">
  <従業員　部署名="システム部">
    <氏名　着任="2008-04-01">須木間定義</氏名>
  </従業員>
</社員名簿>
```

では●リスト 5-13 のように，社員単位に分割したデータファイルを作成し，●リスト 5-14 のような<社員 ID>と<氏名>のみの管理ファイルを別個に作成して，連携する方法ではどうか（●図 5-11）。この構造であれば，データの追跡も可能である。また，詳細データを記録する XML 文書には，これ以外にも，参加プロジェクトや実績や給与，勤務日数，健康診断結果など，個々の社員のデータを含めることが可能であると考えられる。

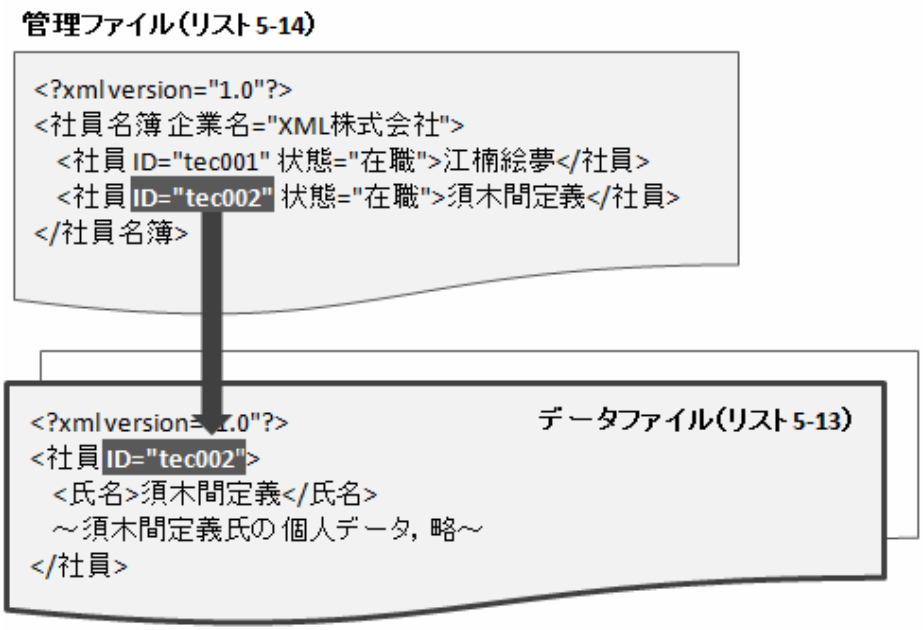
●リスト 5-13 社員単位に分割したデータファイル。社員の詳細データ。

```
<?xml version="1.0"?>
<社員 ID="tec002">
  <氏名>須木間定義</氏名>
  <異動履歴>
    <所属 部署名="IT 事業部" 着任="2007-04-01" 転属="2007-03-31">関西支社</所属>
    <所属 部署名="システム部" 着任="2008-04-01">四国支社</所属>
  </異動履歴>
  <資格>～略～</資格>
  <連絡先>～略～</連絡先>
</社員>
```

●リスト 5-14 管理ファイル。<社員 ID>と<氏名>のみのリスト。

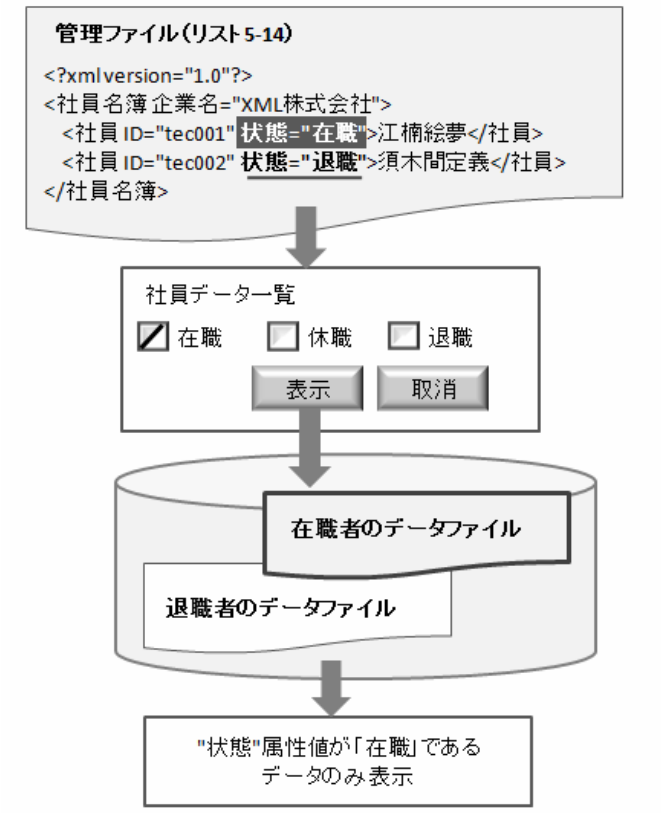
```
<?xml version="1.0"?>
<社員名簿 企業名="XML 株式会社">
  <社員 ID="tec001" 状態="在職">江楠絵夢</社員>
  <社員 ID="tec002" 状態="在職">須木間定義</社員>
  <社員>～</社員>を在職した社員の数だけ繰り返し，略。
</社員名簿>
```

●図 5-11 ●リスト 5-13 のデータファイルと，●リスト 5-14 の管理ファイルを，"ID"属性値によって連携処理。



この方法であれば，もし「須木間定義」氏が退社したならば，●リスト 5-12 中の"状態"属性値を「退職」に変更するだけでよい。在職中の社員の詳細データが必要な場合は，「"状態"属性値が在職である」という条件でフィルタリングすれば済む（●図 5-12）。"状態"属性値が「退職」である社員のデータファイルは，社内サーバに蓄積したままであっても支障はない。不要なファイルは廃棄すべきだが，過去に在職した社員データのような，必要になる可能性のあるファイルは残しておく方がよいだろう。

●図 5-12 退職後にデータファイルが蓄積されたままであっても，管理ファイルを用いてフィルタリングできる。



ただし，複数ファイルを連携させるには，各々のファイルに記録するデータの線引きが重要である。●リスト 5-14 の管理ファイルに，支社名や部署名まで記載してしまうと，異動の度に，管理ファイルとデータファイル双方のファイルの支社名や部署名を変更しなければならない。ここでは，一社員のデータのみ例として挙げているが，データ件数が多ければ，更新作業の煩雑さから，編集ミスによる不整合を引き起こす恐れがある。

以上はあくまで説明用の簡易な例であって，企業規模や企業構造，管理するサーバの設置･管理状況，XML 文書の使用目的によって最適の構造は異なる。もし，各社員の所属先部署での実績も記録するならば，さらに細分化した構造が必要になるであろう。

データ内容やデータ件数のいかんに関わらず，設計段階から，編集･追加･削除によって生じる不整合の可能性を想定して，ルールを決めておくことの重要性を，分かっていただけただろうか。

5.4 文書ファイル群の構成

5.4.1 複数 XML 文書ファイルの設計

ファイル構成とファイル間の関係

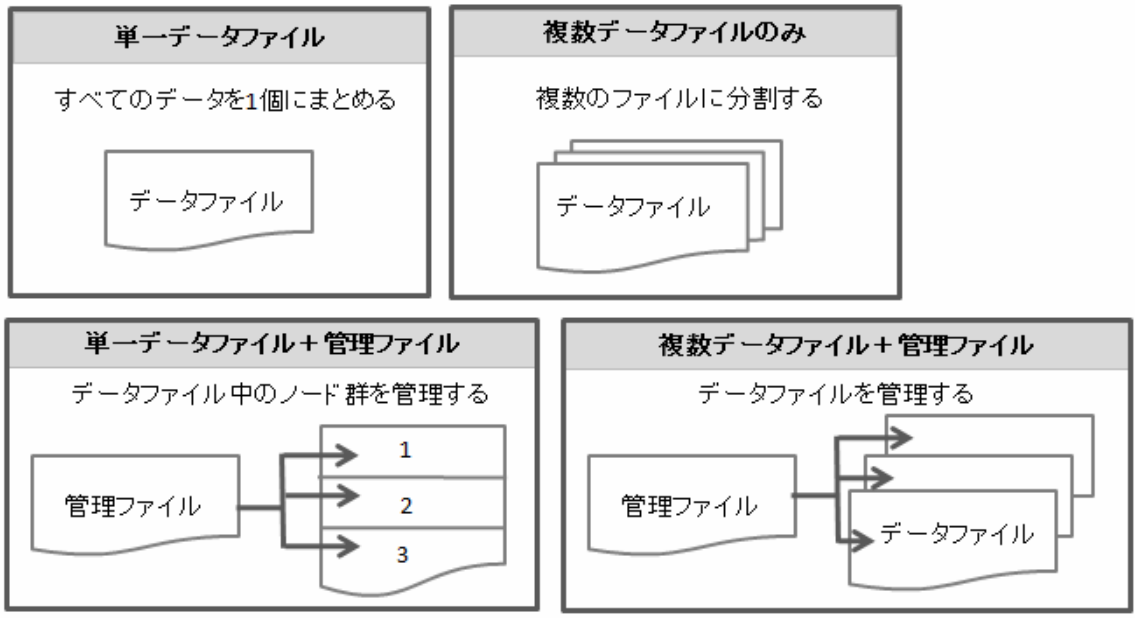
レガシーXML データは，必ず単一の文書ファイルとして作成「しなければならない」わけではない。前節●図 5-11 のように，社員ごとの複数データファイルを作成することもある。

また，レガシーXML データは，その構造のままで活用「しなければならない」わけではない。活用に際して，分割したり，別の構造に変換したファイルを作成してもよい。

さらに，XML 化の目的の処理を，単一のファイルのみで実現「しなければならない」わけでもない。複数文書ファイルによる構成で実現してもよい。

XML 文書ファイルの構成は，●図 5-13 のように，主に 4 種類である。データを記録した文書ファイルが 1 個であっても，データを管理するファイルを別途作成して，複数ファイル構成とすることもある。

●図 5-13 XML 文書ファイル群の構成



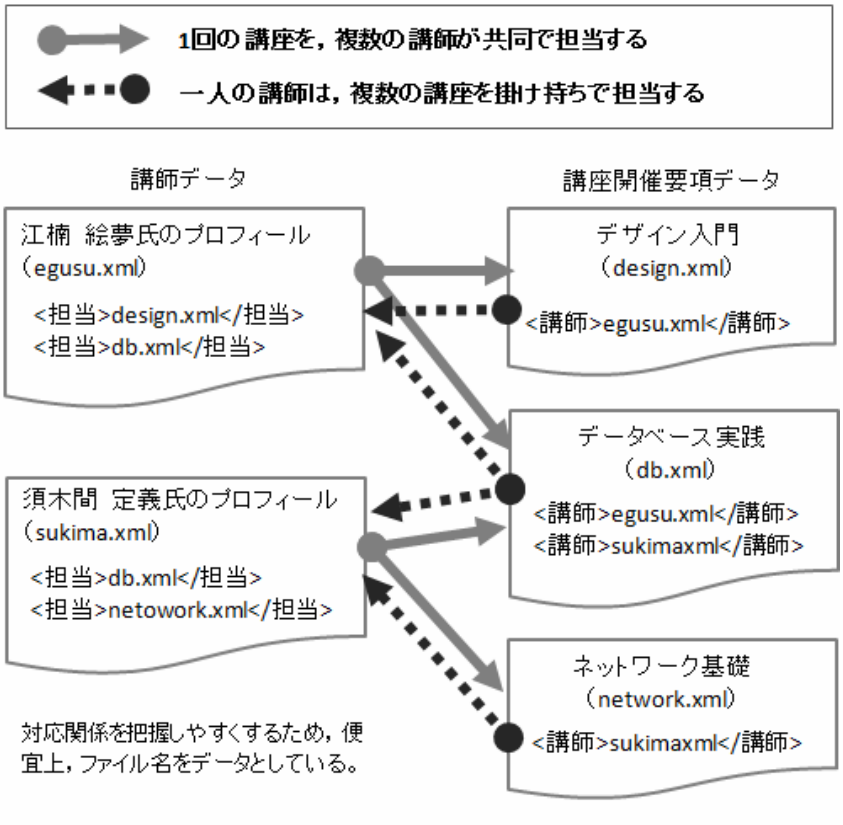
XML は木構造であるから，一文書内のノード間の関係は，1 対 1，あるいは 1 対多である。そして，ファイル間の関係も，基本的に，1 対 1，1 対多，多対 1 である（後述）。例をあげれば，1 対 1 は「1 社員の個人情報」→「1 社員の履歴書」といった関係である。1 対多は，「全社員リスト」→複数の「各社員の個人情報」といった関係である。多対 1 は，複数の「各社員データ」→「勤務先情報」といった関係である。

多対多の関係は，頻度は低いが，可能ではある。たとえば，1 回の講座を複数の講師が共同で担当し，一人の講師は複数の講座を掛け持ちで担当する，という多対多の関係では，●図 5-14 のように，各 XML 文書の中に，対応する XML 文書ファイル名を記録することにより，プログラムで必要なデータを取得することができる。ただし，ファイル名ではなく，データを多対多とする関係では，処理方法によっては，双方からの参照が無限ループに陥りかねないので注意が必要である。

このように，設計者は，一文書内の構造のみならず，文書ファイル群の構成も検討しなければならないのである。

本書では，これ以降，便宜上，データを記録した XML 文書ファイルのことを「データファイル」と表記する。これは，前掲の●リスト 5-13 にあたるものである。また，管理ファイルの内容は，複数 XML 文書ファイル名を列記したファイルリストであることが多いが，それ以外のケースもあるため（後述），「管理ファイル」という表記としている。これは，前掲の●リスト 5-14 にあたるものである。

●図 5-14 複数ファイル間の多対多対応の関係



文書ファイルの構造

複数データファイルにおける，各 XML 文書ファイル内の構造は，必ずしも同一であるとは限らない。次のように，4 種類の構造が考えられる（次ページ，●図 5-15）。

(1) 同一構造の複数ファイルを作成する。
(2) 特定の階層以下を分割する。
(3) 定型部分と不定形部分に分割する。
(4) 別の構造に変換した XML 文書ファイルを追加作成する。

(1)は，レガシーXML データ内で規則的に繰り返すデータを 1 グループとして，分割するものである。前掲●リスト 5-13 のように，全社員のデータを個人単位のデータに分割するといったものである。他にも，商品データを商品分類別，発売時期別に分けたり，店舗や支社といった管理者別に分けることも考えられる。入力データから XML 文書ファイルを自動生成するのであれば，1 日分や 1 月分といった時間単位や，入力方法によって分割することもできる。

(2)は，階層構造が深く，且つデータ件数も多く，ユーザインタフェースのクオリティを守るために，任意の階層で分割した方がよい場合の措置である。

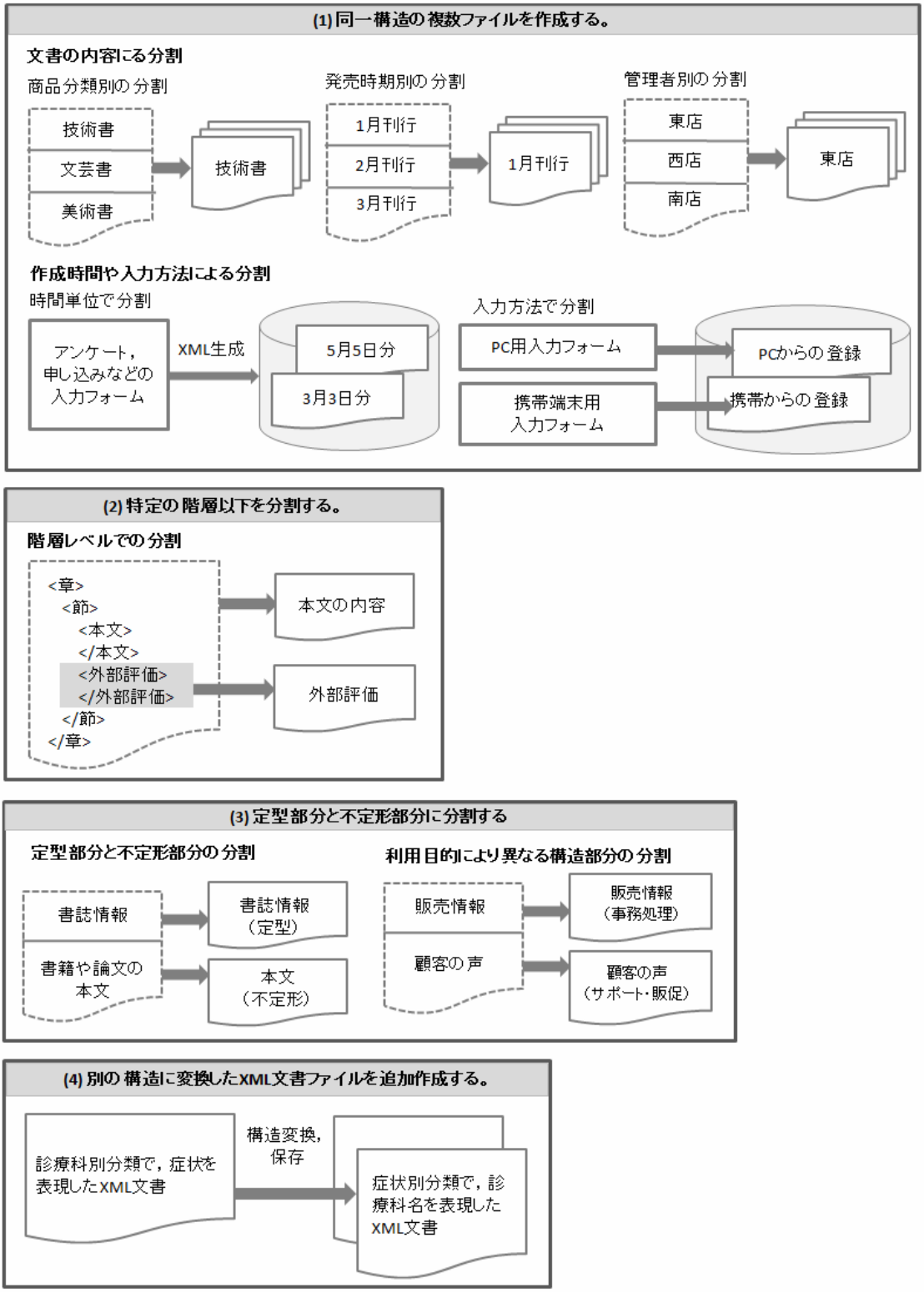
(3)は，更新可能性のあるデータが不定形部分に集中しているとき，更新作業の容易さから，不定形部分を切り離すものである。マニュアルや論文での，書誌情報（定形）と本文（不定形）の分割がこれにあたる。また，商品の販売情報と顧客の声のように，構造が異なる部分を分割するのも一方法である。

(4) は，レガシーXML データとは異なる構造の XML 文書を，別ファイルとして作成するものである。。XSLT スタイルシートで変換した結果ツリーを保存したり，DOM や LINQ を用いて異なる構造の XML 文

書を生成することがある。その場合，もちろん，レガシーXML データだけでなく，変換結果となる木の構造も設計しておく必要がある（第 1 章「1.2.2. 木構造は柔軟に変化する」参照）。

これらは，最初から複数ファイルとして作成することもあれば，レガシーXML 文書ファイルは単一ファイルとして作成しておき，活用に際して複数に分割することもある（次章で詳説）。

●図 5-15 複数文書ファイルの例



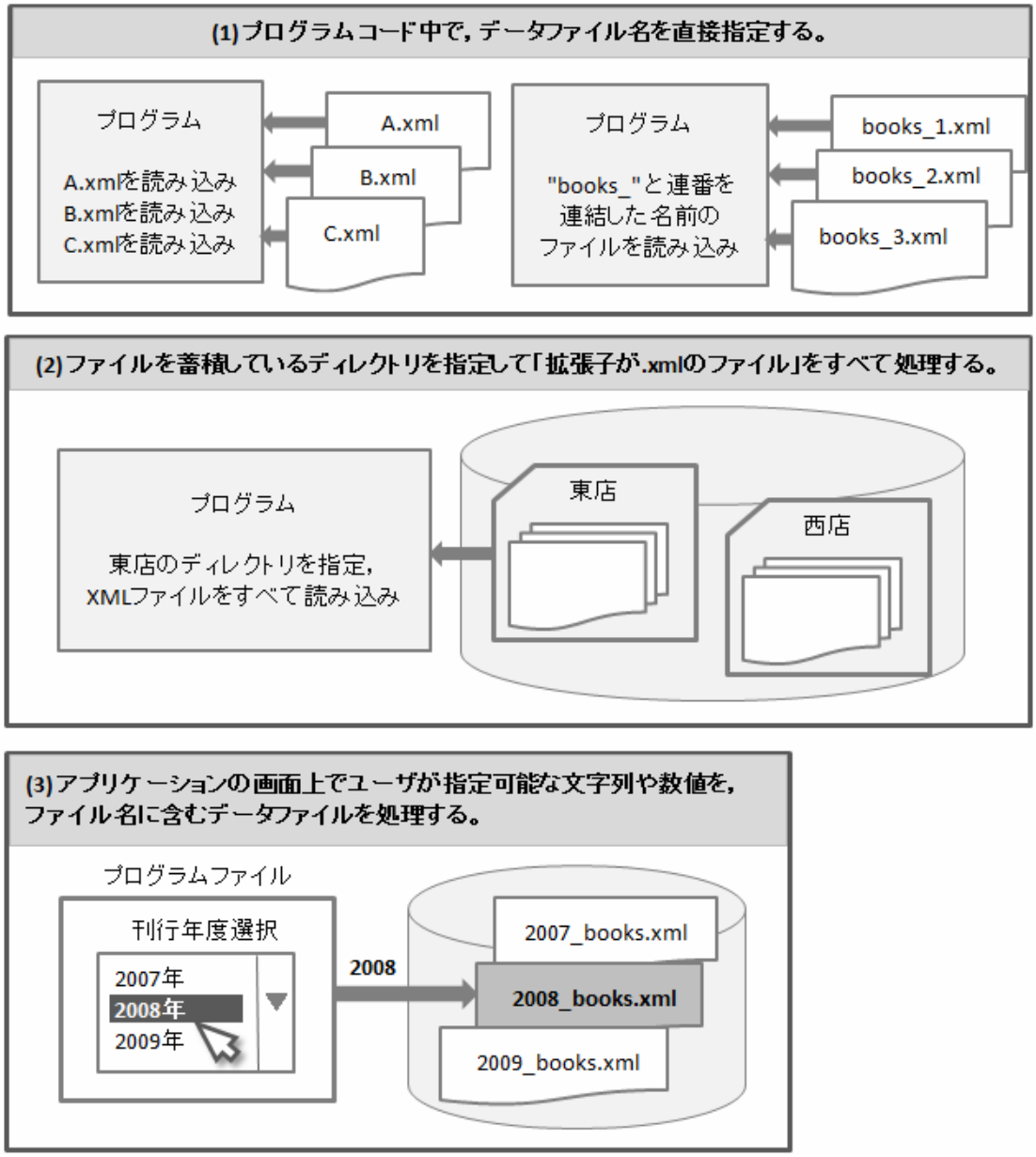
5.4.2 処理対象ファイルの指定

管理ファイルを併用しない場合

では，複数ファイルは，どのように用いればよいのか。複数ファイルを連携･参照･縦断処理するには，必ず，処理対象となるデータファイルを特定できる仕組みがなければならない。

まずは，管理ファイルを併用しない場合について説明する（●図 5-16）。

● **図 5-16 管理ファイルを併用せず，複数のデータファイルを特定する仕組み**



(1) プログラムコード中で，データファイル名を直接指定する。

(2) ファイルを特定のディレクトリに蓄積し，ディレクトリを指定して「拡張子が.xml のファイル」をすべて処理する。

(3) アプリケーションの画面上でユーザが文字列を入力あるいは選択すると，これをファイル名に含むデータファイルを処理する。

(1)は，運用期間中，データファイル名に変更が生じなければ，簡易な方法である。しかしながら，データファイルが追加，あるいは削除された場合には，プログラムコードを修正しなければならず，修正漏れのリスクがある。

(2)は，データファイルを蓄積した特定のディレクトリを，プログラムコード中で指定するだけでよい。複数のディレクトリに分けている場合は，複数の処理を記述すればよい。ただし，ディレクトリ内に処理対象ではないデータファイルが蓄積されてはならず，厳しい管理がもとめられる。参照先のデータファイルの過不足を確認するには，データファイルを置いたフォルダ内のファイル数と，管理ファイル内のノード数が一致しているかどうかを調べる。

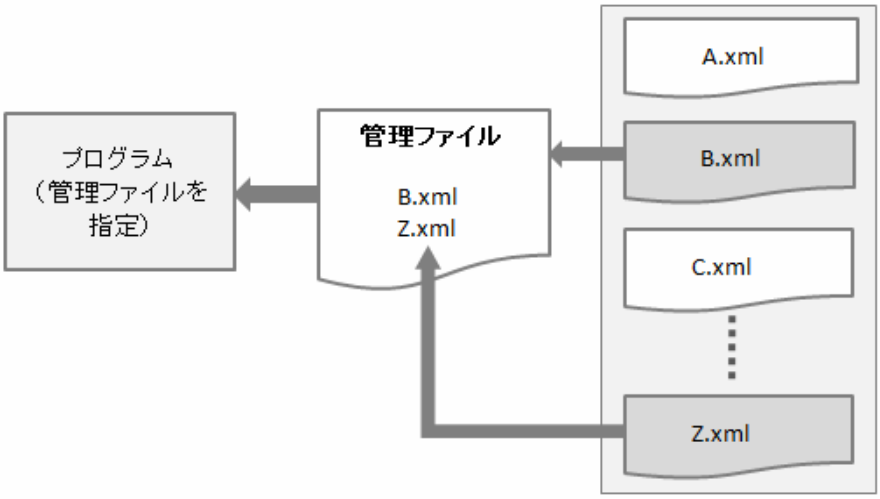
(3)は，XML データ処理 Web アプリケーションでよく用いられる方法である。たとえば，アプリケーションの画面上にある選択ボックスから商品名を選択すると，商品名をデータファイル名とする商品詳細を記録したデータファイルの内容を表示する，といった処理である。選択された項目のインデックスを，データファイルのファイル名に含めておく方法もある。

管理ファイルを併用する場合

複数ファイル構成では，管理ファイルを併用する方が，メンテナンスの容易なケースが多い。管理ファイルは，複数のデータファイル間の関係性を定義する役割を果たす。

管理ファイルを用いると，ピンポイントで処理対象とするデータファイルを指定することができるという利点もある（●図 5-17）。

●図 5-17 管理ファイルを用いると，全ファイル走査ではなく，ショートカットが可能になる。



管理ファイルを用いたデータファイルの指定方法には，主に，次の 7 種類がある。管理ファイルそのものは，通常，プログラムコード中で直接ファイル名を指定する。

(4) データファイルの URI を記録した管理ファイルを参照して，データファイルを処理する（※10）。

(5) 固定文字列と連番を連結したファイル名を持つ，データファイルを処理する。

(6) 管理ファイル中の，キーとなる要素の内容や属性値をファイル名とした，データファイルを処理する。

(7) 管理ファイルとデータファイルに存在する同一データを利用する。管理ファイル中の，キーとなる要

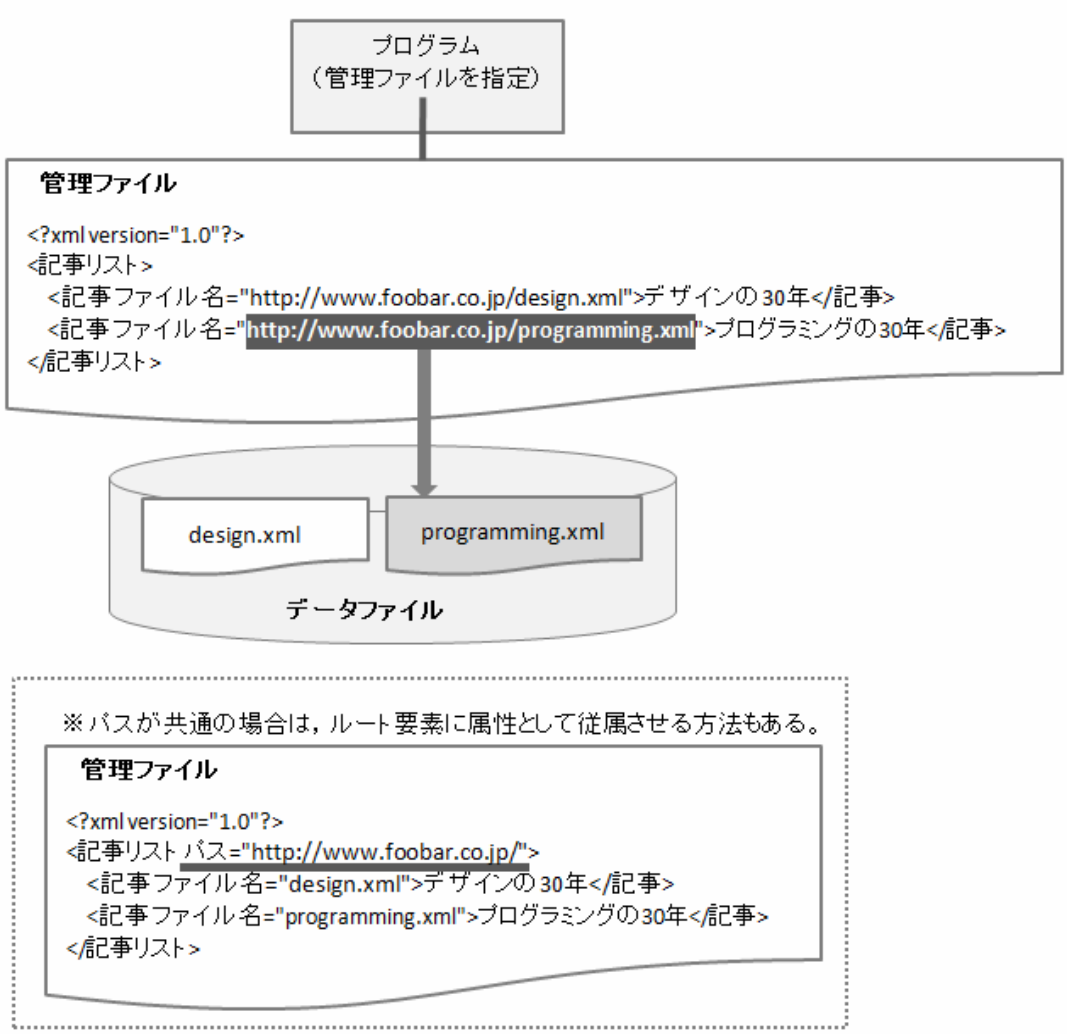
素の内容や属性値が，処理対象データファイル内の要素の内容や属性値に合致する，データファイルを処理する。

(8) 管理ファイル中の，キーとなる要素名あるいは属性名をファイル名とした，処理対象データファイルを処理する。

(4)は利用頻度の高い方法である。●図 5-18 のように，データファイルの URI をリストとして記録した，管理ファイルを作成しておき，記録された URI を持つデータファイルを処理する。この方法では，データファイルはテキスト主体とは限らない。XML 形式の画像や動画によるコンテンツであってもよい。

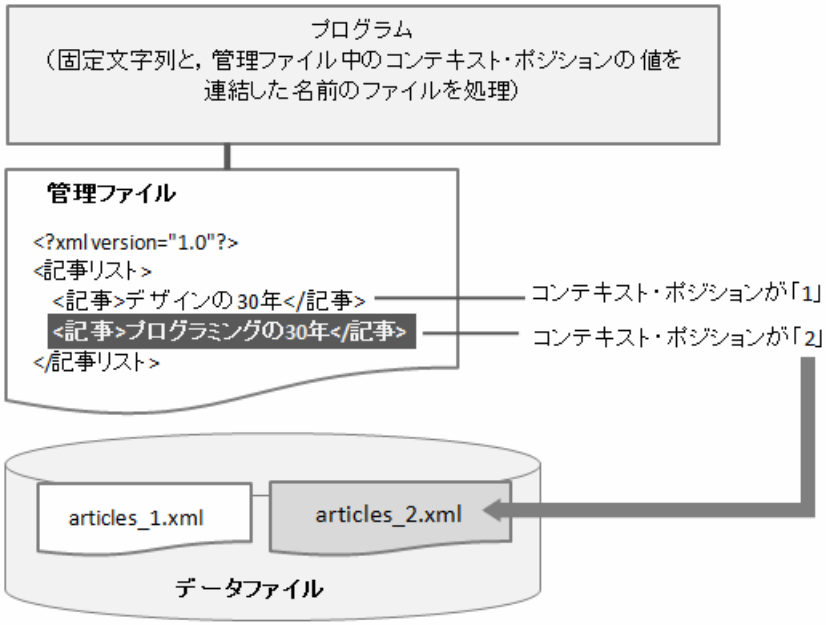
ただし，注意点がひとつ。たとえば，オンラインメディアの URI は，アーカイブやサイトのリフォームによって変更されることがある。URI の変更を管理ファイルに逐一反映させなければ，デッドリンク状態を招くことになる。

●図 5-18 処理対象データファイルの URI を，管理ファイルに記録する



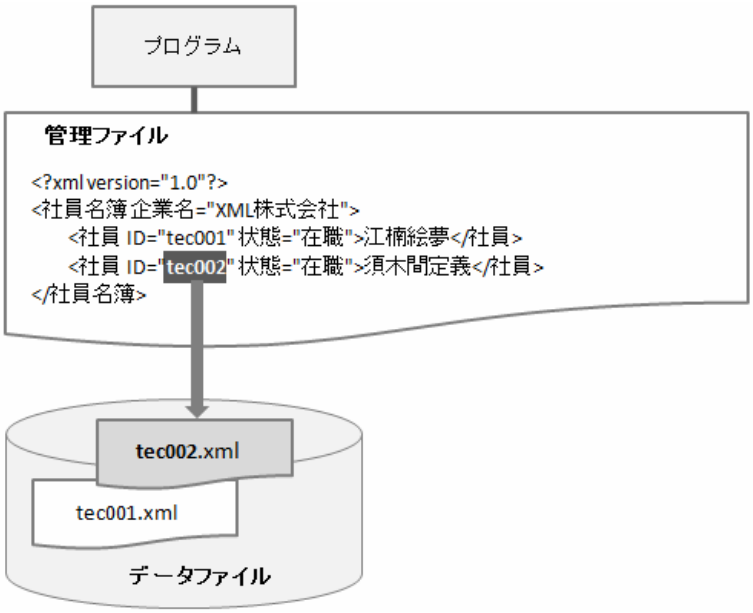
(5)は，各データファイルに，固定文字列と連番を連結したファイル名を付けておき（※11），数値をファイル名に含むデータファイルを処理するという方法である。ここでいう数値とは，多くの場合，管理ファイル中の任意のノードのコンテキスト・ポジションである（●図 5-19）。

●図 5-19 各データファイルに固定文字列と連番を連結したファイル名を付ける



(6)は，管理ファイル中のキーとなる要素の内容や属性値をファイル名とする，データファイルを処理するというものである（●図 5-20）。他にも，たとえば，管理ファイル中の任意の<型番>要素を選択すると，その<型番>要素の内容テキストをファイル名とするデータファイルの商品詳細データを表示する，といった処理に利用することができる。当然のことながら，データファイルの特定に用いる，管理ファイル中の要素内容や属性値は一意なデータでなければならない。

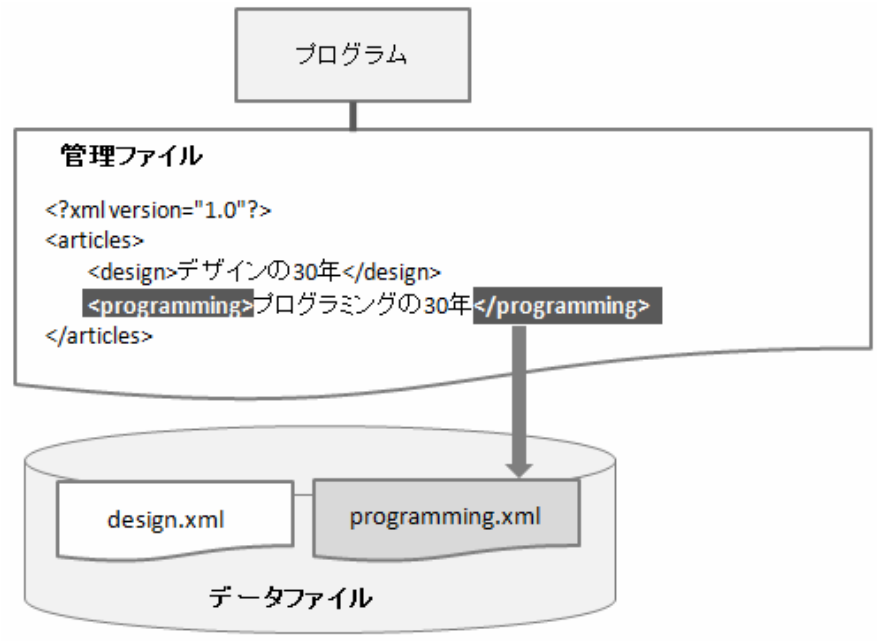
●図 5-20 管理ファイル中の要素の内容や属性の値を，データファイル名とする



(7)は，いわば一種の検索処理で，ユーザの選択したデータが，処理対象データファイル内に存在するデータファイルを処理する，といった方法である。前節●図 5-11 がこれにあたる。ただし，合致するデータが見つかるまでデータファイルを走査する必要があるため，データファイル数が膨大な処理には適していない。

(8)は(7)と異なり，管理ファイル中のデータではなく，キーとなる要素名あるいは属性名をファイル名とした，対象データファイルを処理する方法である（●図 5-21）。使用頻度は低い。

●図 5-21 管理ファイル中の要素名あるいは属性名を，データファイル名とする



※10：複数の XML 文書ファイルを管理するための物理的ロケーション（サーバ）が異なっても，縦断処理は可能である。ただし，IP アドレスの変更，一部サーバのメンテナンスによる停止，蓄積されるデータの品質と安全性，人的セキュリティなど，運用上の問題がある。また，各ファイルの公開のタイミングや，データ廃棄のサイクル，データ更新のサイクルが異なると，整合性の維持も課題となる。

※11：レガシーXML データを単純分割して，連番をファイル名に含むデータファイルを作成する場合，ミスを防ぐためにも，手作業ではなく，自動分割･生成プログラムを開発することが望ましい。

5.5 ユーザインタフェースの設定と適用

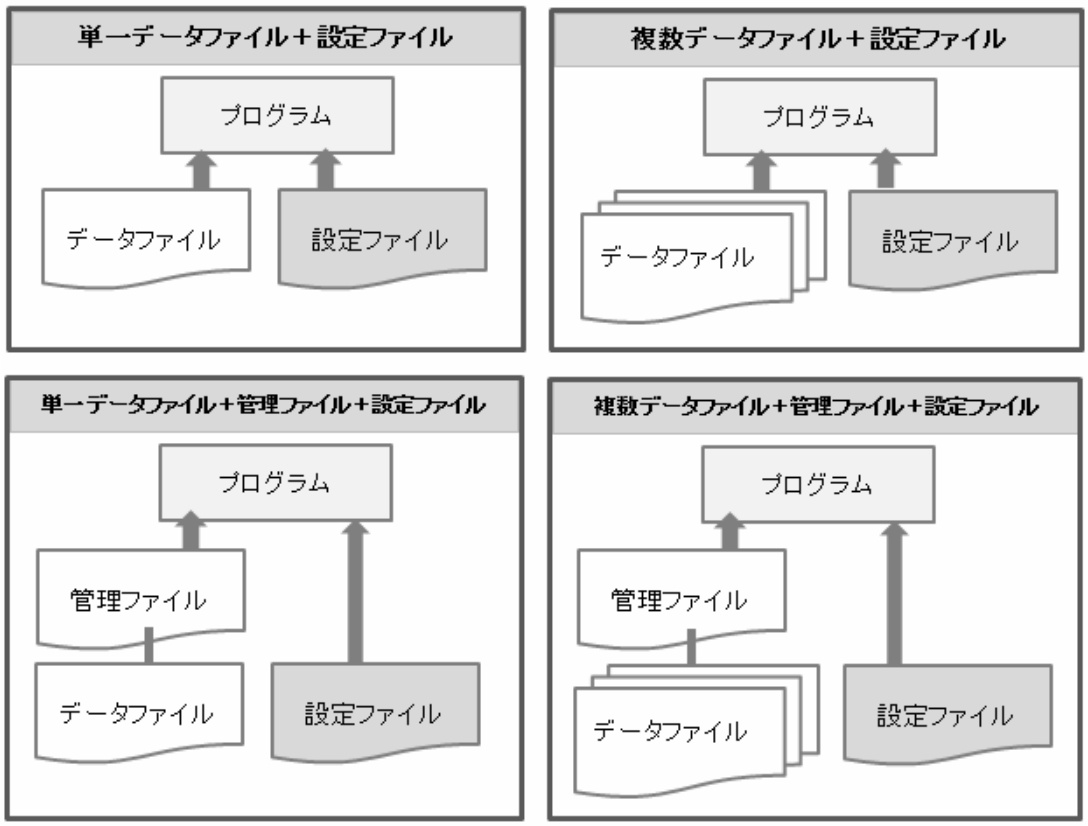
5.5.1 設定ファイルによる表示や動作の制御

XML 文書設計とアプリケーションの基本設計を併行するケースでは，アプリケーションの表示や動作を制御するための設定ファイルを，XML 文書ファイルとして設計することがある。設定ファイルを用いた，複数ファイル構成となる。

ただし，その目的は，これまでに述べたデータファイルや管理ファイルのように文書の蓄積や管理というよりもむしろ，アプリケーションの表現や機能の汎用性を高めることにある。また，管理ファイルがデータファイルと 1 セットでデータを表現するのと異なり，設定ファイルはプログラムやデザインの一部であって，データファイルとは別個に単独で成立する。設定ファイルがなければ，処理や表示に支障をきたすことはあっても，データが欠落するわけではない。

設定ファイルは，プログラムから直接指定して読み込んで処理する（●図 5-22）。本節では，ユーザインタフェースの設定と適用という目的にしぼって，設定ファイルの利用方法の一部を紹介する。

●図 5-22 設定ファイルを用いた，複数ファイルによる構成



※レガシーXMLデータを記録したXML文書ファイルが1個であっても，設定ファイルがXML形式であれば，XMLアプリケーションとしては，複数XML文書ファイルによる構成となる。

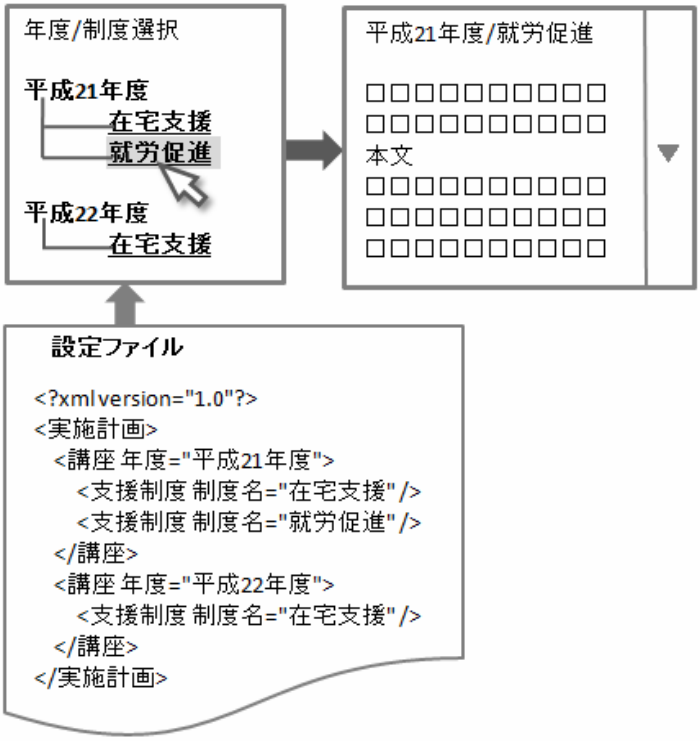
5.5.2 設定ファイルによるナビゲーションの制御

木構造のナビゲーション

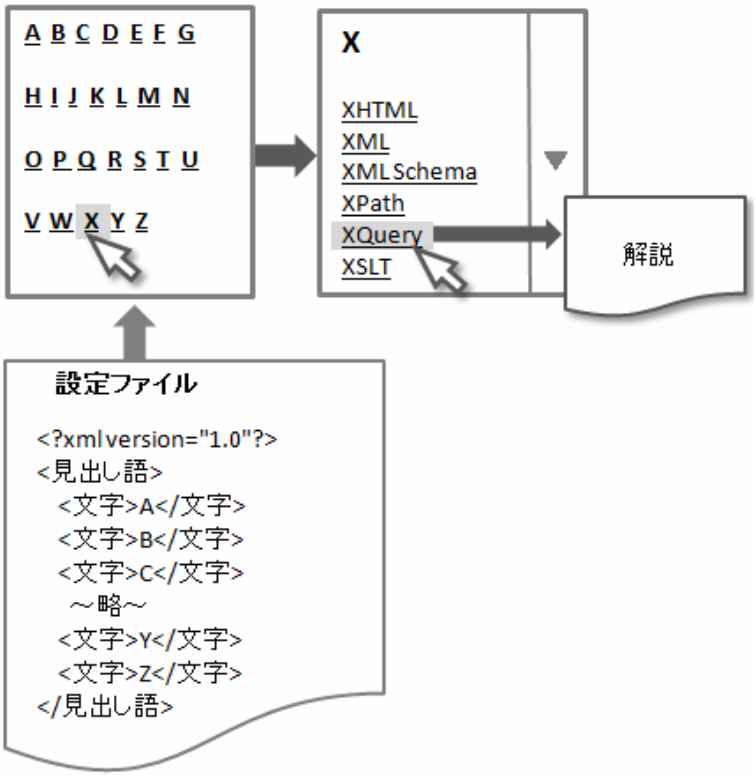
設定ファイルは，その構造を生かして，しばしば木構造のナビゲーションに用いられる。次ページ●図 5-23 のような，構造化メニューの選択肢を表示するユーザインタフェースは，よく見られるもののひとつである。これは一種の絞り込み検索処理であり，任意のノードをクリックすると，次の階層のノード群が現れる。

また，たとえば，見出し語（五十音・アルファベット）を選択して詳細データを表示する処理では，見出し語を XML 形式の設定ファイルとして作成することがある（次ページ●図 5-24）。このような設定ファイルは，一度作成しておけば，同じ目的のアプリケーションに何度でも流用することができる。

●図 5-23 設定ファイルを用いて，木構造のナビゲーションを表現する



●図 5-24 見出し語選択に，設定ファイルを利用する。



レガシーデータの一部を利用した表引き

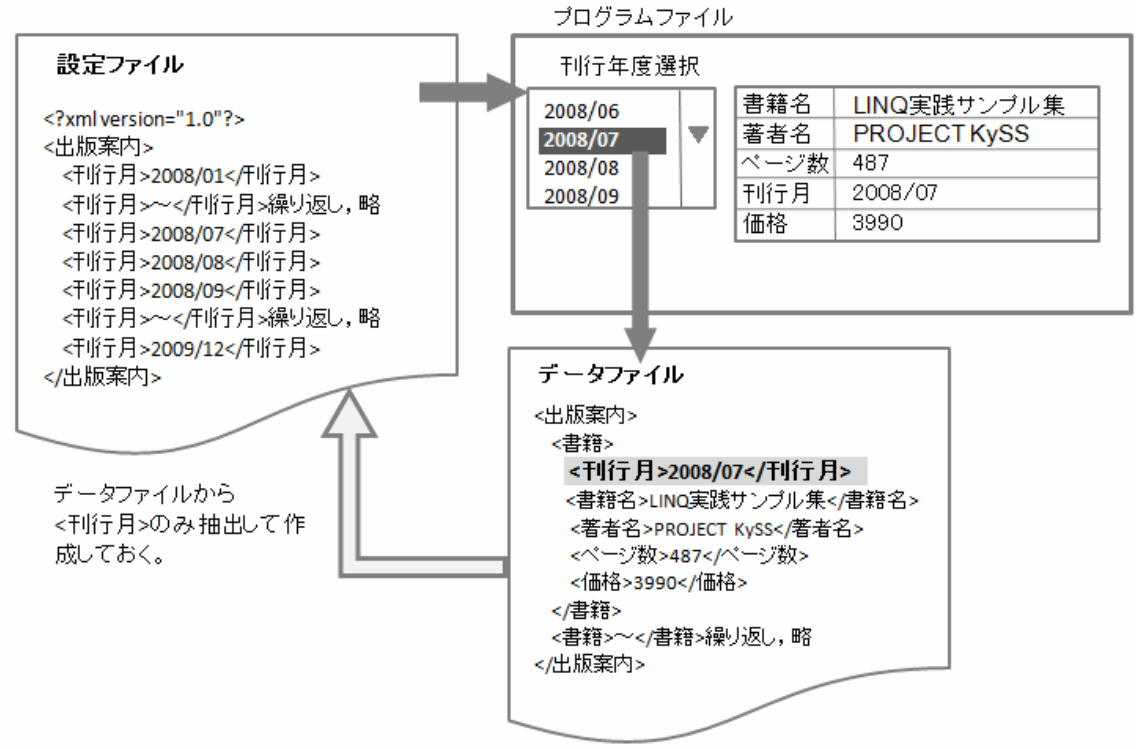
設定ファイルは，表引きを実装する際にも用いられる。主キー相当データを選択して詳細データを表示する処理では，詳細データの中の主キー相当データを別ファイルとして作成し，表引き用の設定ファイルとして用いる。データファイルはプログラムコード中で直接指定して読み込む。

この方法では，設定ファイル中のすべてのデータがデータファイル中に含まれているので，別途設定

ファイルを作成する手間がかかることで非効率的であるように思われる。だが，データファイルのノード数が多ければ，何件かの表引き用のリストを表示するために，データファイルの当該ノードをすべて走査する必要が生じてしまう。その方がむしろ非効率的である。設定ファイルとデータファイル中のデータが重複してもよいので，設定ファイルを用いるとよい（●図 5-25）。設定ファイルは，XSLT，DOM，LINQ 等を用いて，自動的に生成する。

この方法は，Web アプリケーションの選択肢の表示にも用いることができる。選択肢の項目を画像やボタンで作ってしまうと，選択肢が追加あるいは変更される度に，変更の手間が生じる。かといって，プログラムコード中に，DropDownList や ListBox の選択項目を直接記入していると，プログラムコードの修正が生じる。このような問題を解決するには，設定ファイルを作成し，読み込んで利用するとよい（※12）。

●図 5-25 データファイルのノード数が多ければ，表引き用のリストは設定ファイルとして別個に作成する



＊12：XAML のような XML 形式のベクターグラフィクス仕様を用いて，設定ファイルのテキストを画像の一部に利用する方法も考えられる。

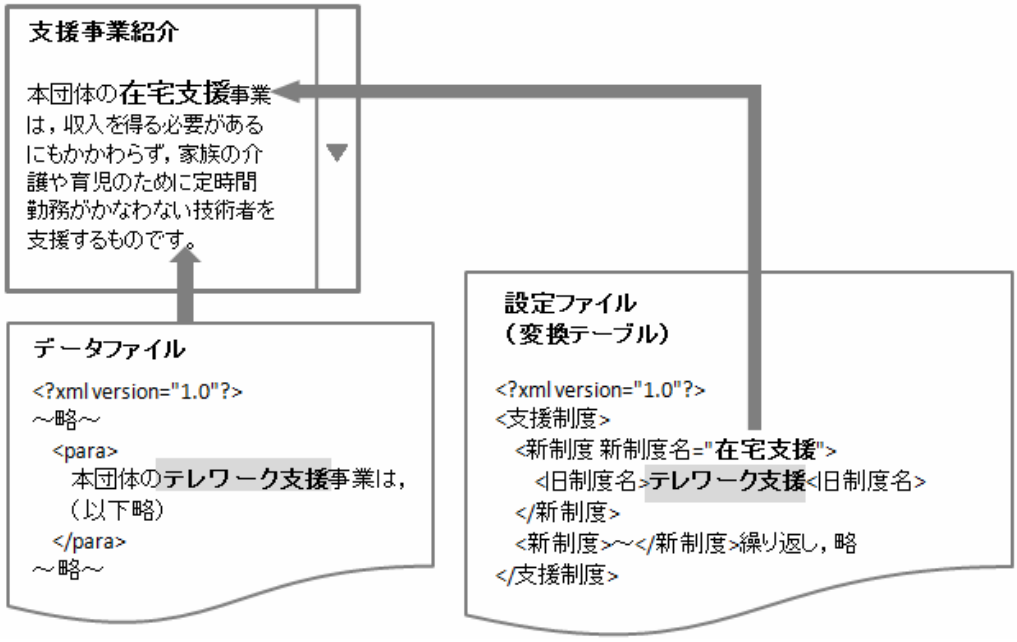
5.5.3 設定ファイルによるデザインの制御

変換テーブルの XML 化

XML 設計にあたっては，社会変化を予測しなければならないが（第 4 章「社会情勢の問題～時代の空気を読む～」参照），それが困難なケースもある。たとえば，市町村合併後の町名を予測することは難しい。しかしながら，物品を紹介するアプリケーションでは，できるだけ早く決定した町名データが必要である。あるいは，カタカナ語の表記について国語審議委員会の決定を予測することは難しいが，Web マニュアルの表記は逐次変更したい，といったケースを想定することができる。

それならば，応急処置として，一時的に変換テーブルをXML形式の設定ファイルとして作成し，アプリケーションから利用するのも一方法である（●図5-26）。その役割は，DTDにおける実体参照（第2章参照）に近い。

●図5-26 設定ファイルに変換テーブルのような役割を持たせる例

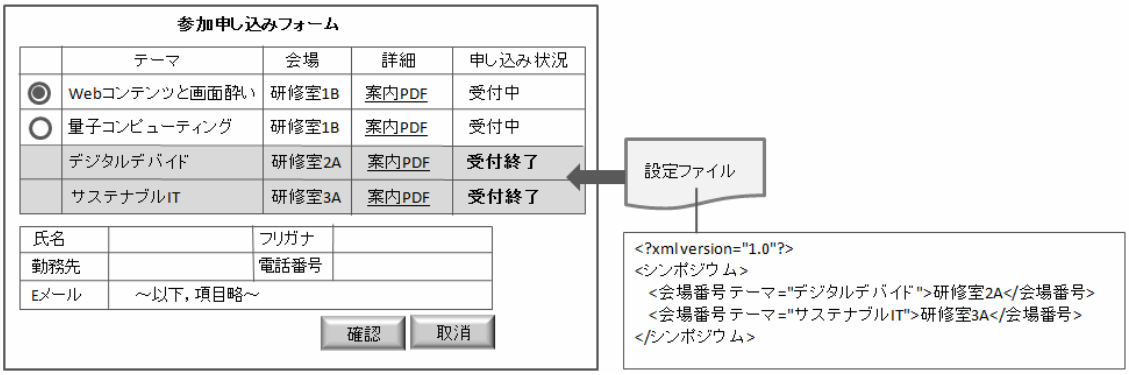


手動によるデータ変更を表示に反映

設定ファイルは，一時的にWebページ上のテキストデータを変更するために，用いることもできる。

たとえば，複数の講座の受講申し込み処理をXMLアプリケーションとして実装したとする。こういった処理では，受講希望人数が定員に達すると，ただちに受付を中止しなければならない。そこで，講座内容のデータファイルとは別に，会場番号や口座番号といった一意なデータを持つ設定ファイルを作成しておく。定員に達したら，設定ファイル中に，定員に達した講座名などのデータを追加する。そして，受講申し込みフォームでは，受付中の講座については選択可能とし，定員に達した講座については選択用のボタンを非表示として，「受付終了」というメッセージを表示する（●図5-27）。

●図5-27 設定ファイルにデータを手動で追加して，表示に反映させる例

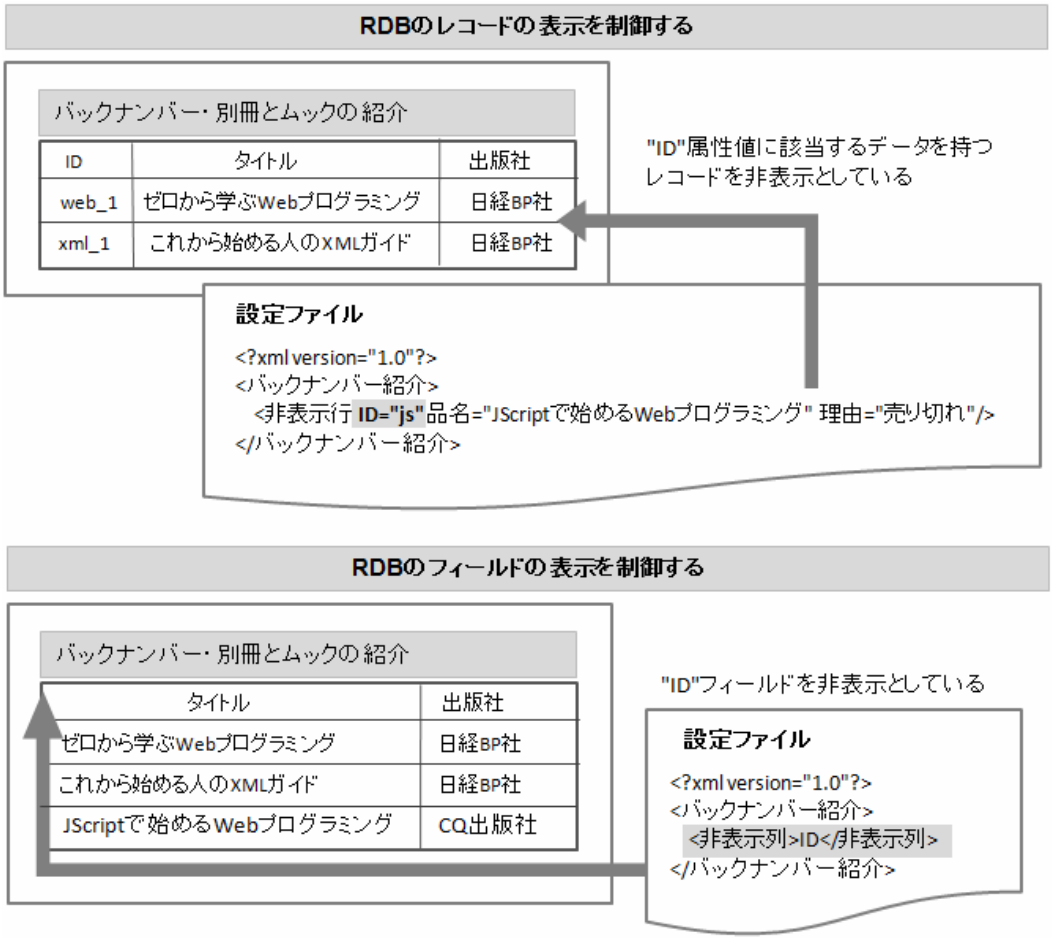


もし，これが全自動で受付を中止するのであれば，講座内容を記録したデータファイルの中に，受講人数の上限を属性値として追加しておき，カウントすればよい。PC および携帯端末からの申し込み内容をデータベースに登録し，申し込み者の人数が，受講人数の属性値に達した時点で，受付を中止すればよい（第 4 章●図 4-14 参照も参照）。だが，紙媒体の申込用紙や電話やファクシミリによる受付も許可する場合，自動化することはできない。Web フォームからの申し込み数と，それ以外の方法で受け付けた数を集計する作業が必要となるからである。こういった手作業が必要なケースでは，設定ファイルの利用が有効である。

RDB のレコードあるいはフィールドの表示の制御

先に「ミドルウェア導入の判断（●図 5-3」で述べたように，ミドルウェアを XML データベースとしてではなく，通常の RDB として用いて，XML 文書ファイルと連携する方法がある。その場合，ミドルウェアのデータを XML 文書ファイルの管理に用いるのではなく，逆に，XML 文書ファイルを RDB の表示制御に用いることもできる。たとえば，XML 設定ファイルにより，RDB の任意のレコードあるいはフィールドの表示と非表示を制御することができる（●図 5-28）。

●図 5-28 XML 設定ファイルにより，RDB の表示を制御する例



具体的には，まず，RDB 中の一意なデータを属性値とする XML 設定ファイルを作成しておく。

RDB のデータをバインドしている表の特定セル内のデータが，XML 設定ファイル中の属性値と合致するならば，当該レコードを非表示にする，というプログラムコードを記述する。

また，RDBのフィールド名を属性値とするXML設定ファイルを作成しておく。RDB中のデータをバインドしている表のヘッダテキストが，XML 設定ファイル中の属性値と合致するならば，データバインド時に当該フィールドを非表示にする，というプログラムコードを記述する。

ただし，RDBに連番の主キーを付け，これをXML設定ファイルからの制御に利用すると，レコードが追加あるいは削除された場合，目的通りの結果が得られなくなる恐れがあるので，RDB のデータ設計時には十分に注意する必要がある。

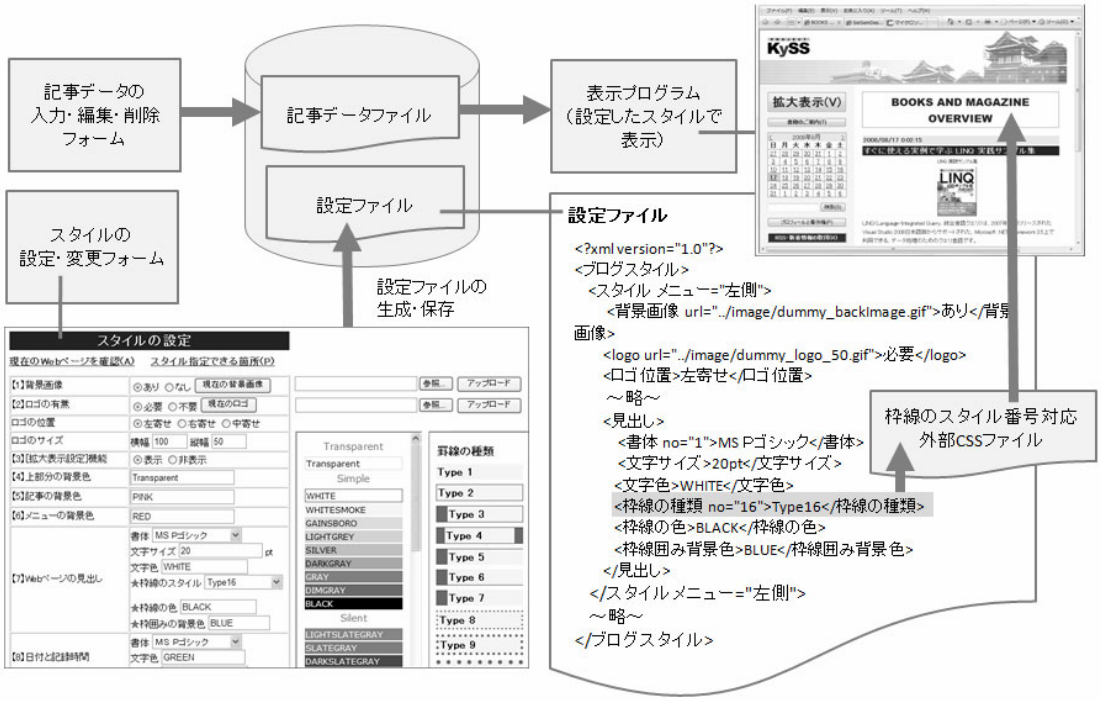
1ユーザ1設定のカスタマイズ

XML 設定ファイルは，Web アプリケーションのヴィジュアル･デザインを制御するために用いることもできる。

たとえば，設定ファイル中に，CSS のプロパティに対応するデータを要素の内容として記録しておく。これをプログラムファイルに読み込み，XML データファイルのデータを表示する際に適用して，表示を変えるといったものである。

この方法は，CMS ツールやブログツールに応用することができる（●図 5-29）。デザイン設定用のファイルを，本文データおよびコメントデータのデータファイルと分けることにより，プログラムロジックのコードと，ヴィジュアル･デザインを適用するコードを切り離すことができる。ユーザの利便性が増すだけでなく，プログラムのメンテナンスも容易になる。

●図 5-29 ブログツールにおける XML 設定ファイルの利用例



また，Web アプリケーションに限らず，ローカルアプリケーション，モバイルアプリケーションにおいて，画面上に表示するメッセージや画像ファイル名，再生する音声ファイル名を，XML 設定ファイルとして記録しておくことにより，1ユーザ1設定のカスタマイズが可能となる。

5.6 複数ファイル間における整合性の確認

5.6.1 複数ファイルの利点と問題点

管理ファイルとデータファイルの処理

複数ファイル構成とするならば，1 個の XML 文書内でのノード間の整合性だけでなく，ファイル間の整合性も検討しなければならない（1 文書内でのノードの整合性については，5.3 参照）。

複数ファイル構成であっても，複数のデータファイルのみによる構成であれば，ファイル名の変更等の際に人為的ミスが入り込まない限り，ファイル間に不整合の生じる可能性は極めて低い。しかしながら，管理ファイルを用いる構成では，相互の各ファイルの変更･生成･追加･削除によって，不整合の生じる恐れがある。

そこで，ファイルの変更，生成･追加，削除処理（●表 5-8）を想定して，ファイル間の整合性を検討しておく必要がある。

●表 5-8 管理ファイルおよびデータファイルの処理

処理	処理内容	
	管理ファイル	データファイル
変更	参照先データを変更する。	保存先やファイル名を変更する。
	参照するノードの順序を入れ替える。	必要であれば，ファイル名の連番部分を変更する。
生成･追加	末尾に参照先データを追加する。	新規ファイルを生成する。
	任意の位置に参照先データを追加する。	新規ファイルを生成する。
削除	管理ファイルを削除する。	データファイルを削除する。
	管理ファイルを削除する。	データファイルは維持する。
	管理ファイルは維持する。	データファイルを削除する。
	管理ファイルから，参照先データを削除する。	参照先のデータファイルを削除する。
	管理ファイルから，参照先データを削除する。	データファイルは維持する。
	管理ファイル中の参照先データは維持する。	参照先のデータファイルを削除する。

ファイルの変更と整合性

●表 5-8 に挙げたとおり，データファイルあるいは管理ファイルの編集において，データファイルの保存先やファイル名を変更するのであれば，管理ファイル中のデータも変更する必要がある。パスについては，全データファイルで共通ならば，管理ファイル中のルート要素の属性値として記録しておいてもよい（前掲●図 5-18 参照）。

また，管理ファイル中のノードを入れ替えると，そのコンテキスト･ポジションをデータファイル名としている場合には不整合が生じるので，データファイル名を変更しなければならない。（「管理ファイルを併用する場合(6)」●図 5-19 参照）。

ファイルの新規生成･ 追加と整合性

ファイルの追加においても，編集と同様の不整合が生じる恐れがある。

管理ファイル側から URI 指定によりデータファイルを参照している場合は，データファイルを新規生成して追加したならば，管理ファイル中の任意の位置にデータファイルのリストを追加する必要がある。

ただし，データファイルに連番を含むファイル名を付けているケースでは，管理ファイル中の末尾ではない任意の位置に参照先を追加すると，コンテキスト･ポジションが変わるために不整合が生じることがある。基本的には，末尾に追加すべきである。逆に，末尾ではなく先頭に追加していく方法もある（次章で解説，●図 6-17 参照）。

ファイルの削除と整合性

最も不整合を引き起こしやすいのは，ファイルの削除である。

データファイルを削除すると，管理ファイルから参照先データを削除しなければ，不整合となる。データファイルが HTML ファイルであればデッドリンクで済むが，プログラムファイルから XML 文書ファイルを読み込み処理していると，アプリケーションエラーになりかねない。ただし，管理ファイルを削除し，データファイルを維持する場合は，アプリケーションと管理ファイルの間の整合性にのみ注意すればよい（前掲●図 5-12 参照）（※13）。

データファイルに連番を含むファイル名を付けて，管理ファイルから参照しているケースでは，管理ファイル中の要素ごと削除すると，前項の新規生成･追加の場合と同様の不整合が生じる。

＊13：公開 Web アプリケーションの場合は，データファイルが拡張子.htm の XHTML であると，サーバに残しておく限り，他サイトからのリンクによって閲覧可能であり，ロボット避けを追加しないならば，検索エンジンのキャッシュに残る可能性がある。

CRUD のタイミング

以上のような変更･新規生成･追加，削除処理は，実行するタイミングも重要である。

ファイルの変更と追加においては，基本的に，管理ファイルよりもデータファイルの変更および新規生成･追加を優先する。ファイルの削除においては，逆に，データファイルよりも管理ファイル中のデータの削除を優先する。

なお，当然のことながら，ファイルを削除した後で，当該ファイル内のノードの編集･追加･削除を行うことができないのは，言うまでもない。

例外的なケースではあるが，空のデータファイルあるいは，ルート要素の中が空要素のみのデータファイルの扱いには注意しなければならない。XML 文書生成処理では，まずルート要素のみの木を生成することがある。プログラムによっては，追加するデータがなければ，ルート要素のみで内容が空のデータファイルが生成される。

ところが，管理ファイル内の参照先データの数と，空のデータファイルの数が等しいので，両者が対応していると勘違いしてしまう。データの内容が伴っていなくても，1 対 1 対応の不整合は生じていないであろうと判断してしまう恐れがある。

5.6.2 単一ファイルと複数ファイルの使い分け

以上を踏まえたうえで，単一の XML データファイルと，複数ファイルによる構成の，どちらを採用するか決定しなければならない。もちろん，単一ファイル，複数ファイル構成ともに，利点と問題点がある（●

表 5-9)。このうち，特に注意したいのは，管理(更新方法)である。たとえば，社員プロフィールの XML 文書ファイルを社員単位で分割しておくと，自身のデータのみ更新すればよく，上書き問題を回避することができる。複数ファイルに分割しておけば，編集･追加･削除処理を行うためのアプリケーションを開発する際，ユーザ認証によるアクセス制限を考えることができる(※14)。

●表 5-9 単一ファイル，複数ファイル構成の特長

	単一ファイル	複数ファイル
RDBとの連携	ノード単位で連携する。主に定型繰り返しのデータと対応させることになるため，RDBとの連携の意義は薄いことがある。	ファイル単位で連携する。各ファイルの構造は不定形でもよく，完全に異なってもよい。
処理速度	特に処理にDOMを用いる場合，巨大なツリーを展開すると，初回読み込み時の処理速度がメモリに依存する	管理ファイルの設計が適切で，必要なファイルだけを読み込むのであれば，高速に処理することができる。
ノード数	最大同時アクセス数や処理の内容を考慮しないならば，(DOMを用いた処理の場合)数GBのメモリで100万ノード程度を上限とする。	ファイル単位の処理であれば，特に上限を設定する必要はない。一千万単位以上のファイル数でも，所要時間を無視するならば処理自体は可能である。
ファイル操作	ファイルのオープン/クローズが一度だけで済む。n個のファイルのオープン/走査/クローズを繰り返すのではなく，1個のファイルをオープンし，n件以下のデータを走査してクローズすることができる。	管理ファイルなしで多数のファイルを何度も読み込む処理には適していない。管理ファイルを用いた場合でも，処理によっては，ファイル数分のオープン/クローズが発生することもある。
設計	比較的容易である。	階層が深い不定形文書については，分割設計が難しい。アプリケーションの設計においても，連携処理が複雑になりがちである。
作成	オペレータのスキルによるところが大きい。階層が深く不定形であると，ミスが生じやすい。別途，XMLデータ生成アプリケーションを開発する方法もある。	簡易な構造であれば，作成の分業が可能である。単一ファイルを分割する，XMLデータ生成アプリケーションを開発する方法もある。
管理(更新方法)	管理するファイルが1個でよいので，一人の担当者が管理する場合は，データの更新作業やファイルの管理が容易である。	ファイルが適切に分かれていれば，変更の必要なファイルのみ更新すればよいため，メンテナンスが容易である。上書き問題も回避しやすい。他方で，データ更新を自動化せず手作業で行う場合，管理ファイル側の関連データの修正漏れが発生する恐れがある。

単一ファイルと複数ファイルの使い分けについては，以上の事項について検討し，RDB との連携の有無，データ件数，ノード数，XML 文書を蓄積するサーバ環境，XML プロセッサによる XML 走査上の優位性，データの更新頻度，更新方法などから総合的に判断することが望ましい。

前節と本節で述べた確認作業を，今問題が発生するわけではないから，などと軽く捉えてはならない。かといって，検討に時間をかけすぎると，後工程に影響する。XML 設計者には，段取り八分を心得つつ，検討段階の適切な時点で決定を下すバランス感覚が必要である。

※14：少人数のユーザが閉じられたアプリケーションを利用する場合で，且つユーザ全員に高いモラルがある場合は，簡易な方法として，ログイン用の ID やパスワードを，ミドルウェアを用いずネイティブ XML 文書として管理することもできる。ただし，プログラムで人工キーを発行して ID に連結したものを，頻繁に変更しながら用いるなどの安全策は必要であろう。

5.7 タグ名の確定

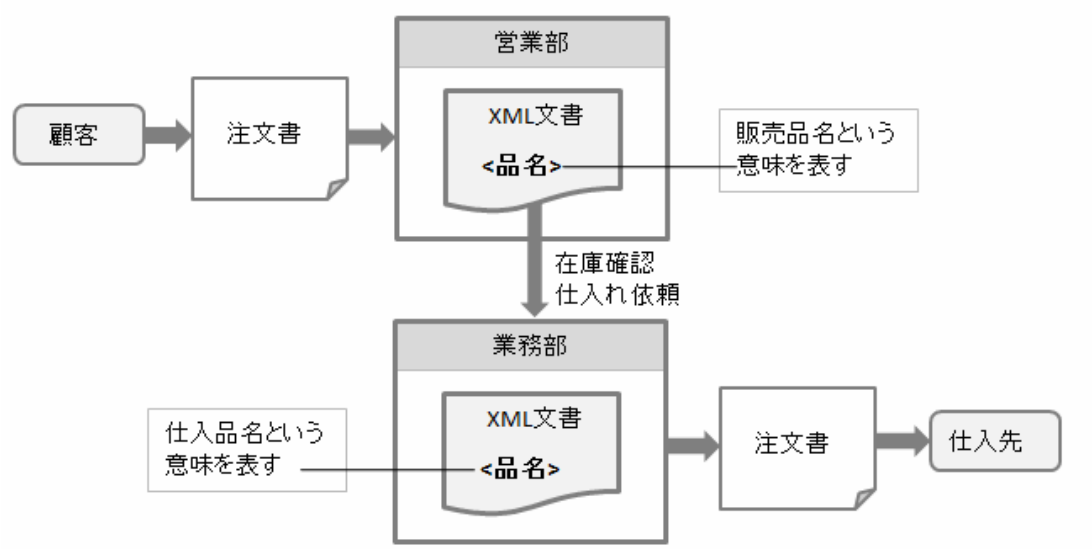
5.7.1 タグ名の重複可能性の確認

この段階まで，ルート要素以外については，暫定的な仮のタグ名で作業を進めてきている。ここで，そのままのタグ名とするのか，変更するのかを検討し，タグ名を確定する。

まず，異なる意味を持つデータに対して同一の要素名や属性名が使われる可能性について検討しておこう。

たとえば，レガシーXML データを複数の部門や工程で利用する可能性があるならば，営業部が顧客から受け取る「注文書」と，業務部(調達部) が仕入れ先に発行する「注文書」に，同じ<品名>要素があっても，メタデータの持つ意味は異なる。一方の意味は販売品名であり，もう一方の意味は，仕入品名である (●図 5-30) 。

●図 5-30 同じ要素名であっても，文書利用者の立場によって意味が異なることがある。



この問題の対処方法としては，主に 4 つが考えられる。

1 つ目は，レガシーXML データの要素名は<品名>のまま据え置き，利用時に，<注文書 管理="営業">，<注文書 管理="業務">などのように，ルート要素に管理部署を追加する方法である。

2 つ目は，レガシーXML データの要素名は<品名>のまま据え置き，利用時に，<仕入品名><販売品名>などのように，異なる要素名に変換して処理する方法である。ただし，この方法では，当該要素に属性が付属している場合，要素名だけでなく，属性の従属先を再確認する必要がある。

3 つ目は，利用時にも同じ<品名>要素を用いるが，各工程･各部門別の名前空間を用いて識別する方法である。ただし，予定運用期間の間に，組織改編や統廃合の可能性が高ければ，難しいかもしれない。

4 つ目は，同一の<商品コード>といった要素を追加して，統一をはかる方法である。

また，<年月日>や<番号>のような，複数の書類に共通して使われる名前について，使い分けるべきか，統一すべきかも考える必要がある。安易に定義すると，受注年月日･納品年月日，注文番号･商品番号･伝票番号などが混在することになるからである。

さらに，定義の単純ミスにも注意する。同一部署内の一連の文書において，データの意味と内容が同一であるにも関わらず，ある文書では<型番>，別の文書では<品番>といった要素名を付けてしまうと，連携処理の際に問題が発生する。単独設計で設計書のレビュアーがいない場合，こういった「生じるはずのない，単純ミス」の発生する余地があり，しかも，単純なミスほど見逃されやすい。

膨大な文書が蓄積された後で誤りに気付くと，XML 文書はそのまま据え置いて，処理プログラムの変更によって対処せざるをえなくなるため，バグ発生のリスクが高まる。タグ名が重複する恐れがないか確認し，重複するのであれば対応方法を考えなければならない。

5.7.2 要素内容および属性値のルール作り

タグ名が決定したら，テキストノードに関するルールを作成しておく。こうした構造や名前以外の確認事項は，見落としがちであるが，重要である。

よくある例として，氏名の旧漢字と新漢字，婚姻などにより姓が変更となった場合の，旧姓と新しい姓の扱いといった問題がある。いずれかに統一する方法もあれば，氏名だけでなく，ID などの一意なデータも併用して管理する方法もある。

もし，DTP ソフトや PDF の既存テキストから，コピー＆ペーストによって XML 文書中のデータを作成するのならば，パスを記述する際のバックスラッシュや円記号等について，XML 設計者とオペレータの間で，刷り合わせとテストが必要である。なぜなら，多くの場合，両者の使っている OS が異なり，文字化けの入り込む余地があるからである。

テキストそのものの表現についても検討しておく必要がある。表記の規定は時代とともに変わる（※15）。その変化によって要素名や属性名が影響を受けることは望ましくない。その意味では，専門用語は，要素の内容あるいは属性値として扱う方がよいだろう。ただし，利用範囲が専門家に限定される XML 文書に限っては，ケース･バイ･ケースで検討する。たとえば，<既往歴>や<ナノテクノロジー>などを要素名として採用するかどうかといったことである。

※15：一般医学用語や外来語の言い換えについては，独立行政法人　国立国語研究所（http://www.kokken.go.jp/）の Web サイトを参照。これ以外にも，テクニカルコミュニケーター協会「カタカナ表記検討ワーキンググループ」などによる，外来語末尾の長音符の表記問題などにも注意する必要がある。

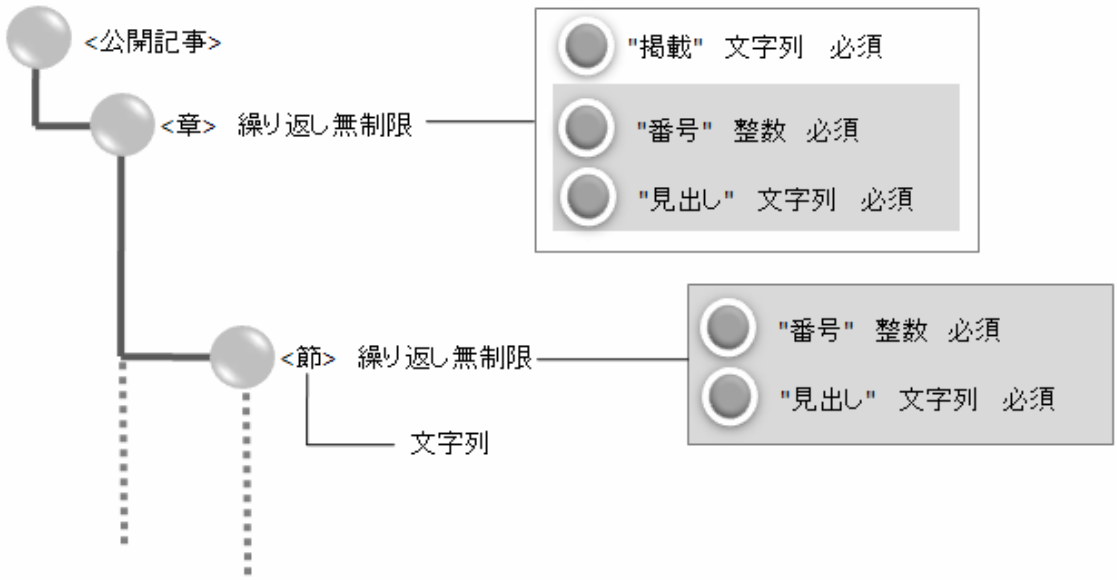
5.8 設計後の作業

5.8.1 仕様書の作成

設計が完了したら，その経過と決定事項を，必ず仕様書として作成する。仕様書には，データ定義仕様書，サンプル XML 文書ファイルを含めておく。

データ定義仕様書は，XML スキーマ文書を作成して代用してもよい。RDB 技術者が XML に取り組む場合は ER 図を用いるかもしれない。ただし，顧客や，XML 文書作成オペレータといった，XML にも RDB にも無関係な人々に理解してもらうには，木構造を絵で表現する方が伝わりやすい。たとえば，第 3 章掲載の●リスト 3-38 を絵で表すと，●図 5-31 のようになる。このような図であれば，実際，アプリケーション開発にかかわる Web デザイナや，プロジェクトを支える事務方にも，理解を得ることができる。また，これ以外にも，XML 文書の作成方法，データ更新方法，木構造を決定するにいたった経緯と，最終決定の理由を，すべて記録しておく。文字コードの記載も忘れないようにしたい。XML アプリケーション開発の際に，しばしば，出力形式の指定が必要となるからである。

●図 5-31 技術者以外に伝えるには，木構造を絵で表現すると理解してもらいやすい。



サンプル XML 文書ファイルには，最大限のデータを盛り込む。つまり，出現してもしなくてもよい要素については，必ず出現するものとして作成する。さらに，できれば，この後開発するデータ処理アプリケーションのテストに利用できるよう，複数のパターンのサンプル XML 文書ファイルを作成しておくことが望ましい。

もし，データ件数が少なく，短期間で廃棄する XML 文書であっても，サンプル XML 文書ファイルを設計書に代えることは，お勧めしない。できるだけ，設計内容は文書化しておいた方がよい。それは，後工程での伝達ミスを防ぐ役割を果たし，後進に経験を伝える資料ともなるからである。

5.8.2 実データの準備とタグ付け

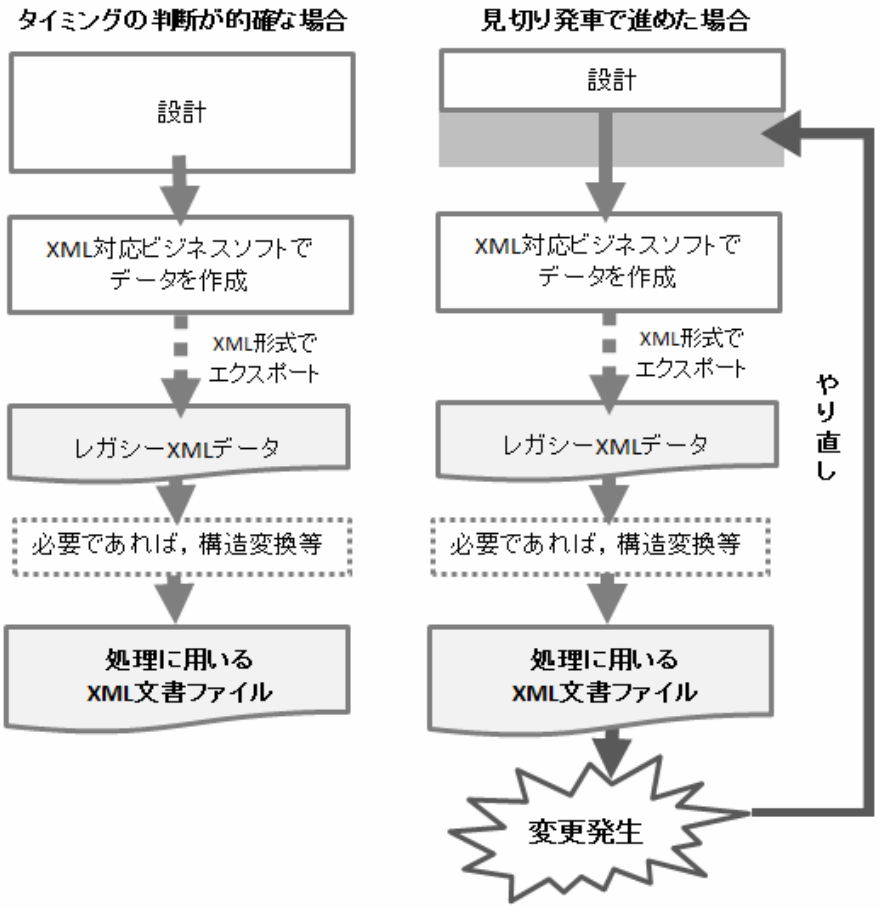
仕様書ができたら，これに基づいて XML 文書を作成することになる。

それには，レガシーXML データをそのまま処理対象文書として用いる方法もあれば，XSLT スタイルシートなどにより構造を変換する方法もある。レガシーXML データのもとになるデータは，オペレータが入力する必要があるが，レガシーXML データの生成や変換，つまりタグ付けは，手作業ではなく，XML 文書生成用アプリケーションを開発して実行する方がよい（次章で詳説）。

手作業であっても，XML 文書生成用アプリケーションを開発した場合であっても，入力ミスや操作ミスを防ぐために，データの作成手順は標準化しておく。レガシーXML データに誤りがあると，最終工程まで誤りが引き継がれてしまったり，誤った下流データがさらに上流工程にリサイクルされるからである。データの流用過程で不具合が生じると，原因を探るために多くの時間と手間を費やすことになりかねない。

特に，XML 対応ビジネスソフトで作成されたデータをもとにレガシーXML データを作成するケースでは，XML 化のタイミングが重要である。項目名やデータ型が曖昧な状態で XML 化すると，その後で，必ずといっていいほど変更が生じる。そこで，ビジネスソフトで作成されたデータを変更した上で，再度 XML 化するという二度手間が生じる。度重なる変更は，ミスを誘発する。どの時点のデータを採用するかという判断が，その後の作業効率に大きく影響するのである（●図 5-32）。

●図 5-32 XML 化のタイミングを正しく判断し，手間とミスを防ぐ



以上，本章では基本的な設計方法について述べたが（※16），実務では，XML データ生成アプリケーションやデータ処理アプリケーションの基本設計を視野に入れたうえで，木構造の設計を行うことが少なくない。次章では，検索用のネイティブ XML 文書設計をテーマに，より実践的な話をしよう。

※16：本章で述べた諸々の注意事項は，筆者が設計作業中に気付いた問題（設計後に問題を引き起こしたことは一度もない）や，起こりうるであろうから注意を喚起しておいた方がよいと考える問題，リメイク案件などで見聞きした問題を中心にまとめている。

第 6 章　XML 設計の実践

本章では，検索処理に用いるネイティブ XML 文書設計の一方法と，

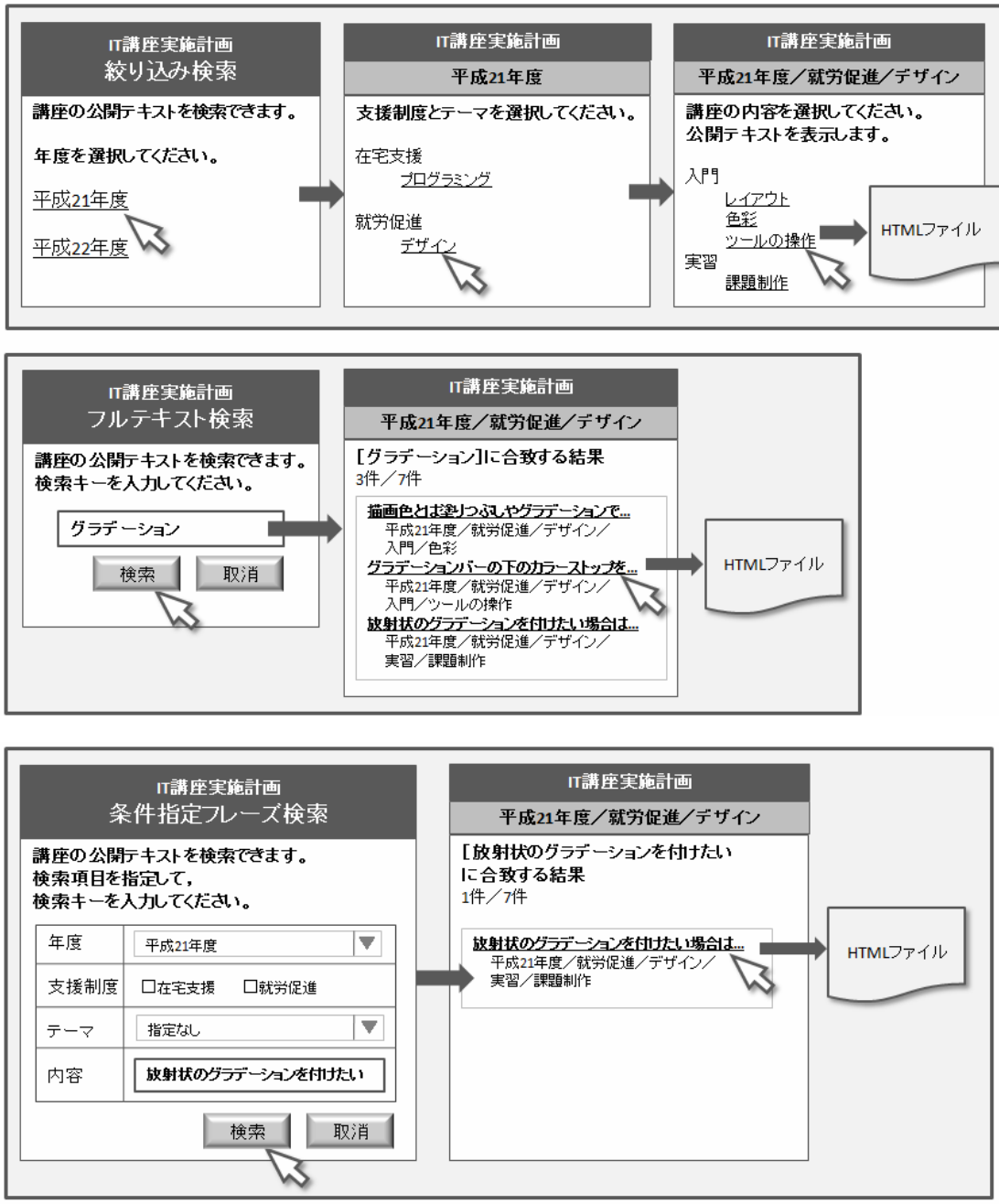
手順におけるいくつかの課題について取り上げる。

6.1 ネイティブ XML 文書の検索処理

6.1.1 検索処理における XML の利点

ネイティブ XML 文書は，情報開示を前提とする，データの書き込みを伴わない処理に用いると，大きな効果を見込むことができる。代表的な例としては，検索処理におけるデータベースとしての利用があげられる。(●図 6-1)。

●図 6-1 XML の特長を生かした検索処理。絞り込み検索，フルテキスト検索，条件指定フレーズ検索。



※この図は，Webアプリケーションとして実装した場合の画面遷移例である。
※図を分かりやすくするため，通常，検索アプリケーション画面にある，検索手順のヘルプ，1ページあたりの表示件数，ページング等は記載していない。

XML 文書は，SGML を母胎として誕生したという背景から（第 4 章参照），マニュアルや論文，ビジネス文書，書籍や記事といった長文のテキストも，容易に電子化することができる。それらの文書を対象とした，<章><節><項>と階層をたどるような絞り込み検索では，階層構造を利用すれば，目的のデータをピンポイントで特定しやすいという利点がある（第 5 章●図 5-23 も参照）。

絞り込み検索の結果は XSLT スタイルシートによって XHTML（あるいは HTML）として出力すれば，多彩なユーザ・インタフェース・デザインが可能となる（**※1**）。検索結果から，XHTML ファイルや PDF ファイル，動画コンテンツなどにリンクを張ることもできる。また，Microsoft Silverlight アプリケーションにおいても，XML 文書をデータベースとして活用することができ，いっそうリッチなインタフェース・デザインが可能になる。

※1：アクセシビリティも，Web ページのそれに準じる。文字サイズの拡大や，白黒反転表示は，適用する CSS を切り替えれば実装することができる。絞り込み検索ならば，検索キーに指定する文字入力が不要であるため，音声入力を伴わず，[Tab]キーと[Enter]キーだけで操作可能である。

6.1.2 データ作成への表計算ソフトの利用

検索処理のデータベースとして用いられるような XML 文書は，当然のことながら，ノード数が膨大であり，テキストエディタで記述できるような代物ではない。データ型やデータ長については，XML Schema などを用いて検証することはできても，長文テキストの誤字脱字まではチェックすることはできない。

そこで，データの入力・編集ミスを発生させない工夫が必要となる。現実的な方法のひとつは，新規にデータを入力するにせよ，既存データを活用するにせよ，オペレータが日常業務で使い慣れているソフトウェアを用いることである。昨今のビジネスソフトは XML 形式の入出力に対応しているので，何ら問題はない。

とりわけ有効なのは，表計算ソフトを用いる方法である。単純な表への入力であれば，ソフトウェアのバージョンの違いによる影響も受けることはないといっていい。運用期間中のデータの編集・追加・削除作業についても，操作に慣れていれば比較的短時間で済む。

リレーションを設定していない 1 枚の表なら一度に把握しやすく，視点を左右に移動するだけで，フィールド名と 1 件分のデータを把握できるという利点がある。これが XML 文書のソースであれば，視点を左右と同時に上下に移動する必要がある。構造が複雑であれば，現在のディスプレイでは一目で把握できるとは言い難い。W3C 仕様書の「XML documents should be human-legible and reasonably clear.」という目標を裏付けるセマンティクスやコンテキストの理解も，感覚器官から得られる情報あればこそである。

そこで本書では，レガシーXML データの作成に，ユーザの裾野の広い Microsoft Excel を用いるものとして，解説を進める。

Excel レガシーデータ（**※2**）から，レガシーXML データを作成すると，浅い定型のフラットな構造となる。だが，それでは整形式 XML 文書の衣を借りた表データに過ぎない。フラットな構造のまま蓄積・処理す

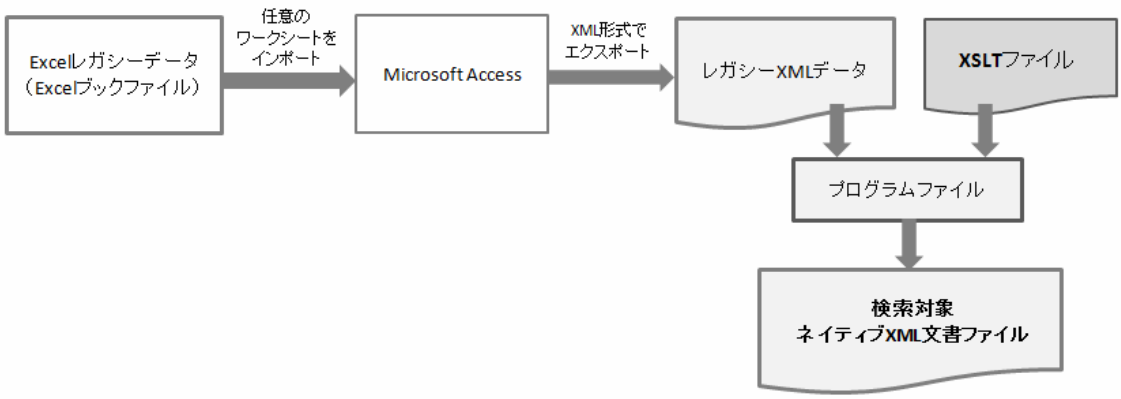
るのであれば，RDB の方が適している。このフラットなデータを，木構造化する必然性があればこそ，XML 化の意味がある。

そこで，次節からは，Excel レガシーデータから，レガシーXML データを作成する（※3）だけでなく，DOM を用いたプログラムや XSLT スタイルシート（※4）によって構造を変換し，検索対象 XML 文書ファイルを作成する手順を紹介していく（●図 6-2）（※5）。

Office のバージョンに関わらず，Excel レガシーデータを XML 化する分かりやすい方法は，Excel の任意のワークシートを Microsoft Access に読み込み，XML 形式でエクスポートするというものである。

●図 6-2 は，後述の●図 6-4 の Excel レガシーデータを Access 経由でエクスポートし，●リスト 6-1 を作成する手順の一部分である。

●図 6-2 Excel ワークシートから，検索処理対象とするネイティブ XML 文書を作成する，一方法。



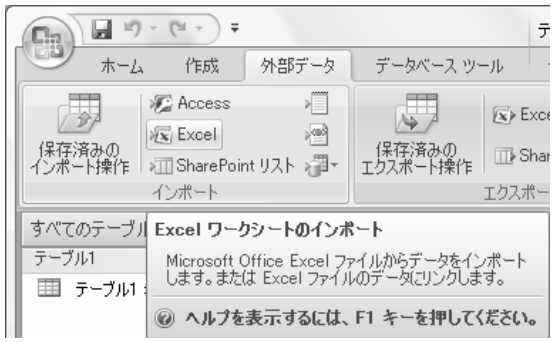
Microsoft Excel で入力されたデータ

	A	B	C	D	E	F	G	H	I	J	K
1	年度	支援制度	テーマ	内容	詳細	バス	コンテンツ	検索テキスト	所属1	氏名1	所属2
2	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_01.xml	女性技術者の会	古見 久美	
3	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_02.xml	シルバーギーク会	秋葉 定数	
4	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_03.xml	シルバーギーク会	木板 雄拓	女性技術者の
5	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	design_01.xml	XML株式会社	江楠 絵夢	
6	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	design_02.xml	XML株式会社	江楠 絵夢	
7	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	design_03.xml	XML株式会社	江楠 絵夢	夢織デザイン
8	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	design_04.xml	XML株式会社	江楠 絵夢	視覚スタジオ
9	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_01.xml	—	—	
10	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_02.xml	XML株式会社	須木間 定義 江楠 絵夢	DBユーザグル
11	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_03.xml	XML株式会社	須木間 定義 江楠 絵夢	視覚スタジオ
12											

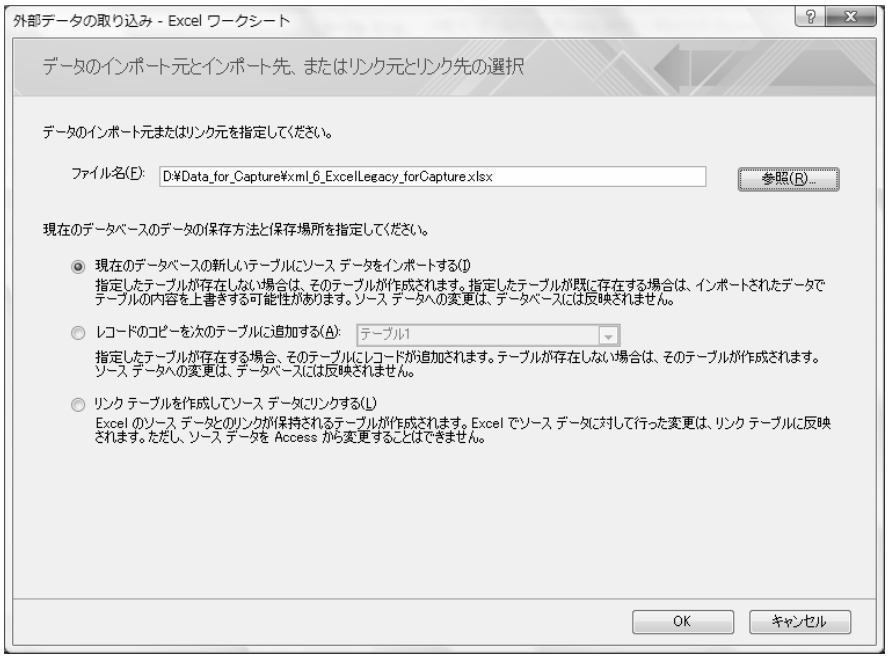
Microsoft Access の操作手順

①空のデータベースを新規作成。

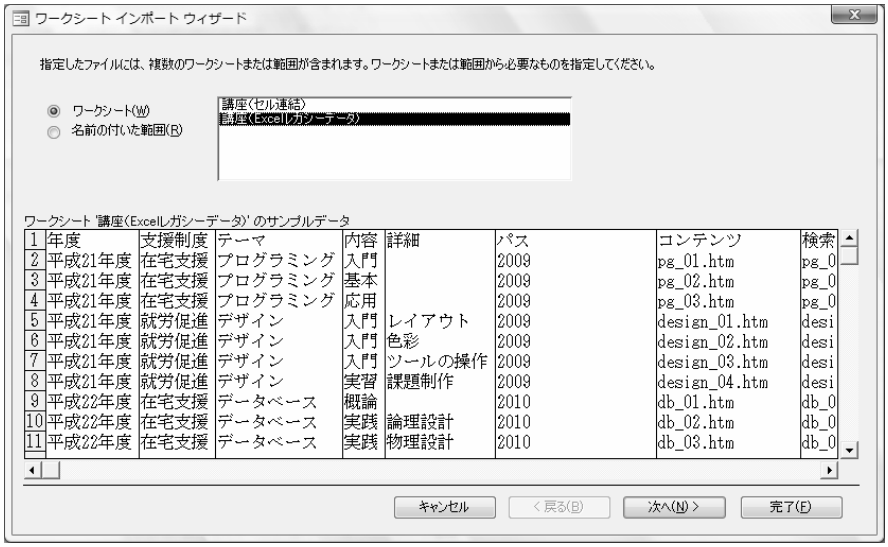
② [外部データ] タブ [Excel] 選択。



③［参照］クリック，インポートする Excel ファイルを指定して［OK］。

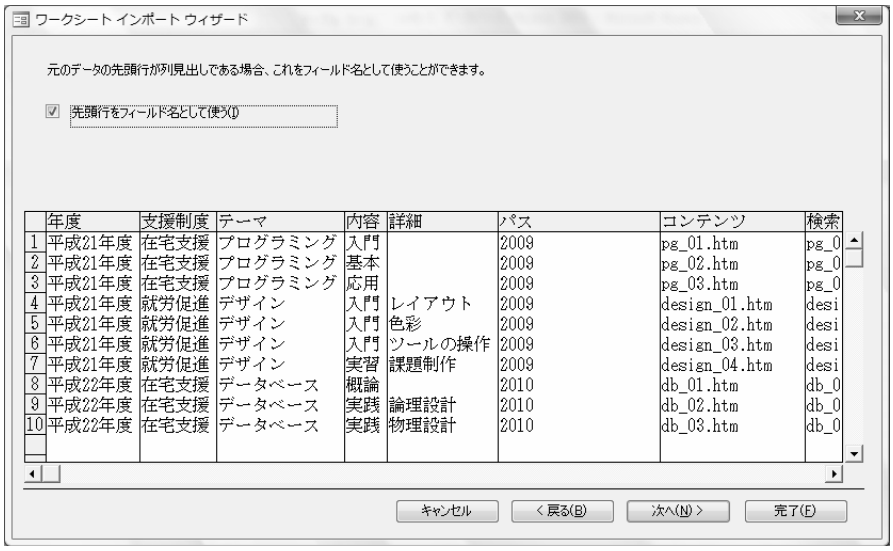


④ワークシート選択，［次へ］。



1	年度	支援制度	テーマ	内容	詳細	パス	コンテンツ	検索
2	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_0
3	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_0
4	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_0
5	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	desi
6	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	desi
7	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	desi
8	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	desi
9	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_0
10	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_0
11	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_0

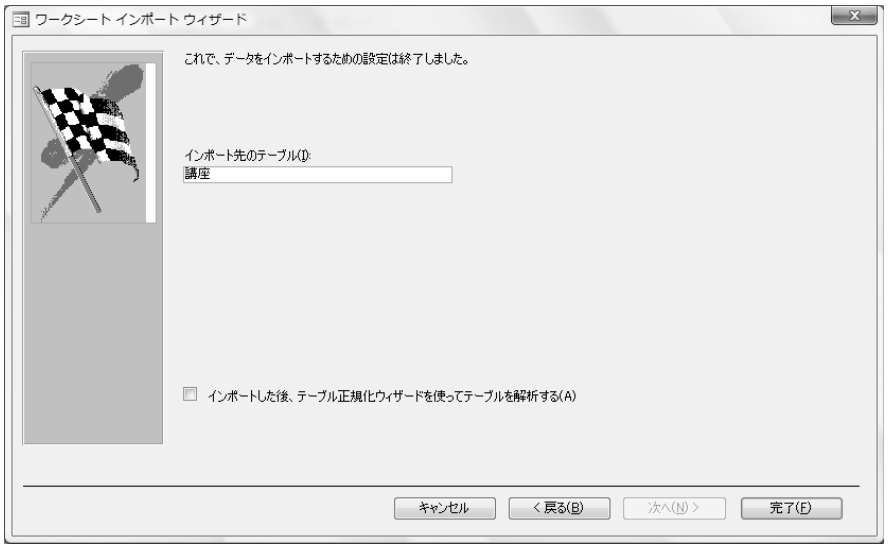
⑤［先頭行をフィールド名として使う(I)］にチェックを付けて，［次へ］さらに［次へ］。

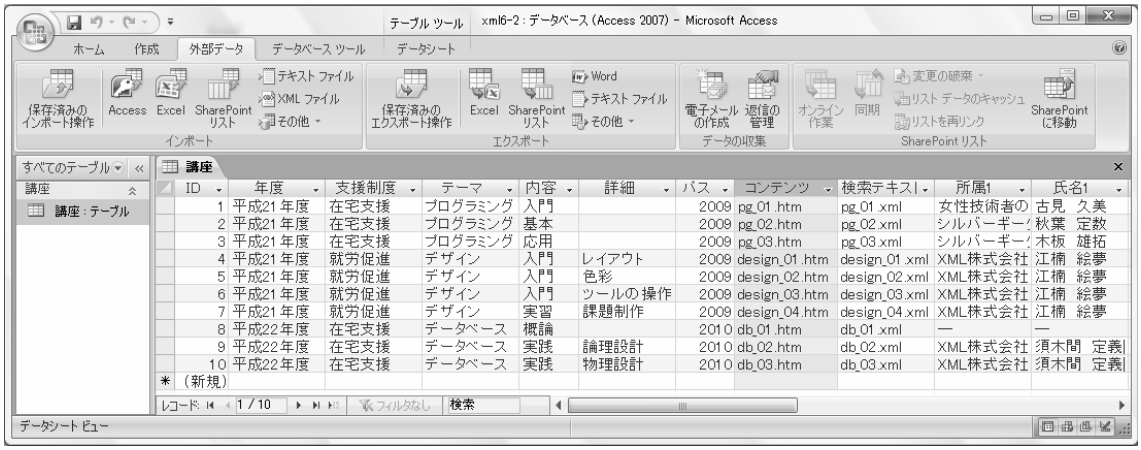


	年度	支援制度	テーマ	内容	詳細	パス	コンテンツ	検索
1	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_0
2	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_0
3	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_0
4	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	desi
5	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	desi
6	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	desi
7	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	desi
8	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_0
9	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_0
10	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_0

⑥［主キーを自動的に設定する(A)］にチェックがデフォルトの状態で，［次へ］。

⑦［インポート先のテーブル名］を入力して，［完了］［閉じる］。このテーブル名が，XML 形式でエクスポートした際に，ルート要素直下で繰り返す要素の名前となる。

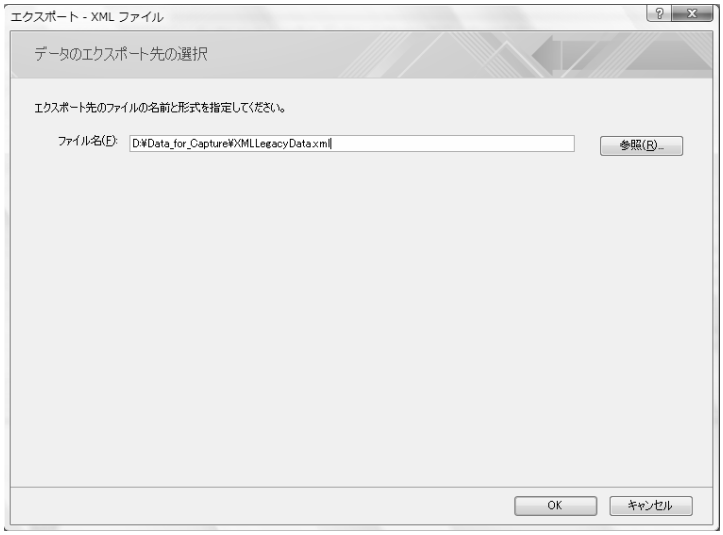




ID	年度	支援制度	テーマ	内容	詳細	パス	コンテンツ	検索テキスト	所属1	氏名1
1	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_01.xml	女性技術者の	古見 久美
2	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_02.xml	シルバーギー	秋葉 定数
3	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_03.xml	シルバーギー	木板 雄拓
4	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	design_01.xml	XML株式会社	江楠 絵夢
5	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	design_02.xml	XML株式会社	江楠 絵夢
6	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	design_03.xml	XML株式会社	江楠 絵夢
7	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	design_04.xml	XML株式会社	江楠 絵夢
8	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_01.xml	—	—
9	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_02.xml	XML株式会社	須木間 定義
10	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_03.xml	XML株式会社	須木間 定義
＊	(新規)									

⑧［エクスポート］［その他］［XML ファイル］選択。

⑨エクスポート先と XML ファイル名を指定。



⑩［データ (XML) (D)］のみにチェックして［OK］。（デフォルトでは［データのスキーマ (XSD) (S)］にもチェックが付いている），［閉じる］。

中小規模案件であれば，このような Excel レガシーデータの準備と編集作業，および XML レガシーデータの作成は人手によって行い，XML レガシーデータからの検索対象 XML 文書の生成については，アプリケーションを開発して自動化するとよい。その方が，後々の更新作業が容易になる。なにしろ Excel 上で編集･追加･削除を行って XML 形式でエクスポートすれば，後は検索対象文書生成アプリケーションを実行するだけで済む。自動であるから，後述する問題（6.5 設計後の工程に影響する問題）が発生しない限り，ミスの生じる余地がなく，しかも速い。

更新の手順はといえば，生成した最新の XML 文書ファイルを，運用中の XML 文書ファイルと差し替えるだけである。ファイルサイズやユーザ層にもよるが，アクセスのない時間帯を事前に調査しておいて更新作業を行えば，Web サーバの予告停止も不要である。

当然のことながら，XML 生成アプリケーションの開発には多少の工期が必要になる。だが，運用期間が長期に及び，頻繁にデータを更新するのであれば，更新の都度手作業でデータを追加するよりも，XML 生成アプリケーションを開発する方が，結果的に作業時間を大幅に短縮することができる。これは，設備の初期投資費用と運用経費の関係に似ている。

※2：ここでいう「Excel レガシーデータ」とは，いわゆる「Excel レガシー（含マクロ）」のことではなく，Excel で作成された始原データという意味である。

※3：本章で解説する方法は，操作手順の違いや機能の制限はあるが，Microsoft Office System2003 以降で可能である。Excel では，雛型となる XML 文書や XML Schema ファイルを読み込んでデータを追加し，XML 形式でエクスポートすることによって，XML 文書を作成することもできるが，異なる階層にある同一要素名の扱いなどに注意が必要となる。Excel2008 では，Excel ブックを Office Open XML ドキュメントパーツに分解し，処理対象ノードを抽出してツリーを再構築することもできる。

※4：本章で変換に用いる XSLT スタイルシートの仕様のバージョンは，XSLT1.0 である。XSLT の概要と文法については，第 3 章参照。

※5：以降で述べることは，XML 黎明期に，筆者が企画設計を手がけ，長年の運用期間中一度のトラブルも生じなかった方法――ただし，実際のデータは，テーマも内容も本章のサンプル・データとは全く異なり，複雑な構造でノード数も膨大であった――に基づいている。そのため，XML データの生成および検索処理の実装方法については，XML 黎明期からの技術である DOM と XSLT を前提とした記述となっているが，LINQ 等で代替えする場合であっても，設計者の思考方法は同じである。コードの表面を追うのではなく，検索処理対象とするネイティブ XML 文書設計の過程を読み取ってほしい。なお，LINQ を用いた XML 文書の構造変換については，オンライン連載記事中で述べる予定。掲載情報については筆者サイトを参照。

6.1.3 XML 化に適した Excel レガシーデータの作成

表のレイアウトとフィールド名

前掲●図 6-2 の手順により作成したレガシーXML データは，フラットな構造となる。これを検索対象文書として適切な階層構造に変換するには，次のように，あらかじめ Excel ワークシートのレイアウトを編集したうえで，XML 化するとよい。

ここでは，●図 6-3 のような，「IT 講座の実施計画要綱」の基本データを例に解説する（※6）。ワークシート上の各フィールドは，「年度＞支援制度＞テーマ＞内容」という階層関係にあると考えてほしい。講習内容は HTML ページとして作成して Web 上で公開するものとし，リンク先となる HTML ファイルのパスを「コンテンツ」列に記入している。この表は，XML 化を目的としないのであれば，可も不可もない，ごくありふれたものである。

しかしながら，このワークシートを XML 化のための Excel レガシーデータとして用いるには，修正すべき点がいくつかある。XML 化しやすい形に成形したのが，●図 6-4 である。

●図 6-3 XML 化は考慮せず入力された一般的な Excel ワークシート。

	A	B	C	D	E	F	G	H
1	年度	支援制度	テーマ	内容		コンテンツ	講師	
2							所属	氏名
3	平成21年度	在宅支援	プログラミング	入門		2009/pg_01.htm	女性技術者の会	古見　久美
4				基本		2009/pg_02.htm	シルバーギーク会	秋葉　定数
5				応用		2009/pg_03.htm	シルバーギーク会	木板　雄拓
6							女性技術者の会	半田　富士子，半田　妹子
7		就労促進	デザイン	入門	レイアウト	2009/design_01.htm	XML株式会社	江楠　絵夢
8					色彩	2009/design_02.htm	XML株式会社	江楠　絵夢
9					ツールの操作	2009/design_03.htm	XML株式会社	江楠　絵夢
10							夢織デザイン	阿戸　美子
11				実習	課題制作	2009/design_04.htm	XML株式会社	江楠　絵夢
12							視覚スタジオ	志井　紗歩
13							夢織デザイン	阿戸　美子
14	平成22年度	在宅支援	データベース	概論		2010/db_01.htm		
15				実践	論理設計	2010/db_02.htm	XML株式会社	須木間　定義，江楠　絵夢
16							DBユーザグループ	江須久　絵瑠
17					物理設計	2010/db_03.htm	XML株式会社	須木間　定義，江楠　絵夢
18							視覚スタジオ	林　功

●図 6-4 XML 化に適した Excel ワークシート。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	年度	支援制度	テーマ	内容	詳細	パス	コンテンツ	検索テキスト	所属1	氏名1	所属2	氏名2	所属3	氏名3
2	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_01.xml	女性技術者の会	古見　久美				
3	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_02.xml	シルバーギーク会	秋葉　定数				
4	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_03.xml	シルバーギーク会	木板　雄拓	女性技術者の会	半田　富士子|半田　妹子		
5	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	design_01.xml	XML株式会社	江楠　絵夢				
6	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	design_02.xml	XML株式会社	江楠　絵夢				
7	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	design_03.xml	XML株式会社	江楠　絵夢	夢織デザイン	阿戸　美子		
8	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	design_04.xml	XML株式会社	江楠　絵夢	視覚スタジオ	志井　紗歩	夢織デザイン	阿戸　美子
9	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_01.xml	—	—				
10	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_02.xml	XML株式会社	須木間　定義|江楠　絵夢	DBユーザグループ	江須久　絵瑠		
11	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_03.xml	XML株式会社	須木間　定義|江楠　絵夢	視覚スタジオ	林　功		

必ず1回出現

データがなくても，空要素を生成しない

テキスト検索，あるいはタグ名指定検索に利用する，本文データを記録したXMLファイル名

データがなくても，「—」を内容とする要素を生成する

所属と講師については，さらに増える可能性に対応するため，最後に入力している

必ず1回出現

まず，セルの連結を解除し，同一データも各セルにコピーしている。ヒトが理解するだけではなく，コンピュータに処理させるのであるから，必要なデータは逐一入力しておいた方が処理しやすい。

パスとコンテンツを F 列と G 列に分けているのは，コピー＆ペーストの効率向上と入力ミスを減らす目的からである。ここでは，リンク先の HTML ページへのパスを記入しているが，もちろん，PDF，LZH，ZIP 形式のファイル名を記入するフィールドを追加しておくことにより，対象ファイルをダウンロードさせることも可能である。

なお，H 列「検索テキスト」フィールドのデータは，フルテキスト検索のために用いる（後述「6.3 ネイティブ XML 文書のフルテキスト検索」で詳説）。

※6：Excel ワークシート内のデータは，本書解説用のダミーであり，実在の制度や人物とは無関係である。

データの有無と要素の最低出現回数の表現

XML 化した場合の要素の最低出現回数を 0 回とするか 1 回とするかは，Excel ワークシート上で，セル内に何らかのデータを入力するか，空にするかによって表現しておくとよい。

必ず 1 回出現するものについては，必ず何らかのデータを入力しておく。●図 6-4 では，A，B，C，D，F，G，H，I，J 列（地アミのフィールド）が，これにあたる。そこで，I9，J9 のようにデータがないセルには「—」を入力している。セル内が空でさえなければよいので，「—」でなくとも「未定」「調整中」などでもよい。

一方，出現してもしなくてもよいものについては，空白としておく。本章で説明するXML化の方法では，セル内にデータがなければ，レガシーXMLデータ作成時点で，要素自体が生成されないため，空要素とはならない。ここでは，●図 6-4 の，E，K，L，M，N 列（白地のフィールド）については，データがなければ出現しなくてもよいものとしている。

もし，Excel ワークシート上では空白セルとしておきたいが，後々の処理では空要素が必要ということであれば，XSLT スタイルシートを用いて空要素を生成すればよい。

実務では，説明用の見本である●図 6-4 とは異なり，最低でも何千件，何万件のデータを扱うことになるだろう。仮に，その何割かのデータが空白であったなら，空要素の多い XML 文書を検索対象とすることになる。可読性のために必要でない限り，無用な要素によるデータ爆発は避けたいものである。

深い階層の子孫要素の表現

1 枚の表の上では階層構造を表現するには，一個のセル内に，前進方向のノード，つまり弟のノードや子孫ノードのデータも入力することになる。

●図 6-4 では，L4 の「氏名 2」フィールドに，「半田　富士子|半田　妹子」という二人の氏名が入力されている。これは，最終的に，<氏名>半田　富士子</氏名><氏名>半田　妹子</氏名>という兄弟ノードを生成するための方策である。

それならば，ミドルウエアに XML ツリーを格納する場合のように，1 個のセルの中に，最初から<氏名>半田　富士子</氏名><氏名>半田　妹子</氏名>と記入すればよいように思われる。だが，オペレータが XML を理解していないこともあれば，理解していたとしても，タグの閉じ忘れなどのミスも起こりうる。優れて健康なオペレータでない限り，図のように，同階層のデータの区切り子として「｜」を挿入しておき，XML 生成アプリケーションによって，「|」を</氏名><氏名>に置換する方が安全である。

また，XML ならば同名の要素は何度でも出現できるが，表データにおける同一フィールド名の重複は避ける必要がある。そこで，所属や講師といったメタデータの意味自体は同じであるが，「所属 1」「所属 2」…および「講師 1」「講師 2」…というように，異なるフィールド名としている。これらのフィールド名は，レガシーXML データを作成した後，XSLT スタイルシートを用いて変換することによって統一する（詳細は次節）。

出現回数未定要素の表現

●図 6-4 では，「所属」「氏名」については 3 個のフィールドを設けている。だが，このデータが「IT 講座の実施計画要綱」であるという性格上，運用期間中に，講師を派遣する企業の数が増えないとも限らない。「所属」が 4 社，5 社と増えれば，当然，フィールドを追加する必要が生じる。そこで，出現回数が未定のデータについては，いつでもフィールドを追加できるよう，列の末尾に入力しておく。

もちろん，現段階で考えられるフィールド数（この例では各 3 個）に限定せず，あらかじめ 4 個，5 個といったフィールドを設けておき，データがなければ空のセルのまま残しておく方法もある。

以上のような，少しばかりの工夫により，Excel レガシーデータを不定形多階層構造の XML 文書作成のために活用できるようになる。表の在り方にとらわれない，XML データの表現方法がポイントである。

6.2 不定形多階層構造文書の絞り込み検索

6.2.1 表データからの XML 文書の作成

データの内容から判断するファイル構成と構造

前掲●図 6-4 の Excel レガシーデータを，●図 6-2 の手順でエクスポートすると，レガシーXML データの構造は●リスト 6-1 のようになる（※7）。これを，●図 6-1 のような「絞り込み検索」に適した構造に変換してみよう。

この XML 文書の内容は「IT 講座の実施計画要綱」である。「計画」であるからには，年々データが増えてく可能性がある。そこで，データが増えた場合に備えて，XML 文書ファイルは年度別に分割し，各年度とも，●リスト 6-2 のような階層構造になるよう変換していくものとする。

＊7：●図 6-2 の手順では，ルート要素名は自動的に<dataroot>となる。また，Excel レガシーデータのワークシート名をルート直下の子要素名とし，各フィールド名を孫要素名とするフラットな構造となる。Access に外部データ（この場合 Excel ファイル）をインポートする際，「主キーを自動的に設定する」項目にチェックを付けると，<ID>要素が自動的に追加される。ルート直下の子要素名を Excel レガシーデータのワークシート名以外の名前にするには，[インポート先のテーブル]名を設定する画面で変更する。この名前は，Access におけるテーブル名となる。

エクスポートした XML ソースには，UTF-8 の文字コード指定が付加され，ルート要素には生成日時が属性として付加される。

●リスト 6-1 ●図 6-4 の Excel レガシーデータを Access 経由でエクスポートして生成したレガシーXML データ。


```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
generated="2008-10-21T19:56:24">
<講座>
<ID>1</ID>
<年度>平成 21 年度</年度>
<支援制度>在宅支援</支援制度>
<テーマ>プログラミング</テーマ>
<内容>入門</内容>
<パス>2009</パス>
<コンテンツ>pg_01.htm</コンテンツ>
<検索テキスト>pg_01.xml</検索テキスト>
<所属 1>女性技術者の会</所属 1>
<氏名 1>古見　久美</氏名 1>
</講座>
<講座>～</講座>繰り返し，略。
<講座>
<ID>10</ID>
<年度>平成 22 年度</年度>
<支援制度>在宅支援</支援制度>
```

```
<テーマ>データベース</テーマ>
<内容>実践</内容>
<詳細>物理設計</詳細>
<パス>2010</パス>
<コンテンツ>db_03.htm</コンテンツ>
<検索テキスト>db_03.xml</検索テキスト>
<所属 1>XML 株式会社</所属 1>
<氏名 1>須木間　定義|江楠　絵夢</氏名 1>
<所属 2>視覚スタジオ</所属 2>
<氏名 2>林　功</氏名 2>
</講座>
</dataroot>
```

●リスト 6-2 ●リスト 6-1 のレガシーXML データを変換して作成する，年度別検索対象 XML 文書ファイル。このリストは平成 21 年度分。

```
<?xml version="1.0" encoding="utf-8"?>
<実施計画>
  <講座>
    <年度>平成 21 年度</年度>
    <支援制度>
      <支援制度名>在宅支援</支援制度名>
      <テーマ>
        <テーマ名>プログラミング</テーマ名>
        <内容>
          <タイトル　公開テキスト="2009/pg_01.htm">入門</タイトル>
          <講師>
            <所属>女性技術者の会</所属>
            <氏名>古見　久美</氏名>
          </講師>
        </内容>
        <内容>
          <タイトル　公開テキスト="2009/pg_02.htm">基本</タイトル>
          <講師>
            <所属>シルバーギーク会</所属>
            <氏名>秋葉　定数</氏名>
          </講師>
        </内容>
        <内容>
          <タイトル　公開テキスト="2009/pg_03.htm">応用</タイトル>
          <講師>
            <所属>シルバーギーク会</所属>
```

```
            <氏名>木板　雄拓</氏名>
          </講師>
          <講師>
            <所属>女性技術者の会</所属>
            <氏名>半田　富士子</氏名><氏名>半田　妹子</氏名>
          </講師>
        </内容>
      </テーマ>
    </支援制度>
    <支援制度>
      <支援制度名>就労促進</支援制度名>
      <テーマ>
        <テーマ名>デザイン</テーマ名>
        <内容>
          <タイトル>入門</タイトル>
          <詳細>
            <見出し 公開テキスト="2009/design_01.htm">レイアウト</見出し>
            <講師>
              <所属>XML 株式会社</所属>
              <氏名>江楠　絵夢</氏名>
            </講師>
          </詳細>
          <詳細>
            <見出し 公開テキスト="2009/design_02.htm">色彩</見出し>
            <講師>
              <所属>XML 株式会社</所属>
              <氏名>江楠　絵夢</氏名>
            </講師>
          </詳細>
          <詳細>
            <見出し 公開テキスト="2009/design_03.htm">ツールの操作</見出し>
            <講師>
              <所属>XML 株式会社</所属>
              <氏名>江楠　絵夢</氏名>
            </講師>
            <講師>
              <所属>夢織デザイン</所属>
              <氏名>阿戸　美子</氏名>
            </講師>
          </詳細>
        </内容>
```

```
            <内容>
              <タイトル>実習</タイトル>
              <詳細>
                <見出し 公開テキスト="2009/design_04.htm">課題制作</見出し>
                <講師>
                  <所属>XML 株式会社</所属>
                  <氏名>江楠　絵夢</氏名>
                </講師>
                <講師>
                  <所属>視覚スタジオ</所属>
                  <氏名>志井　紗歩</氏名>
                </講師>
                <講師>
                  <所属>夢織デザイン</所属>
                  <氏名>阿戸　美子</氏名>
                </講師>
              </詳細>
            </内容>
          </テーマ>
        </支援制度>
      </講座>
    </実施計画>
```


変換方法の検討

●リスト 6-1 から●リスト 6-2 への変換方法は，ひとつではない。変換方法は，構造やノード数などを加味して，ケースバイケースで決めればよい。それには，次の 2 つの先入観にとらわれないよう注意しよう。

まず，「必ず 1 個の XSLT スタイルシートを用いて 1 回で変換しなければならない」わけではない。複数の XSLT スタイルシートを用いて，段階を追って変換を繰り返す方が，変換の過程を確認しやすいという利点がある。

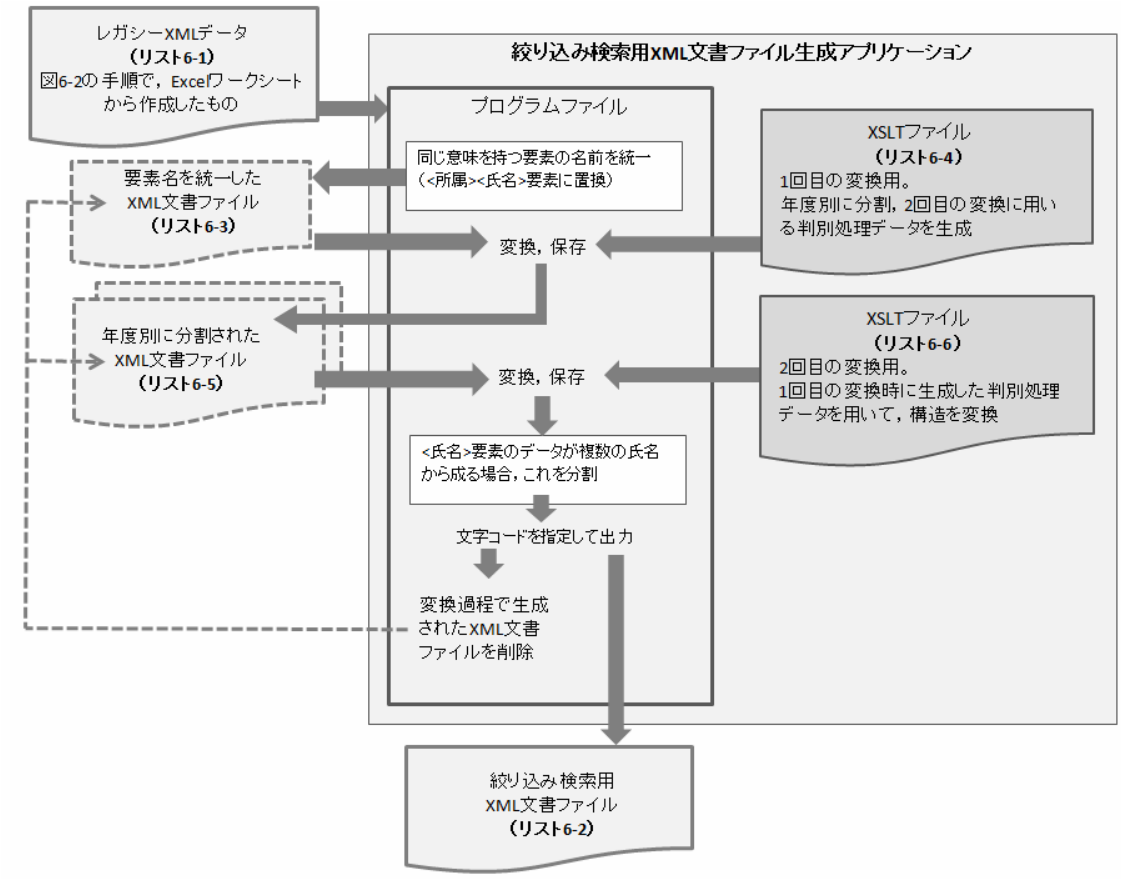
次に，「必ず XSLT スタイルシートのみで変換しなければならない」というわけでもない。条件分岐や変数，複雑なパス式を多用すればするほど，XSLT スタイルシートの可読性は極端に損なわれる。プログラムの方が見通しが良く記述しやすい処理については，XSLT ではなく， LINQ を用いるなどの判断を下してもよい。

ここでは，複数のプログラムと複数の XSLT スタイルシートを併用して変換していく。処理の流れは，次ページ●**図 6-5** のとおりである。

プログラムから XSLT による変換を実行するには，プログラムファイル中にソースとなる XML 文書ファイルと，XSLT ファイルを読み込んで変換を実行し，結果ツリーを XML 文書ファイルとして保存する（第 3 章●図 3-11(2)参照）。変換後の XML ツリーをファイルとして保存する際には，文字化けを防ぐためにも，

出力する文字コードのデフォルト値の確認あるいは指定を忘れないようにしよう。

●図 6-5 ●リスト 6-1 を●リスト 6-2 のような構造の複数文書に変換する処理の流れの例



6.2.2 プログラムと XSLT スタイルシートによる変換

同じ意味を持つ要素の名前の統一

レガシーXML データ（●リスト 6-1）には，Excel ワークシートのフィールド名であった<所属 1><講師 1><所属 2><講師 2>…といった要素名が見られる。これらは，同じ意味とプライオリティを持つので，<所属><氏名>といった同一要素名に統一しなければならない。

そこで，●図 6-5 にもあるように，まずは，これらのタグ名を，プログラムを用いて置換により統一する。

置換処理は，カウンタ用の整数型の変数 i を宣言し，<所属 & i>で表わされる要素名を，<所属>に置換するだけの簡単なものである。<氏名>についても同様である。ただし，変換元の要素数が二桁を超える場合には，注意が必要となる。一桁の場合は，たとえば<所属 9>は<所属>となるが，<所属 10>は，i=10 で置換する前に i=1 の時点で<所属 0>に置換されてしまう。<所属 0>～<所属 i-10>の再処理が必要となる（※8）。

この処理による置換結果は，●リスト 6-3 の通りである。この時点では，講師が複数いる場合，下線部分のように，「|」はそのままである。

※8：.NET Framework 上で動作するプログラムの解説が本書の趣旨ではないので，本章では，実装するうえでのポイントやヒントを解説している。

●リスト 6-3 講師の所属や氏名をプログラムにより置換した結果の一部。

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
generated="2008-10-21T19:56:24">
<講座>～</講座>繰り返し，略。
<講座>
<ID>3</ID>
<年度>平成 21 年度</年度>
<支援制度>在宅支援</支援制度>
<テーマ>プログラミング</テーマ>
<内容>応用</内容>
<パス>2009</パス>
<コンテンツ>pg_03.htm</コンテンツ>
<検索テキスト>pg_03.xml</検索テキスト>
<所属>シルバーギーク会</所属>
<氏名>木板　雄拓</氏名>
<所属>女性技術者の会</所属>
<氏名>半田　富士子|半田　妹子</氏名>
</講座>
<講座>～</講座>繰り返し，略。
<講座>
<ID>6</ID>
<年度>平成 21 年度</年度>
<支援制度>就労促進</支援制度>
<テーマ>デザイン</テーマ>
<内容>入門</内容>
<詳細>ツールの操作</詳細>
<パス>2009</パス>
<コンテンツ>design_03.htm</コンテンツ>
<検索テキスト>design_03.xml</検索テキスト>
<所属>XML 株式会社</所属>
<氏名>江楠　絵夢</氏名>
<所属>夢織デザイン</所属>
<氏名>阿戸　美子</氏名>
</講座>
<講座>～</講座>繰り返し，略。
</dataroot>
```

1 枚のワークシートの複数ファイルへの分割

同一の意味を持つ要素名を統一できたところで，今度は，年度別に分割した XML 文書ファイルを生

成する。それには，分割の基準となる対象要素の内容を，条件分岐処理の判別に用いる。●リスト 6-4 は，「平成 21 年度」分の XML 文書ファイルを作成するコードである。

<xsl:if test="年度='平成21年度'">～</xsl:if>という条件分岐コードにより，生成するデータを「平成21年度」のみに絞り込んでいる。

なお，本例とは異なり，これがアンケート集計表などであれば，既存 Excel データには，●図 6-6 のように，"0"と"1"が記入されている場合も少なくないだろう。その場合は，<xsl:if test="フィールド名='0'"></xsl:if>という分岐処理で対応すればよい。

●図 6-6 アンケート集計表などでよく見受けられる"0"や"1"が記入されている例。

	A	B	C	D	E
1	平成21年度	平成22年度	在宅支援制度	就労促進制度	テーマ
2	1	0	1	0	プログラミン
3	1	0	1	0	プログラミン
4	1	0	1	0	プログラミン
5	1	0	0	1	デザイン
6	1	0	0	1	デザイン
7	1	0	0	1	デザイン
8	1	0	0	1	デザイン
9	0	1	1	0	データベース
10	0	1	1	0	データベース
11	0	1	1	0	データベース

●リスト 6-4 ワークシートのデータを，年度別の複数文書ファイルに分割する XSLT スタイルシートの一例。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" version="1.0" indent="yes"/>

<xsl:template match="/">

<実施計画>
<xsl:for-each select="dataroot/講座">  ―――― ①
 <xsl:if test="年度='平成 21 年度'">  ―――― ②
  <講座>
   <年度><xsl:value-of select="年度"/></年度>
   <支援制度><xsl:value-of select="支援制度"/></支援制度>

   <xsl:variable name="theme" select="テーマ"/> ―――― ③
   <支援制度別テーマ><xsl:value-of select="concat(支援制度,$theme)"/></支援制度別テーマ>
   <テーマ><xsl:value-of select="$theme"/></テーマ>

   <xsl:variable name="contents" select="内容"/> ―――― ④
   <支援制度別テーマ別内容>
     <xsl:value-of select="concat(支援制度,$theme,$contents)"/>
   </支援制度別テーマ別内容>
```

```
    <内容><xsl:value-of select="$contents"/></内容>

    <xsl:variable name="details" select="詳細"/>―――⑤
    <支援制度別テーマ別内容別詳細>
      <xsl:value-of select="concat(支援制度,$theme,$contents,$details)"/>
    </支援制度別テーマ別内容別詳細>
    <詳細><xsl:value-of select="$details"/></詳細>

    <パス><xsl:value-of select="パス"/></パス>
    <コンテンツ><xsl:value-of select="コンテンツ"/></コンテンツ>

    <xsl:for-each select="所属">―――⑥                    ⑦
      <所属><xsl:value-of select="."/></所属>              |
      <xsl:variable name="instructor" select="following-sibling::氏名[position()=1]"/>
      <氏名><xsl:value-of select="$instructor"/></氏名>
    </xsl:for-each>
  </講座>
 </xsl:if>
</xsl:for-each>
</実施計画>
</xsl:template>
</xsl:stylesheet>
```

①dataroot/講座に対して，処理を繰り返す。
②<年度>が「平成 21 年度」ならば以下の処理を実行する。
③<テーマ>要素の内容を変数に代入し，これを用いて<支援制度別テーマ>要素を生成するとともに，<テーマ>要素を生成する。
④<内容>要素の内容を変数に代入し，これを用いて<支援制度別テーマ別内容>要素を生成するとともに，<内容>要素を生成する。
⑤<詳細>要素の内容を変数に代入し，これを用いて<支援制度別テーマ別内容別詳細>要素を生成するとともに，<詳細>要素を生成する。
⑥<所属>要素に対して処理を繰り返す。
⑦following-sibling 軸を使って，<所属>要素の後の兄弟である<氏名>要素を選択し，<氏名>要素をを生成する。

●リスト 6-4 では，年度別分割以外にもうひとつ，重要な処理がある。それは，<支援制度別テーマ>要素と<支援制度別テーマ別内容>要素と<支援制度別テーマ別内容別詳細>要素を生成しておくことである（※9）。

なぜ，これらの要素を生成する必要があるのか説明しよう。

最終的な結果として得たい前掲●リスト 6-2 を見れば分かるように，データが「在宅支援」である<支援

制度>要素は一度しか出現しておらず，データが「プログラミング」である<テーマ>要素は一度しか出現していない。つまり，年度別に分割した後，その次の段階の構造変換（後述●リスト 6-5）では，データが同一であれば，要素を生成することなくスルーする処理が必要となる。具体的に言えば，「<支援制度>が同じで，<支援制度>別の<テーマ>が重複していなければ，<テーマ>要素を生成する」「<支援制度>と<テーマ>が同じで，<支援制度>別<テーマ>別の<内容>が重複していなければ，<内容>要素を生成する」…といった重複を避ける変換処理が必要となる。

そこで，この段階で，データが同一かどうかを判別するための条件となるデータを，あらかじめ生成しておくのである。

ここで，ひとつ疑問が湧くかもしれない。それは，「<支援制度>別の<テーマ>が重複していなければ」という条件ではなく，「<テーマ>が重複していなければ」という条件で十分ではないかという考え方である。たしかに，この例では一見不要に見える。だが，それは，たまたま，今変換しようとしているデータでは，<支援制度>と<テーマ>のデータが 1 対 1 対応となっているからにすぎない。

ここで例として用いているデータは，この先 2 年以降の「計画」を表したものである。●図 6-7 のようなデータの変更や追加の可能性は否定できない。単に，「<テーマ>が重複していなければ」という条件では，「デザイン」を内容とする要素は一度しか生成されないので，「就労促進」の「デザイン」はスルーされてしまうことになる。

このように，現段階で必要な最低限の変換処理を記述していると，想定外のデータが追加されたとき，誤った検索対象 XML 文書が生成されることになる。データ件数が多ければ，誤った文書が作成されていることに，気づかない恐れすらある（後述，「6.5 設計後の工程に影響する問題」も参照）。

実務では，将来追加されると予測されるデータを見越した上で，次の段階での変換処理との連携変換処理も考えなければならない。なにごとも，現段階では不要だからといって，将来的にも不要であるとは限らないのである。

●図 6-7 誤った構造変換を避けるために，データ重複を予測する。

現在のデータ

支援制度	テーマ	内容	詳細
在宅支援	プログラミング	入門	
在宅支援	プログラミング	基本	
在宅支援	プログラミング	応用	
就労促進	デザイン	入門	レイアウト
就労促進	デザイン	入門	色彩
就労促進	デザイン	入門	ツールの操作
就労促進	デザイン	実習	課題制作

予測されるデータの例

支援制度	テーマ	内容	詳細
在宅支援	プログラミング	入門	ツールの操作
在宅支援	プログラミング	基本	
在宅支援	プログラミング	応用	
在宅支援	デザイン	入門	ツールの操作
就労促進	デザイン	入門	レイアウト
就労促進	デザイン	入門	色彩
就労促進	デザイン	入門	ツールの操作
就労促進	デザイン	実習	課題制作

追加されたデータ

「<テーマ>が重複していなければ」という条件だけでは，構造変換の際にスルーされてしまう。

なお，Excel ワークシート（●図 6-4）では空要素を生成しないセルについては，データを空白としているが，●リスト 6-4 の XSLT スタイルシートでは，空要素が生成されることになる。この空要素は以降の変換で用いる。

●リスト 6-4 による年度別分割処理の結果は，●リスト 6-5 のようになる。

●リスト 6-5 ●リスト 6-4 の XSLT スタイルシートによる変換結果。

```
<?xml version="1.0" encoding="utf-8"?>
<実施計画>
  <講座>
    <年度>平成 21 年度</年度>
    <支援制度>在宅支援</支援制度>
    <支援制度別テーマ>在宅支援プログラミング</支援制度別テーマ>
    <テーマ>プログラミング</テーマ>
    <支援制度別テーマ別内容>在宅支援プログラミング入門</支援制度別テーマ別内容>
    <内容>入門</内容>
    <支援制度別テーマ別内容別詳細>
      在宅支援プログラミング入門
    </支援制度別テーマ別内容別詳細>
    <詳細></詳細>
    <パス>2009</パス>
    <コンテンツ>pg_01.htm</コンテンツ>
    <所属>女性技術者の会</所属>
    <氏名>古見　久美</氏名>
  </講座>
  <講座>～</講座>繰り返し，略。
  <講座>
    <年度>平成 21 年度</年度>
    <支援制度>就労促進</支援制度>
    <支援制度別テーマ>就労促進デザイン</支援制度別テーマ>
    <テーマ>デザイン</テーマ>
    <支援制度別テーマ別内容>就労促進デザイン入門</支援制度別テーマ別内容>
    <内容>入門</内容>
    <支援制度別テーマ別内容別詳細>
      就労促進デザイン入門レイアウト
    </支援制度別テーマ別内容別詳細>
    <詳細>レイアウト</詳細>
    <パス>2009</パス>
    <コンテンツ>design_01.htm</コンテンツ>
    <所属>XML 株式会社</所属>
    <氏名>江楠　絵夢</氏名>
  </講座>
  <講座>～</講座>繰り返し，略。
</実施計画>
```


※9：長い日本語の要素名を採用しているのは，本節の説明を分かりやすくするためである。膨大なノード数を扱う実務では，英字の短い要素名にする方が処理を高速化することができる。

重複を避けた構造化

●リスト 6-5 は，まだレガシーXML データ同様，フラットな構造である。XSLT スタイルシートを用いて，これを階層構造化するには，●リスト 6-6 のように記述する（※10）。

この構造変換では，同一データを持つ同一要素の重複を避けることが重要である。それには，<講座>要素ノードに対して処理を繰り返し，重複しないデータに対してのみ結果ツリーを生成していく。

具体的には，まず，<xsl:for-each>要素を用いて<講座>ノードを繰り返し走査する（①）。そして，<xsl:if test="not(支援制度=preceding::支援制度)">のように記述し，「<支援制度>要素の内容が重複していなければ」という条件を指定する（②）。つまり，重複を避けたいデータが，その前に出現していない（not）ならば，という条件を指定しているのである（パス式については第 3 章●表 3-2 参照）。この条件に合致するならば，<支援制度>要素の内容をテキスト化する。ここで取得した<支援制度>データは，<xsl:variable>要素で宣言した変数 supportProgram に代入しておく（③）。

次に，再度<xsl:for-each>要素を用いて<講座>ノードを繰り返す（④）。その中で，取得したデータと変数に代入されているデータを比較し，「<支援制度>データが同じであれば」という条件を指定する（⑤）。さらに，「<支援制度別テーマ>要素の内容が，文書順で先に出現している内容と等しくないならば，つまり重複していなければ」という条件を指定する（⑥）。これにより，条件に合致した要素ノードのみを選択することとなり，結果的に重複を避けることができる。同様の処理の繰り返しにより，結果ツリーの部分を順次構築していく。

●リスト 6-6 では，他にも重要な点がある。ここでは，<パス>要素と<コンテンツ>要素のデータを連結して，"公開テキスト"属性値としている。その"公開テキスト"属性の従属先となる要素の特定である。前掲●リスト 6-2 の通り，<内容>要素があり<詳細>要素がなければ，<内容>要素の子として，たとえば，<タイトル 公開テキスト="2009pg_01.htm">入門</タイトル>を生成する。だが，<詳細>要素があった場合は，<詳細>要素の子として，たとえば，<見出し 公開テキスト="2009design_01.htm">レイアウト</見出し>を生成している。

同様に，<講師>要素も，<内容>要素の子とするのか，<詳細>要素の子とするのかを特定したうえで，生成している。

それには，<xsl:if test="string-length(下位の階層の要素名)=0">のように記述して，下位の階層に，データが存在するかどうかを条件とする。<内容>要素のデータがあり（⑦），且つ<詳細>要素のデータがないならば（⑧），<内容>要素に対して HTML ファイルをリンクさせ，<講師>要素を子要素として生成している（⑨）。また，<詳細>要素のデータがある場合は，<詳細>要素に対して HTML ファイルをリンクさせ，<講師>要素を子要素として生成している（⑩）。

※10：●リスト 6-4，●リスト 6-6 ともに，XSLT スタイルシートの記述方法は例としてあげた一種類だけではない。他の言語によるプログラム同様，百人百様の記述方法がある。要素の各ローカル名をあらかじめ取得したうえで変数を用いる方法もある。ここでは説明に適した記述としている。XSLT では再帰処理も実装可能だが，見通しが良いとはいえないので，お勧めしない。

●リスト 6-6 ●リスト 6-5 を●リスト 6-2 の構造に変換する XSLT1.0 スタイルシート。

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" version="1.0" indent="yes"/>

<xsl:template match="/">
<実施計画>
 <講座>
  <年度><xsl:value-of select="実施計画/講座/年度[1]"/></年度>
   <xsl:for-each select="実施計画/講座">  ←――――― ①
    <xsl:if test="not(支援制度=preceding::支援制度)">←――― ②
     <支援制度>
      <支援制度名><xsl:value-of select="支援制度"/></支援制度名>
      <xsl:variable name="supportProgram" select="支援制度"/>  ←――― ③
      <xsl:for-each select="/実施計画/講座">←――― ④
       <xsl:if test="支援制度=$supportProgram">←――― ⑤
        <xsl:if test="not(支援制度別テーマ=preceding::支援制度別テーマ)">←――― ⑥
         <テーマ>
          <テーマ名><xsl:value-of select="テーマ"/></テーマ名>
          <xsl:variable name="supportProgram_theme" select="支援制度別テーマ"/>
          <xsl:for-each select="/実施計画/講座">
           <xsl:if test="支援制度別テーマ=$supportProgram_theme">
            <xsl:if test="not(支援制度別テーマ別内容=preceding::支援制度別テーマ別内容)">
             <xsl:if test="string-length(内容) &gt;0">  ←――― ⑦
              <内容>
               <タイトル>
                <xsl:if test="string-length(詳細)=0">  ←――― ⑧
                <xsl:attribute name="公開テキスト">
                 <xsl:value-of select="パス"/><xsl:text>/</xsl:text>  ←┐
                 <xsl:value-of select="コンテンツ"/></xsl:attribute>  ←┘ ⑨
                </xsl:if>
                <xsl:value-of select="内容"/>
               </タイトル>
               <xsl:variable  name="supportProgram_theme_contents"  select="支援制度別
テーマ別内容"/>
               <xsl:if test="string-length(詳細)=0">
                <xsl:for-each select="所属">
                 <講師>
                  <所属><xsl:value-of select="."/></所属>
                  <xsl:variable  name="instructor"  select="following-sibling::  氏  名
[position()=1]"/>
```

```
<氏名><xsl:value-of select="$instructor"/></氏名>
</講師>
</xsl:for-each>
</xsl:if>
<xsl:for-each select="/実施計画/講座">
<xsl:if test="支援制度別テーマ別内容=$supportProgram_theme_contents">
<xsl:if  test="not(支援制度別テーマ別内容別詳細=preceding::支援制度別テーマ別内容別詳細)">
<xsl:if test="string-length(詳細) &gt;0">  ←――⑩
<詳細>
<見出し>
<xsl:attribute name="公開テキスト">
<xsl:value-of select="パス"/><xsl:text>/</xsl:text>
<xsl:value-of select="コンテンツ"/></xsl:attribute>
<xsl:value-of select="詳細"/>
</見出し>
<xsl:for-each select="所属">
<講師>
<所属><xsl:value-of select="."/></所属>
<xsl:variable  name="instructor"  select="following-sibling:: 氏 名[position()=1]"/>
<氏名><xsl:value-of select="$instructor"/></氏名>
</講師>
</xsl:for-each>
</詳細>
</xsl:if>
</xsl:if>
</xsl:if>
</xsl:for-each>
</内容>
</xsl:if>
</xsl:if>
</xsl:if>
</xsl:for-each>
</テーマ>
</xsl:if>
</xsl:if>
</xsl:for-each>
</支援制度>
</xsl:if>
</xsl:for-each>
```

```
 </講座>
</実施計画>
</xsl:template>
</xsl:stylesheet>
```

この●リスト 6-6 による変換結果では，まだ一連の複数の氏名「半田　富士子¦半田　妹子」は，一要素のデータとなっている。そこで，プログラムにより「¦」を</氏名><氏名>で置換して，<氏名>半田　富士子</氏名><氏名>半田　妹子</氏名>といった構造にする必要がある。この置換処理は，構造変換の後で実行したのでよい。

これら一連の変換終了後には，途中で一時的に生成した XML 文書ファイルは自動削除するプログラムを記述しておき，不要なファイルは蓄積しないほうがよい。

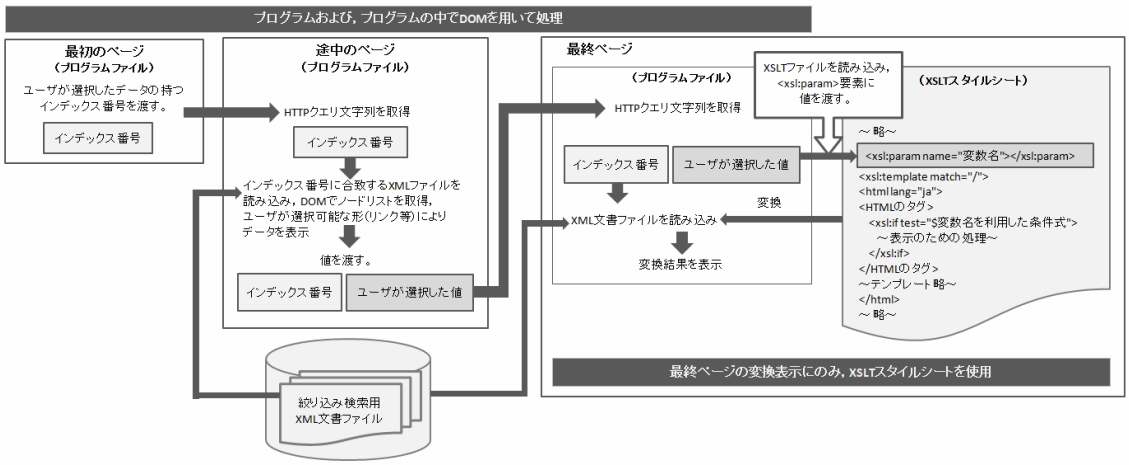
6.2.3 絞り込み検索の実装

絞り込み検索処理を Web アプリケーションとして実装するにあたっては，XML 文書の階層が深くデータ件数が多いならば，スクロール等のユーザインタフェース上の問題から，1 フォームに表示する階層数を限定し，複数のフォームに分割して表示した方がよい場合もある。

その実装には，（もし W3C 仕様を採用するのであれば）DOM と XSLT を使い分け，最終フォームの表示に XSLT を用いると，メンテナンスが容易になる。基本的に，XSLT は構造変換とスタイルのために用い，それ以外はプログラムの中で DOM を使う，と捉えておけばよい（**※11**）。

途中のフォームでは，プログラムの中で DOM を用いて，対象ノードの内容を取得し，クエリ文字列として送信する。そして，最終フォームで，XSLT スタイルシート中の xsl:param 要素に，受け取った内容を代入して変換に利用する（●図 6-8）。

●図 6-8 絞り込み検索における XSLT の用い方



絞り込み検索は，データの内容とその分類方法を熟知しているユーザにとっては，目的のデータに迅速に到達できる便利な機能である。これは，ごく限られた特定の職種のユーザ対象に用いると，効果的である。だが，分類方法を予測できないユーザが多数訪問する Web アプリケーションでは，ユーザは何を選択して先に進めばよいのか見当がつかない。広い層のユーザを対象とするのであれば，基本的に，次節で述べる，フルテキスト検索の方が有効である。

※11：XSLT スタイルシートの中にイベントハンドラを記述して，スクリプトを呼び出したり，XSLT によってスクリプトコード自体を生成することもできるが，実装可能性と実用性は必ずしも一致しない。

6.3 ネイティブ XML 文書のフルテキスト検索

6.3.1 ストップワード方式による検索対象文書の作成

Excel レガシーデータとデータファイルの作成

XML 文書を対象とするフルテキスト検索は，ユーザの入力あるいは選択した文字列が，検索対象 XML 文書ファイル中のテキストノードの文字列に一致するか含まれる場合に，結果を返す処理である。絞り込み検索と異なり，管理ファイルだけでなく，本文内容を記録したデータファイルも検索対象となることが少なくない（冒頭●図 6-1 参照）。前掲●リスト 6-2 は管理ファイルにあたり，"公開テキスト"属性値の HTML ファイル中のデータが本文内容を表す，という位置付けになる。

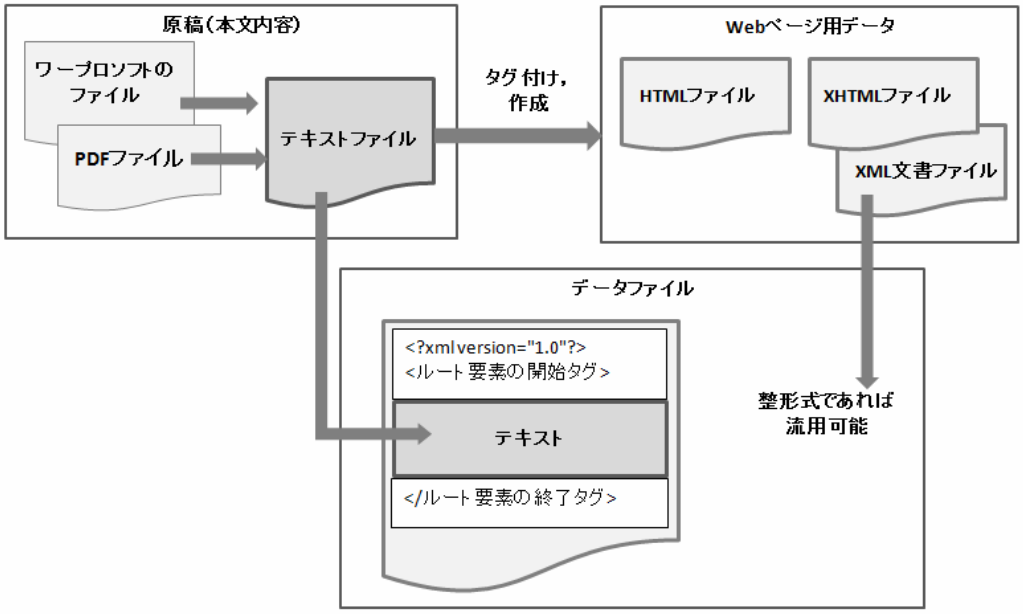
本文内容も検索対象とするならば，データファイルを準備しなければならない。最良の方法は，XML 文書あるいは，XHTML で作成された Web ページをデータファイルとして流用することである。もっとも，実務では，工程上，原稿への完全なタグ付けを待つことができないかもしれない。その場合は，本文内容である原稿のテキストに，XML 宣言およびルート要素のみを追加して，XML 化する方法が現実的である（●次ページ，図 6-9）。フルテキスト検索における検索対象は，ルート要素以下のすべてのテキストノードであるから，XML 文書中の要素がルート要素 1 個しかなくとも，それで問題が生じることはない。

作成した各データファイルは，ローカル環境の特定のフォルダに蓄積する。これらの各ファイル名は，Excel ワークシートに入力しておく。前掲●図 6-4 で，「検索テキスト」フィールドにファイル名を記入しているのが，これにあたる。

しかしながら，多数のデータファイルからの縦断検索となると，頻繁なファイルのオープン/クローズが発生する（前章●表 5-8 参照）。それならば，（全ノード数や実行環境にもよるが）管理ファイルと複数のデータファイルを結合して，単一のファイルとして処理する方が，走査効率は良い。

とはいえ，単一ファイルでは，運用開始後にデータが追加され続けると，ファイルサイズが極端に肥大化する恐れがある。実際のところは，運用期間が長引くにつれ，ハードウェアや開発フレームワーク，通信環境も進化するので，走査効率がさほど極端に低下することはないのだが，第 4 章で述べたように，データ爆発を抑制するための心構えは必要である。

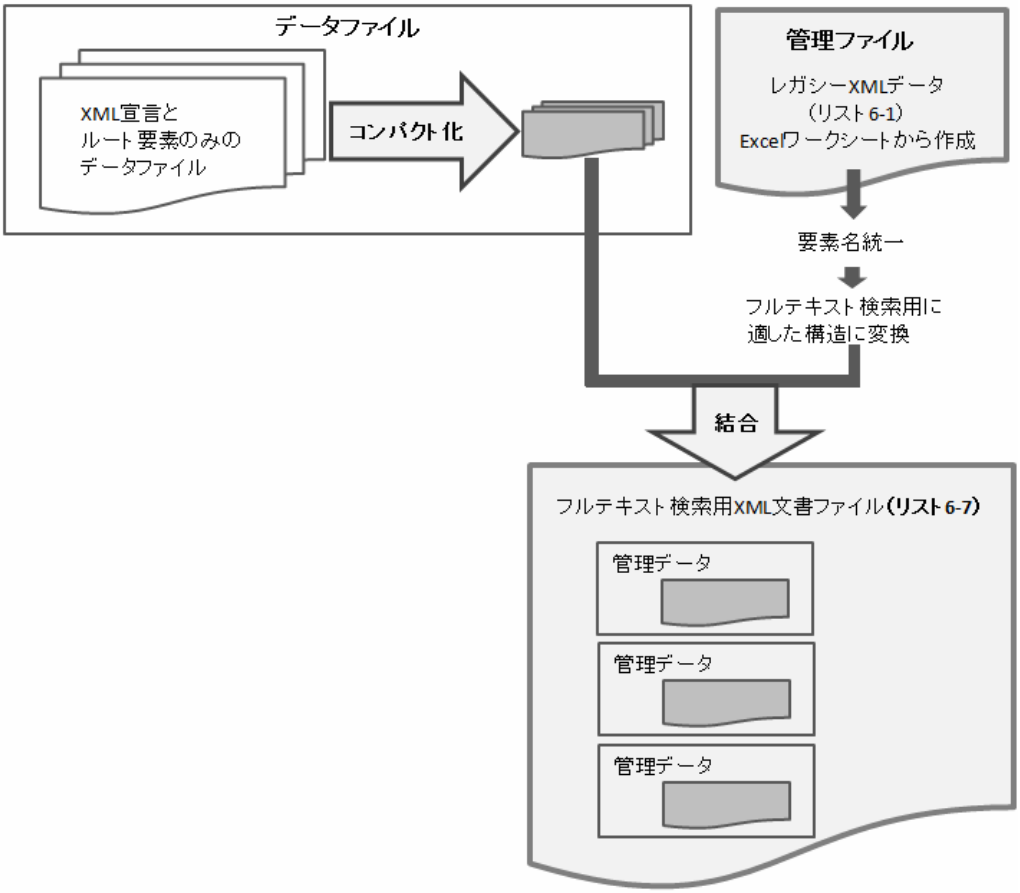
●図 6-9 本文内容の原稿テキストに，XML 宣言およびルート要素のみ追加して，データファイルを作成する



そこで，各データファイルをコンパクト化しておく。そして，それらのデータを，レガシーXML データから変換により作成した管理ファイルに結合するのである（●図 6-10）。●リスト 6-7 のように，データファイルはコンパクト化して，<検索テキスト>要素の内容とする。

以上のような一連の処理を行うアプリケーションを開発し，ローカル環境で実行し，Excel レガシーデータから検索対象 XML 文書ファイルを自動生成すればよい。

●図 6-10 検索対象とする複数の XML 文書をコンパクト化して，Excel レガシーデータと結合する



●リスト 6-7 フルテキスト検索用に構造変換した管理データに，コンパクト化したデータ（下線部）を結合した構造の例

```
<?xml version="1.0"?>
<実施計画>
 <講座>
  <本文内容>「検索テキスト（pg_03.xml）」のデータをコンパクト化したデータを入れる</本文内容>
   <実施要項>
    <年度>平成 21 年度</年度>
    <支援制度>在宅支援</支援制度>
    <テーマ>プログラミング</テーマ>
    <内容>
     <タイトル 公開テキスト="2009/pg_01.htm">入門</タイトル>
     <講師>
       <所属>女性技術者の会</所属>
       <氏名>古見 久美</氏名>
     </講師>
    </内容>
   </実施要項>
  </講座>
  <講座>～</講座>繰り返し，略。
</実施計画>
```


データファイルのコンパクト化

各データファイルをコンパクト化する方法のひとつとして，本節では，形態素解析を用いないストップワード方式をとりあげる。旧来，日本語におけるストップワード（検索に用いられないと予測される語句）といえば，接続詞，助詞，句読点，副詞，場合によっては助動詞，前置詞，冠詞などであるが，本節では，品詞とは無関係に，任意のストップワードを記録した XML 形式の設定ファイルを作成して用いる。

一例をあげれば，技術者のユーザが製品情報を検索するケースでは，「効率化」「コスト削減」といった言葉ならまだしも「実現」「夢」といった言葉を検索キーに指定することはないといっていい。だが，それらの"ユーザにとっては見慣れ過ぎた"言葉を，広告デザイナやコピーライタは製品の数以上に用いる傾向がある（というよりも，それらの言葉を書けば納得するクライアントもいる，という側面もあるだろう）。こういった，検索にはほとんど用いられない言葉をストップワードとして指定しておくのである。

ストップワードを削除すると，語句単位に区切られることになり，フレーズでの検索は困難となる。だが，フルテキスト検索処理では，通常，語順よりも，合致する語句の有無が重要である。語句単位に区切っても，検索結果への影響は，さほど大きいとはいえない。また，ここで紹介するストップワード方式では辞書を用いないので，新しい用語が多用されている文書や，複数の関連性の薄い専門分野にわたる文書からの検索に有効である。

インデックスも作成しない。したがって，初回およびデータ更新に際して，検索対象 XML 文書の生成に要する時間は，マシンスペックやファイル数やノード数にもよるが，担当者二人が示唆呼称しながら進

めたとしても，小規模案件で数分～30 分，中規模案件で数時間も見ておけばよいだろう。

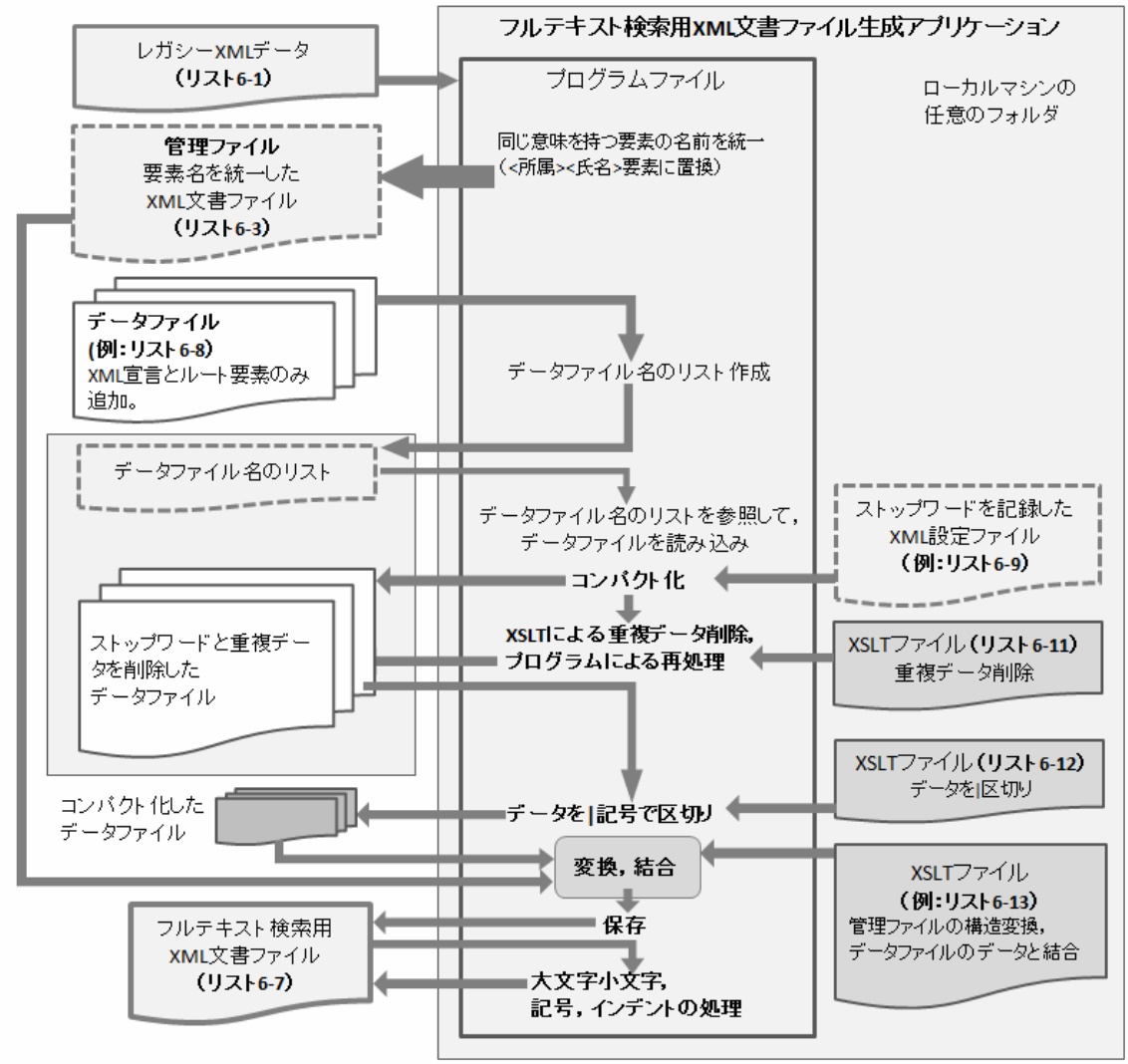
何より，この方法には，ストップワードの設定如何で，コンパクト化率が向上するという利点がある。

処理手順としては，まず，データファイルからストップワードを削除する。

ストップワードによって区切られたデータの中には，重複する語句もあるのでこれを省き，一意な語句のみを残すことによって，一層コンパクト化する。ストップワードを削除したうえで重複語も省くのには理由がある。ネイティブ XML 文書からの検索処理に，検索エンジンのような大規模で汎用的な機能が期待されることは，まず，ないといっていい。その役割は，ユーザ層を限定した小回りの利く検索，とでもいえばいいだろうか。そういった手軽な検索処理においては，出現回数の多い語句が，必ずしも文書の内容を的確に表すとは限らないのである。

さらに，タグもファイルサイズを膨張させる原因であるから一文字の記号「|」（パイプ記号）に置換する。「|」は or を表すという意味で用いている（第 2 章●「表 2-3」参照）そのうえで，それらの各データファイル中のデータを，管理ファイルと結合して，単一の XML 文書ファイルとするのである（●図 6-11）。

●図 6-11 ストップワード方式によるデータファイルのコンパクト化の処理の流れ



ストップワードによるコンパクト化の例

では，●図 6-11 の方法によって，実際にデータファイルはどのようにコンパクト化されるのか。次に簡単な例を示す。

たとえば，●リスト 6-8 のようなテキストを本文とする XML 文書を，●リスト 6-9 のようなストップワードによってコンパクト化すると，●リスト 6-10 のような結果となる。

●リスト 6-8　データファイル中のテキスト

```
<?xml version="1.0"?>
<dataroot>
XML の本質とは，「データに最もふさわしい意味と構造」である。そして，XML の実践とは，「的確な予測と判断に基づいて，その構造をデザインすること」である。
</dataroot>
```

●リスト 6-9 ●リスト 6-8 に適用するストップワードの設定ファイルの例

```
<?xml version="1.0"?>
<words>
  <data>
    <word> そ し て </word><word> そ の </word><word> す る </word><word> で あ る
</word><word> こ と </word><word> と は </word><word> 基 づ い て </word><word> 最 も
</word><word> い </word><word> さ </word><word> し </word><word> と </word><word> な
</word><word> に </word><word> の </word><word> は </word><word> ふ </word><word> わ
</word><word> を </word><word>「 </word><word> 」 </word><word>， </word><word> 。
</word>
  </data>
</words>
```

●リスト 6-10 ●リスト 6-8 から●リスト 6-9 のストップワードを削除し，重複語も削除して，コンパクト化した結果

```
<本文内容>予測 | 本質 | 判断 | 的確 | 実践 | 構造 | 意味 | デザイン | データ | XML | | </本文内容>
```

ストップワードの設定において，注意しなければならないことが，何点かある。ひとつは，ひらがなを設定する場合である。たとえば，「XML の本質とは」といったテキスト対し，「とは」「は」という順序にストップワードを設定すると，「XML の本質」が残る。だが，「は」「とは」という順序に設定すれば，「XML の本質と」が残ることになる。つまり，ストップワードの出現順によっても，コンパクト化の効率は異なるのである。

また，ひらがな一文字をすべてストップワードとして設定すると，たとえばそれが医療情報であれば，「の」ならびに「ど」は省きたいが「のど」まで削除されてしまうことになる。これでは，ストップワード方式では，重要な語句が根こそぎ削除されてしまい，検索対象として不十分なものとなるように見える。

だが，これは，実にささいな問題でしかない。なぜならば，多くの場合，重要な語句の多くは，管理ファイルの中に含まれているからである。管理ファイル中のデータはコンパクト化の対象となっていないため，データファイルのコンパクト化により削除された部分の情報を十分補うことができる。前掲●リスト 6-7 で

いえば，<本文内容>要素内のデータがコンパクト化されたとしても，<実施要項>内のデータはそのままである。そして，重要な語句は<実施要項>内に含まれているのである。

6.3.2 検索対象文書生成アプリケーションの実装

プログラムと XSLT スタイルシートの併用

ストップワード方式を採用するには，まず，ストップワードを記録した XML 形式の設定ファイルを作成しておく必要がある。

そして，処理を円滑にするために，プログラムにより，データファイルを蓄積したフォルダを指定して，フォルダ中のファイル名リストを XML 文書ファイルとして生成しておく。

このファイルリストに基づいて各データファイルを次々読み込み，その中のテキストとストップワード設定ファイルのデータを順次照合しては，不要な語を削除していく。ひらがな一文字だけが残れば，これも削除する。残ったデータは順次配列変数に格納していき，任意のタグを付けて XML 形式とし，これを XML 文書ファイルとして仮保存する。

次に，ストップワードによりコンパクト化された各データファイルから，XSLT スタイルシートを用いて，「完全に一致する」重複データを省く。●リスト 6-11 は，コンパクト化の結果として区切られたデータに，<データ>というタグを付けて仮保存した場合の，重複していないデータを抽出する，つまり重複するデータを削除する XSLT コードの記述例である（前掲●リスト 6-6⑥も参照）。さらに，プログラムでデータを削除したデータファイル中のデータを，2 つの変数に格納して両者を照合し，一方が一方に「含まれる」データを省く。

●リスト 6-11 重複していないデータ**を抽出する XSLT スタイルシートの例**

```
～略～
<xsl:for-each select="データ">
  <xsl:sort select="." data-type="text" order="descending" />
   <xsl:if test="not(.=preceding::データ)">
    <データ><xsl:value-of select="." /></データ>
   </xsl:if>
</xsl:for-each>
～略～
```

また，タグもファイルサイズを膨張させる原因であるから，XSLT スタイルシートを用いて，データファイル中のタグを「¦」のような 1 文字の記号に置き換えて，サイズダウンをはかる（●リスト 6-12）。不要な改行や空白もできるだけ削除する。以上の処理によりコンパクト化したデータファイルは，コンパクト化する前と同一ファイル名で，別フォルダに保存する。

●リスト 6-12 ●リスト 6-11 で完全な重複を省き，さらにプログラムでも部分一致する重複を省いたデータを，タグではなく「|」記号で区切り，サイズダウンをはかる

```
～略～
<xsl:for-each select="データ">
  <xsl:value-of select="." />|
</xsl:for-each>
～略～
```

管理ファイルと複数データファイルの結合

コンパクト化された各データファイルのデータは，レガシーXML データ中の<所属><氏名>要素を置換した文書（●リスト6-3，前述の「要素名あるいは属性名の置換」参照）に結合し，単一のXML文書ファイルとして構築していく。

いま，レガシーXML データ中には，<検索テキスト>pg_01.xml</検索テキスト>というように，データファイル名が記録されている。先の手順では，コンパクト化したデータファイルも同名で保存されているので，●リスト 6-13 のように，XSLT スタイルシートの document 関数でファイル名を指定してアクセスし，ルート要素以下を処理すればよい（document 関数については第 3 章●リスト 3-29 参照）。

●リスト 6-13 レガシーXML データを変換する処理の中で，各データファイルのデータも結合する XSLT の一部分

```
～略～
<xsl:variable name="fileName" select="検索テキスト"/>
<xsl:variable name="contents" select="document($fileName)/データファイルのルート要素名" />
<本文内容><xsl:value-of select="$contents"/></本文内容>
～略～
```

大文字小文字，記号，インデントの扱い

このようにして生成した単一 XML 文書ファイル中のデータは，検索プログラムとの兼ね合いから，必要があれば英字の大文字小文字，テキストの全角や半角を統一する。これらの処理を実装する際には，テキストノードだけでなく，タグ名も含めて単純置換してしまうと，整形式文書ではなくなるので注意する。また，記号についても，たとえば"＆"を"&"，"＜"を"<"，"＞"を">"に置換するなどし，文字コードを指定して保存する。

逆に，XML データ処理アプリケーション側では，ユーザが入力した値に，「<」や「&」などの文字が含まれる場合に備えて，フレームワークの持つ検証機能なども利用して，安全に実行できるようにしておく必要がある。

なお，ヒトが XML 文書のソースを見るためには，タグはインデントされている方がよいが，コンピュータにとってインデントは不要である。Excel レガシーデータや，それから展開したレガシーXML データが存在するのであるから，検索処理用の XML 文書が可読性に欠けても問題はない。むしろ可読性に欠ける方が，修正変更が生じた場合に，検索対象 XML 文書の中だけを応急処置で編集する恐れがないから，Excel レガシーデータとの不整合を回避することができる。プログラムによってインデントを削除するだけでもファイルをサイズダウンすることができ，読み込み速度の向上を見込むことができる。

6.3.3 ストップワードのカスタマイズ

先に，この方法には，ストップワードの設定如何で，コンパクト化率が向上するという利点があると述べた。実際，ストップワードの指定如何によって，サイズダウンの幅やユーザの利便性が大きく異なる。ストップワードには，ユーザが検索キーとして指定しないことがほぼ確実である語句を設定しなければならない。検索処理において，キーワードを指定するのはコンピュータではなく，ヒトである。また，検索キーワードの蓄積からコンピュータが判断できるのは，検索に用いられやすい言葉であって，検索にまず用いることのない言葉ではない。したがってこれは，ユーザと同じ母国語を使うヒトが行うべき作業である。ヒトがデータファイルに目を通し，コンテキスト思考によって，ストップワードを設定することに意義がある。

とりわけ，対象文書が特定の分野や業種のものであれば，その方面のテキストを読み慣れているエディタやライタが，ストップワードを設定すると，コンパクト化率が向上しやすい。だからこそ，ストップワードの設定ファイルを，ヒトが見て理解しやすい XML 文書とすることに意義がある。

●リスト 6-14 は，ある特定分野の論文検索システムのために，実際に用いたストップワードの一部である。約 1000 個のストップワードによって，データファイルのサイズを 3 割までコンパクト化した。

●リスト 6-14 ストップワードを記録した XML 文書の例（ごく一部）。

```
～略～
<word>fig.1</word><word>fig.2</word><word>fig.3</word>～略～
<word>検討</word><word>考察</word><word>推進</word>～略～
<word>同上</word><word>下記の</word><word>上記の</word>～略～
<word>同様の</word><word>これらの</word>～略～
<word>前述のように</word><word>そのような</word><word>あのような</word>～略～
<word>および</word><word>また</word><word>なお</word>～略～
<word>①</word><word>②</word>～略～
<word>Ⅰ</word><word>Ⅱ</word>～略～
<word>（図 1)</word><word>（図 2)</word>～略～
<word>(上段)</word><word>(中段)</word><word>(下段)
～略～
```

6.4 検索条件指定によるフレーズ検索

6.4.1 要約記述の義務付けによるサイズダウン

条件指定検索の目的

フルテキスト検索だけでなく，条件指定によるフレーズ検索も，比較的ニーズの高いものである（本章冒頭●図 6-1 参照）。ネイティブ XML 文書からの条件指定フレーズ検索は，検索項目を指定したうえで，入力あるいは選択された文字列が，検索項目と対応する要素の内容あるいは属性値に完全一致するか，あるいは部分一致する場合に，結果を返す処理である。

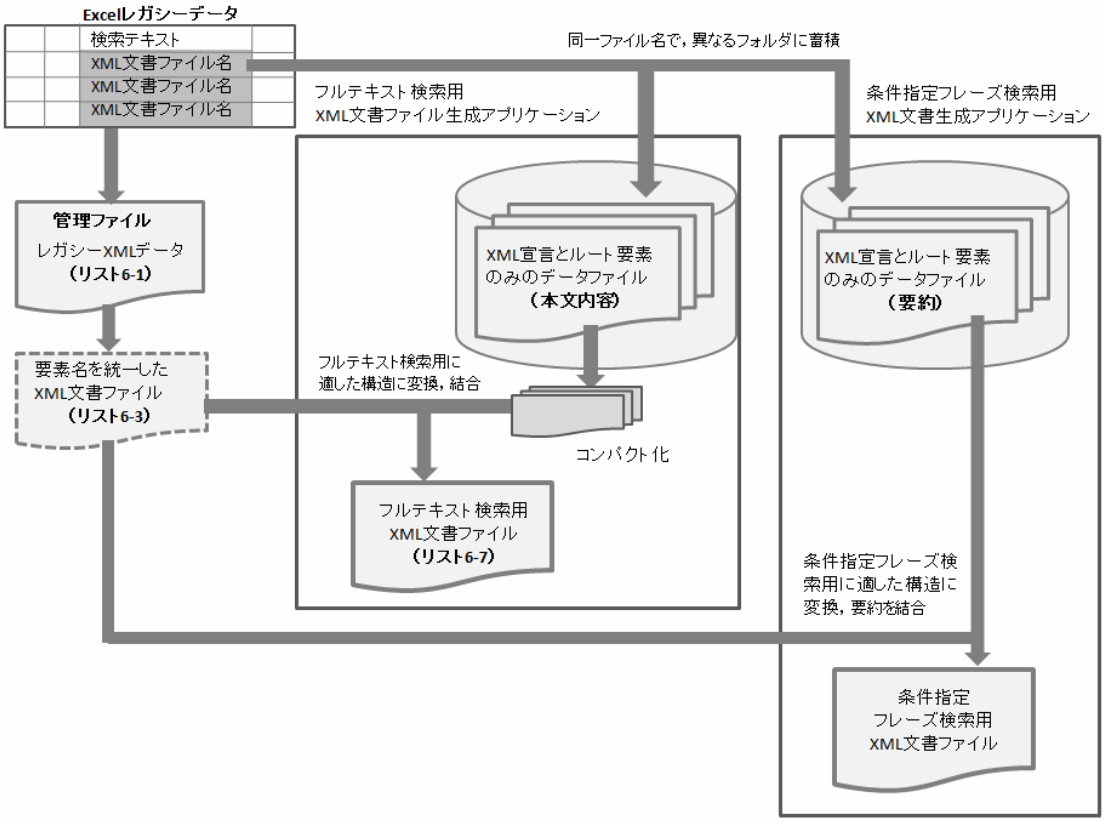
フルテキスト検索がいわゆる「検索文字列のみ指定して，アタリをつける」ために用いられるのに対し，条件指定検索は「細かく条件を指定することによって，確実に目的の文書を探し出す」ために用いられる。ユーザが情報提供側と同程度かそれ以上にデータの内容に詳しく，正確な検索キーを指定できる場合には，条件指定検索は非常に効果的である。

この処理では，該当する語句の有無だけでなく，言葉の並びも，重要である。適合率が高くても，呼出率が低くてはならず，さらに，呼出率が高いだけではなく，ユーザが検索結果に満足する機能を提供しなければならない。

複数の要約と管理ファイルの結合

したがって，検索対象文書の作り方も，フルテキスト検索とは異なる。データファイル中の大量の語句よりも，文書内容を的確に表すフレーズが必要となる。それは，端的で，且つ，ユーザが検索キーとして指定する可能性の高い語句が含まれるものでなければならない。たとえば，書籍でいえば表紙やオビや「はじめに」や各章のトビラに書かれたテキスト，新聞ならばリード，論文ならば要約にあたる部分である。それらには，情報発信者側の意図が凝縮されており，検索キーとして指定される可能性のある言葉が，高い頻度で含まれている。

●図 6-12 Excel レガシーデータから，フルテキスト検索用と，条件指定フレーズ検索用のファイルを別個に作成する。



検索対象とする文書の要約は，文書内容を最もよく理解している人物，つまりデータファイルの元となる原稿を執筆した者が記述すべきである。その際，サイズダウンをはかるために，最大文字数を規定したうえで要約記述を義務付けるとよい。単語単位ではなく，フレーズで検索されるとなれば，情報を発信

する側は，正確且つ，意図を伝えるテキストを，練りに練って書かざるをえなくなる。自ずと，検索対象テキストのクオリティは高まる。データのゴミを生み出さないという意味で，この検索方法の利点は大きい。

フルテキスト検索では，データファイルをコンパクト化した結果のデータを 前掲●リスト 6-7 の<検索テキスト>要素内に入れ込む。条件指定検索では，その代わりに，文書内容を表すフレーズが入ると考えればよい。つまり，各データファイルの要約と，管理ファイルを結合して，単一の検索対象 XML 文書ファイルを生成するのである。

もし，フルテキスト検索と条件指定検索の両方を実装するのであれば，前ページ●図 6-12 のように，Excel レガシーデータから処理方法別の検索対象ファイルを作成すればよい。各データファイルのファイル名は同一とし，異なるフォルダに蓄積しておく。検索対象 XML 文書生成アプリケーションのプログラムコードの中で，それぞれのフォルダを指定してデータファイルを処理するプログラムを記述する。

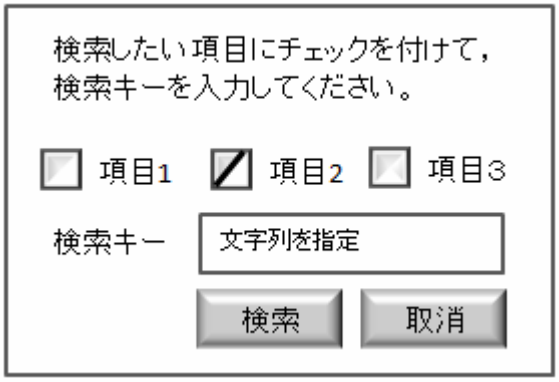
6.4.2 ユーザの選択に対応するプログラム

二者択一の選択肢に対応するデータ作成

条件指定検索において，処理プログラムのコードまで考慮して XML の構造を考えた方がよいケースがある。それは，検索に足るデータを持たない要素の多い文書を対象とし，且つユーザが，データの有無を最優先する場合である。

具体的にいえば，たとえば画面上に 3 個の CheckBox があり，ユーザが「項目 1」と「項目 3」に対応する要素のデータは不要だが，「項目 2」に対応する要素のデータの中から検索キーに合致する結果を得たい，という場合である（●図 6-13）。

●図 6-13 ユーザがデータの有無を条件として指定する，条件指定検索の画面の例



これを実装するには，まず，処理アプリケーション側に，CheckBox などの二者択一機能を持つ選択ボックスをレイアウトする。そして，ユーザが各項目を選択した場合には「1」，選択しなかった場合を「0」として，その組み合わせから成る値を，検索プログラムの変数に代入しておく。

これに対し，XML 文書側では，あらかじめ，検索に足るデータが存在する場合を「1」，存在しない場合を「0」とし，この組み合わせから成る値を，属性値として追加しておく（●リスト 6-15）。これは，疑似インデックスのような役割を果たす。

そして，変数に代入されている数値と，XML 文書中の属性値が，合致するかどうかを検索条件のひとつとするのである。

これにより，もしユーザが「項目 2」のみをチェックしたならば，変数には「010」が代入されるので，「010」という"インデックス"属性値を持つ要素の子要素ノード群が検索対象となる。それ以外の要素は検索対象とはならないので，XML プロセッサに，すべての<要素 1><要素 2><要素 3>を順次走査する処理を負担させずに済む。

ここに挙げた例では，ユーザが期待するデータが存在しない場合の要素の内容を「—」としているが，「未定」「調整中」「検討中」といったようなケースでも同様の処理を実装することができる。

●リスト 6-15 ユーザの期待に沿うデータの有無を示す属性値を追加したデータファイル。

```
<?xml version="1.0"?>
<ルート要素名>
  <要素 A インデックス="110">
    <要素 1>要素 1 のデータ</要素 1>
    <要素 2>要素 2 のデータ</要素 2>
    <要素 3>—</要素 3>
 </要素 A>
 <要素 A インデックス="100">
    <要素 1>要素 1 のデータ</要素 1>
    <要素 2>—</要素 2>
    <要素 3>—</要素 3>
  </要素 A>
  <要素 A インデックス="010">
    <要素 1>—</要素 1>
    <要素 2>要素 2 のデータ</要素 2>
    <要素 3>—</要素 3>
  </要素 A>
  ～繰り返し，略～
</ルート要素名>
```

XSLT による，変換処理に利用する属性の追加

●リスト 6-15 のように，"インデックス"属性値の追加された検索対象文書を作成するには，まず，データがなければ Excel ワークシートに「—」を入力しておく。そして，エクスポートによって作成されたレガシーXML データを，●リスト 6-16 のような XSLT スタイルシートを用いて変換すればよい。

●リスト 6-16 レガシーXML データ変換時に，●リスト 6-15 のような疑似インデックスを追加する XSLT スタイルシートの例。

```
～略～
<xsl:for-each select="dataroot/Excel ワークシートのフィールド名">
  <xsl:variable name="indexNo1">
    <xsl:if test="項目 1 に対応する要素名/text()='—'">0</xsl:if>
    <xsl:if test="not(項目 1 に対応する要素名/text()='—')">1</xsl:if>
```

```
  </xsl:variable>
  <xsl:variable name="indexNo2">
    <xsl:if test="項目 2 に対応する要素名/text()='—'">0</xsl:if>
    <xsl:if test="not(項目 2 に対応する要素名/text()='—')">1</xsl:if>
  </xsl:variable>
  ～略～
  <親要素名>
    <xsl:attribute name="インデックス">
      <xsl:value-of select="concat($indexNo1,$indexNo2,～略～)"/>
    </xsl:attribute>
  </親要素名>
</xsl:for-each>～略～
```

疑似インデックスによるプログラムの簡素化

ユーザが指定する選択肢の数は，たとえば CheckBox が n 個あるならば，n^2 通りである。CheckBox が n 個あるならば，単純に条件分岐による検索処理を記述すると，●リスト 6-17 のように，n^2，つまり 8 通りの条件分岐処理を書く必要がある（●図 6-14）。その点，"インデックス属性"を利用すれば，●リスト 6-18 のように，CheckBox の数の 2 倍だけの条件分岐処理で済む。2n と n^2 では，選択肢が増えるほど，プログラムコードの行数の差も大きくなる。

●図 6-14 CheckBox の選択パターンと対応させる疑似インデックス

	CheckBox1 要素1	CheckBox2 要素2	CheckBox3 要素3	属性
1	◪	◪	◪	インデックス="111"
2	◪	◪	□	インデックス="110"
3	◪	□	◪	インデックス="101"
4	◪	□	□	インデックス="100"
5	□	◪	◪	インデックス="011"
6	□	◪	□	インデックス="010"
7	□	□	◪	インデックス="001"
8	□	□	□	インデックス="000"

●リスト 6-17 ユーザがチェックした項目の組み合わせを網羅する条件分岐処理（言語は VB）。

```
If CheckBox1.Checked And CheckBox2.Checked And CheckBox3.Checked Then
        ～検索処理～
End If
ElseIf CheckBox1.Checked And CheckBox2.Checked Then
        ～検索処理～
ElseIf～略～
Else
～略～
End If
```

●リスト 6-18 "インデックス"属性を利用すると，CheckBox の数×2 種類の条件分岐で済む。

```
～略～
If CheckBox1.Checked Then
  checkNo1 = "1"
Else
  checkNo1 = "0"
End If
If CheckBox2.Checked Then
  checkNo2 = "1"
Else
  checkNo2 = "0"
End If
If CheckBox3.Checked Then
  checkNo3 = "1"
Else
  checkNo3 = "0"
End If
～略～
"インデックス"属性値を代入する変数 = checkNo1 & checkNo2 & checkNo3
～略～
```


以上のように，検索対象データの構造を工夫することによって，文書ファイルのサイズダウンをはかり，処理の無駄を防ぐことができる。

6.4.3 練習から汎用的な実践へ

本章でとりあげた方法は，プログラミングや XSLT コーディングを要する力技ではある。だが，現実に即しているので，ヒントにはなるであろう。実践では，思い描いたように事が運ぶことは，まず，ないといっていい。工期や担当者のスキルや予算，更新頻度や更新時間といった制約があり，その中で企画設計，実装を行うことになる。「～ねばならない」をいう思考の枠を外すことも，時には必要である。

もし，同じようなタイプの業務を何度も受注する可能性が高いのであれば，一度 XML 文書生成処理ならびに検索処理を開発したその後で，汎用化できる部分がないか見直してみるとよい。汎用的な XSLT スタイルシートやプログラム（あるいは DOM と XSLT ではなく LINQ を用いたプログラム）を用意しておくことにより，それ以降の業務を効率化することができることがある。

たとえば，これまでの例に挙げた XSLT スタイルシート（●リスト 6-7，●リスト 6-9）を見れば，属性が要素に対して不規則に従属するのでない限り，基本的な要素の生成コードには規則性があることが分かる。つまり，レガシーXML データ中にフラットに並ぶ要素名（Excel ではフィールド名にあたる）のローカル名を取得して変換するような，汎用的な XSLT スタイルシートを作成してみるのも一手であろう。

また，もし，商品データベースや顧客データベースなど，複数のテーブルから成る，既存の RDB レガシーデータの XML 化が必要なのであれば，参照側のデータを複数の XML 文書として作成し，参照元の XML 文書に結合する方法を試してみる価値はある。完了した業務や進行中の業務の経験を，次回以降の業務に生かすことができないか探る姿勢が大切である。

その昔，XML1.0 勧告から数年間は，セキュリティや排他制御などの問題を抜きにしても，ネイティブ XML 文書は，データベースとしては使いものにならないと考えられていた。なぜなら，DOM を用いると処理速度がメモリに大きく依存し，プログラムコードの記述方法ひとつで処理時間に 20 倍近い差が生じていたからである。

だが今では，ハードウェアのスペックが向上し，ネイティブ XML 文書は利用しやすくなっている（※12）。案件の規模や諸条件によって，ネイティブ XML 文書，RDB，XML データベースを使い分けるとよいだろう。

※12：筆者の経験からいえば，たとえば，約 50000 ファイル 4 階層 150 万ノードからのフルテキスト検索では，ASP.NET2.0 時代のアプリケーションで全件表示してもストレスはなく，検索の多重度が上がるほど，処理時間は短縮された。その 1/10 の 15 万ノード程度であれば，セッション時間と最大同時アクセス数にもよるが，3 世代前のロースペックのマシンで動かしても速度面の問題はまず生じないといっていい。ただし，XML 文書中のノード数が一定数を超えると，ネイティブ XML 文書よりも，ミドルウエアを利用する方が処理速度は向上する。

また，Web アプリケーションではなく，ローカルアプリケーションであれば，1XML 文書ファイルあたりのノード数が小さいならば，ファイル数は多くても，処理は可能である。億単位のノード数のネイティブ XML 文書を扱うことも可能である。

いずれ，テラバイト級のネイティブ XML 文書処理も可能になるであろう。

6.5 設計後の工程に影響する問題

6.5.1 データ作成にかかわる問題

Excel によってレガシーデータを作成し，検索アプリケーションにおいて，検索キーとして指定した文字列の，全角・半角，大文字・小文字を厳密に区別した処理を実装するのであれば，データ入力には一層の慎重さがもとめられる。特に数字の半角と全角，アルファベットの形が類似の大文字小文字（s，c など）の入力ミスは発生頻度が高く，Excel ワークシート上で見逃されやすいうえ，検索アプリケーションのテストでも見過ごされやすい。万が一それがデータファイル名であれば，検索対象文書ファイルが正確に生成されない事態となる。

また，Excel のデータ後に空白が入っていると，不要なデータを増やすことになるので注意する。この空白は，意図的にドラッグして確認しない限り気付かれず，そのまま XML 化されてしまう可能性が高い（●図 6-15）（※13）。

さらに，データ件数が多い場合の入力モレ，類似データがある行の重複，連番の入力ミスにも注意する必要がある。年や月，連番などを入力する際に，ドラッグ操作とコピー＆ペーストの使い分けを誤るケースもある。レガシーデータの入力時にミスがあれば，連番を一意なキーとして XSLT 変換における判別処理に利用すると，誤った変換結果を招くことになる。

●図 6-15 データ後の空白は，空白入りのままエクスポートされる。反転部分は空白である。

B2　在宅支援[redacted]

	A	B	C	D	E
1	年度	支援制度	テーマ	内容	詳細
2	平成21年度	在宅支援	プログラミング	入門	
3	平成21年度	在宅支援	プログラミング	基本	
4	平成21年度	在宅支援	プログラミング	応用	

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
  - <講座>
      <ID>1</ID>
      <年度>平成21年度</年度>
      <支援制度>在宅支援                    </支援制度>
      <テーマ>プログラミング</テーマ>
      <内容>入門</内容>
      <パス>2009</パス>
      <コンテンツ>pg_01.htm</コンテンツ>
      <検索テキスト>pg_01.xml</検索テキスト>
      <所属1>女性技術者の会</所属1>
      <氏名1>古見　久美</氏名1>
    </講座>
  - <講座>
```

※13：Excel2002 では空白は CDATA セクションとしてエクスポートされたが，Excel2003 以降では字数分の空白が入る。

6.5.2 アプリケーションの設計と実装にかかわる問題

処理に必要な値の指定

XPath のコンテキスト･ポジションによるノード指定は，実務で頻繁に用いられる。その際には，軸の方向に注意する必要がある（●図 6-16）。

前進軸では，文書順に 1，2，3．．．と指定すればよいが，後進軸では，コンテキスト･ノードから逆文書順に（ルートに向かう方向に）1，2，3．．．と数える（第 3 章●図 3-2 も参照）。これを誤って，前進軸同様，文書順に数えないように注意しなければならない（※14）。

●図 6-16 軸の方向とコンテキスト･ポジションに注意する

```
<?xml version="1.0"?>
<講座実施報告書>
        <目次>1月度</目次>      ↑  preceding-sibling::目次[2]
        <目次>2月度</目次>      ↑  preceding-sibling::目次[1]
        <目次>3月度</目次>
        <目次>4月度</目次>      ↓  following-sibling::目次[1]
        <目次>5月度</目次>      ↓  following-sibling::目次[2]
        <目次>6月度</目次>      ↓  following-sibling::目次[3]
</講座実施報告書>
```

＊14：ただし，Microsoft LINQ to XML では，前の兄弟を指定するために用いる ElementsBeforeSelf も前進軸であるので，W3C 仕様と混同しないように注意する必要がある。

処理に必要な値の取得

Web アプリケーションから利用する単一の XML 文書では，通常，新しいデータは末尾へと追加される。しかし，選択肢の表示は新着順とすることもある（●図 6-17）。その場合は，選択項目のインデックスと，処理対象とするノードリストのノード数から，処理に必要な値をもとめる必要がある。

このような処理を実装した後で，中途にノードが追加されると，インデックスとコンテキスト・ポジションの間に，不整合の生じる恐れがあるので注意する。

●図 6-17 選択項目のインデックスに対応する，処理に必要な値をもとめる

選択項目の値	処理に必要な値			
プログラム	XPath		DOM	
インデックス（n）	総ノード数（a）	コンテキスト・ポジション（p）	総ノード数（a）	インデックス（i）
0	6	6	6	5
1	6	5	6	4
2	6	4	6	3
3	6	3	6	2
4	6	2	6	1
5	6	1	6	0
もとめ方	fn:count()	fn:position()	length	NodeList.item(index);

スキーマ変更の影響の抑制

要素のローカル名の候補が複数あって，なかなか決定にはいたらないが，工期上開発に着手せざるをえない場合，要素名変更の影響を受けないように，要素名を直接用いずに XSLT スタイルシートを記述することがある。●リスト 6-19 では，local-name 関数と，position を組み合わせて用いて，コンテキスト・ポジションが「n」である子要素の要素名と内容を取得している。

ただし，要素名を直接指定していないために，設計者以外の者には分かりにくい。チーム開発や XML 文書作成を外部委託するのであれば，出現順序と要素名の対応表を仕様書に記載しておく必要があるだろう。

●リスト 6-19 名前変更やノード増減の影響を避けるために，local-name 関数と，position を組み合わせて用いることがある。n には任意のコンテキスト･ポジションを指定する。

```
<xsl:for-each select="親要素名">
～略～
  <xsl:value-of select="local-name(*[n])" /></th>
  <xsl:value-of select="child::*[position()=n] " />
～略～
</xsl:for-each>
```

フルテキスト検索の文書構造に対する誤解

定型の管理ファイルと複数のデータファイルからの検索では，両者のルート要素以下のすべてのテキストを検索対象とし，定型部分のデータを結果として返す。検索処理に限っては，データファイルのルート要素以下の構造の変更が，プログラムに影響を及ぼすことは基本的にはない。これは XML を採用する利点でもある。

だが，設計前の打ち合わせにおいて，この「プログラムに影響を及ぼすことは基本的にはない」と強調しすぎると，「XML は柔軟であるから，妥当でなくてもよい整形式文書，いわゆるスキーマレス文書であれば，いかなる構造の変更にも耐えうる。アプリケーション側への影響は一切生じないので，設計後に構造が変わっても問題ない」という誤解を，顧客に与えかねないので注意する。

定型の管理ファイルに構造変更があれば，検索処理自体に影響はなくとも，検索キーを指定する処理や，結果を表示する処理に変更が及ぶ。

6.5.3 更新・運用・作業管理における問題

頻繁な更新による応急処置

検索対象 XML 文書ファイルを作成した後に，わずかな誤字脱字といった小規模な訂正が発生し，迅速な作業を要求されると，XML 文書ファイルのデータのみを修正しがちになる。だが，いかなる場合であっても，Excel レガシーデータから修正すべきである。一度でも，Excel レガシーデータと検索対象文書ファイル内データの不整合が生じると，その不整合の修正に何倍もの時間を費やすことになる（前章●図 5-32 も参照）。

また，データの更新作業中に，担当者の業務多忙などにより，一部データの公開を延期するというケースがある。整合性を維持するためには，公開延期データを Excel レガシーデータから削除して再作成すべきであるが，緊急性の高い場合は，検索対象文書ファイル中のデータをコメントアウトして，表示を一時保留とする応急処置も考えられる。

ただし，その場合，注意しなければならないことが 1 点ある。XML 文書中のコメントのネストはできないということである（●リスト 6-20）。コメントのネストは，エラーを引き起こす（●図 6-18）。

●リスト 6-20 「講師」が確定しないためにコメントアウトし，さらに「公開テキスト」の執筆が遅れるためにコメントアウトした結果，コメントがネストされている。

```
<?xml version="1.0" encoding="utf-8"?>
<実施計画>
  <講座>
    <年度>平成 21 年度</年度>
    <支援制度>～略～</支援制度>
    <支援制度>
      <支援制度名>就労促進</支援制度名>
      <テーマ>
        <テーマ名>デザイン</テーマ名>
        <内容>
          <タイトル>入門</タイトル>
          <!--<詳細>
            <見出し 公開テキスト="2009design_01.htm">レイアウト</見出し>
            <!--<講師>
              <所属>XML 株式会社</所属>
              <氏名>江楠　絵夢</氏名>
            </講師>-->
          </詳細>-->
          <詳細>～</詳細>繰り返し，略
        </内容>
        <内容>～</内容>繰り返し，略
      </テーマ>
    </支援制度>
  </講座>
</実施計画>
```

●図 6-18 XML ソース中のコメントのネストはできない。

最大フィールド数の増加

前述の「6.1 深い階層の子孫要素の表現」では，Excel ワークシート上に予測される最大出現回数分のフィールドを設けておき，必要データのみを入力した。ところが，長年運用するアプリケーションでは，

当初予測以上のデータが生じることもあろう。すると，XML データ生成アプリケーションのプログラムコードによっては，最大フィールド数が増えたことにより，正常に動作しなくなったり，誤った XML 文書が作成されることにつながってしまう。

当初予定以上に最大フィールド数が増えた時には，プログラムコードを再確認しなければならない。そのためにも，プログラマには，見通しが良くカスタマイズしやすく適切なコメントの挿入されたコードを記述するよう依頼しておこう。

開発後の中途階層追加

運用開始後の階層変更は，処理アプリケーション開発に大きな影響を及ぼすことがある（第 4 章●図 4-17 も参照）。最初のページからクエリ文字列を渡していれば，以降すべてのプログラムにわたって処理の変更が生じたり，ユーザの使い慣れたインタフェースも変えざるをえない状況に陥る。

実務では，処理アプリケーションのプログラムコードを変更すると，動作に支障をきたす恐れがあるので，変更よりも処理の追加によって対応できないかを検討する。

たとえば●図 6-19 では，「助成方法」フィールドが中途階層に追加されている（アミ部分）。このケースでは，●図 6-20 のように，属性値として追加し，表示処理を追加するという措置をとる。

中には，●図 6-21 のように，複数のフィールドが中途階層に追加されることもある（アミ部分）。複数階層の追加では，引き続き何らかの追加が生じる可能性があるので，●図 6-22 のように，別の XML 文書ファイルを作成して対応する方法を検討する。アプリケーションへの影響を最小限にとどめることが，XML 化の目的を損なう構造を容認することにつながるので，設計者から見れば苦肉の策ではある。

中途階層ではなく，最下層が追加された場合は，XSLT スタイルシートの変更だけで済むため，プログラムコードへの影響を阻むことができる。

変更が予想される案件では，要素主体の設計とし，クッションとして階層を追加しておく方法もあるが（第 5 章「クッションとなる要素の追加」参照），予想外の場所に階層が追加されると，クッションは役に立たない（※15）。

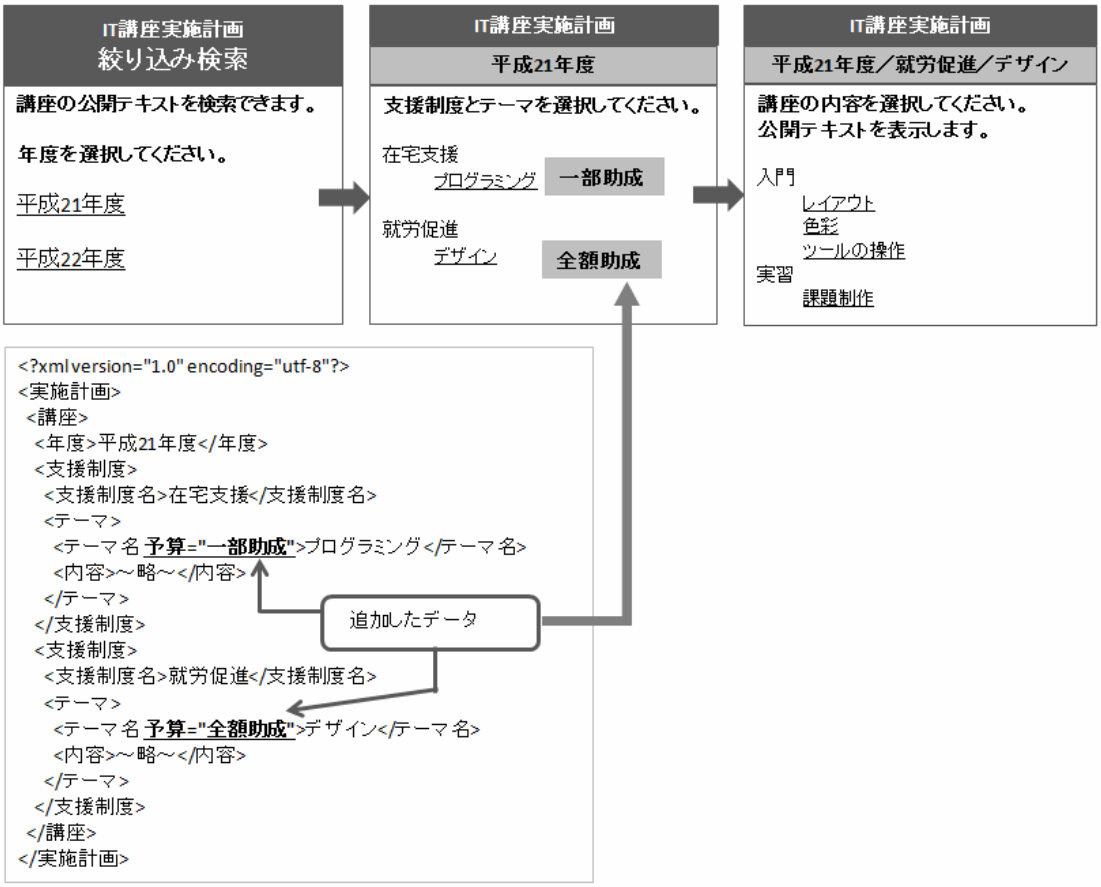
以上はあくまで応急処置であって，基本的には，開発初期段階のディスコミュニケーションを防ぎ，的確な予測によって，変更が生じる事態を避けるべきであることを肝に銘じておこう。

＊15：木構造がいかに変更されようと，処理アプリケーションに影響を及ぼさないようにするには，処理アプリケーションのプログラムを汎用的な設計とするしか方法はない。XML 文書ファイルを読み込んで，その構造を解析しながら処理を行うのである。だが，最大階層の数を固定しないだけでも，XML 文書を読み込む度に，階層数の取得を実行しなければならず，プログラムコードの複雑化は免れえない。また，XML 生成アプリケーションにおいても，できる限り汎用的な処理を目指す姿勢は重要であるが，階層数を無制限とする設計は避けるべきである。n 階層の木を生成するために，再起処理を繰り返す実装を行うことはリスクが大きすぎる。

●図 6-19 運用開始後に「助成方法」フィールドが中途階層に追加されている。

年度	支援制度	テーマ	助成方法	内容	詳細
平成21年度	在宅支援	プログラミング	一部助成	入門	
平成21年度	在宅支援	プログラミング	一部助成	基本	
平成21年度	在宅支援	プログラミング	一部助成	応用	
平成21年度	就労促進	デザイン	全額助成	入門	レイアウト
平成21年度	就労促進	デザイン	全額助成	入門	色彩
平成21年度	就労促進	デザイン	全額助成	入門	ツールの操作
平成21年度	就労促進	デザイン	全額助成	実習	課題制作
平成22年度	在宅支援	データベース	一部助成	概論	
平成22年度	在宅支援	データベース	一部助成	実践	論理設計
平成22年度	在宅支援	データベース	一部助成	実践	物理設計

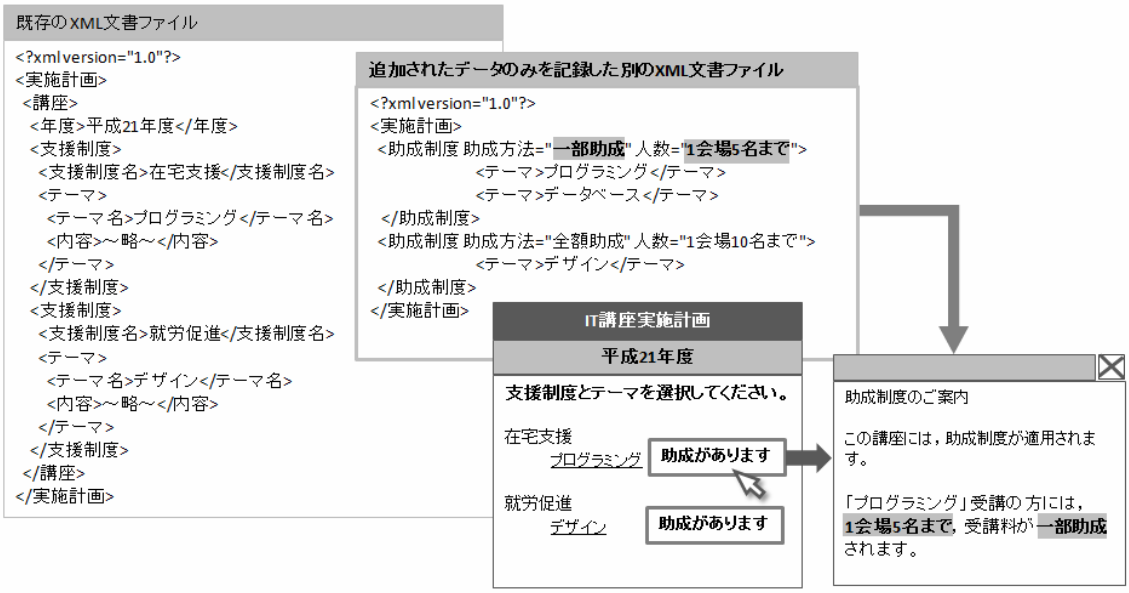
●図 6-20 属性値として追加し，表示処理の追加で対応する方法。



●図 6-21 運用開始後に「助成方法」「支援枠」フィールドが中途階層に追加されている。

年度	支援制度	テーマ	助成方法	支援枠	内容
平成21年度	在宅支援	プログラミング	一部助成	1会場5名まで	入門
平成21年度	在宅支援	プログラミング	一部助成	1会場5名まで	基本
平成21年度	在宅支援	プログラミング	一部助成	1会場5名まで	応用
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	入門
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	入門
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	入門
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	実習
平成22年度	在宅支援	データベース	一部助成	1会場5名まで	概論
平成22年度	在宅支援	データベース	一部助成	1会場5名まで	実践
平成22年度	在宅支援	データベース	一部助成	1会場5名まで	実践

●図 6-22 別の XML 文書ファイルを作成して，追加の処理で対応する方法。



想定外データの追加

先に述べたように（●図6-7参照），運用開始後，同一データが追加される可能性については，よくよく注意しておかなければならない。

たとえば，レガシーデータの元原稿をそれぞれ複数の担当者が執筆し，互いの執筆内容を知らないまま，完全に同じ見出しを付けてしまうという，信じられないような偶然も起こりうるのである。簡単な例をあげれば，「企業における Web システムのセキュリティとリスク管理」をテーマとする「ネットワーク理論に基づくウィルスの拡大における一考察」という見出しの記事をデータとして追加した後，別の筆者による「新型インフルエンザと医療体制」をテーマとした，同じ「ネットワーク理論に基づくウィルスの拡大における一考察」という見出しの記事を追加するといったことが起こりうる。

だからこそ，処理対象 XML 文書生成をアプリケーションによって自動化している場合は，あらかじめ重複を見越した処理を記述しておかなければならないのである（前掲●リスト 6-7 参照）。とりわけ，多階層の最下層の節の小見出しのデータの重複可能性には注意しておかなければならない。

もし，全データが一意なものであるならば，最小単位となる要素に主キーを振って処理する方法を検討するとよい。

もう一つ，注意しておかなければならないことがある。検索処理において，既存データの文字列を含む新規データが追加されると，目的通りの結果が得られないという問題の生じる可能性である。たとえば，次ページ●図 6-23 のように，C 列の「テーマ」に，「インタラクティブデザイン」という新規テーマ（アミ部分）が追加されたとする。

処理プログラムのコードが，●リスト 6-21 のように記述されていれば問題はない。ところが，●リスト 6-22 のように記述されていると，「インタラクティブデザイン」のみ検索したつもりでも，「デザイン」を選択して得られる結果までもが返されてしまう。これは，XML 設計に関わる問題というよりも，処理プログラムに関わる問題ではある。だが，XML 設計者は，AND 検索や OR 検索，完全一致検索や部分一致検索といった処理目的によっては，プログラムコードの中での条件指定が異なるということを念頭に置いた上で，追加されるデータとその処理結果を予測して，仕様書を作成しておかなければならない。

●図 6-23 既存項目の文字列を含む新規項目が追加された例。

	A	B	C	D	E	F
1	年度	支援制度	テーマ	内容	詳細	パス
2	平成21年度	在宅支援	プログラミング	入門		2009
3	平成21年度	在宅支援	プログラミング	基本		2009
4	平成21年度	在宅支援	プログラミング	応用		2009
5	平成21年度	就労促進	デザイン	入門	レイアウト	2009
6	平成21年度	就労促進	デザイン	入門	色彩	2009
7	平成21年度	就労促進	デザイン	入門	ツールの操作	2009
8	平成21年度	就労促進	デザイン	実習	課題制作	2009
9	平成21年度	就労促進	インタラクティブデザイン	入門		2009
10	平成21年度	就労促進	インタラクティブデザイン	基本		2009
11	平成22年度	在宅支援	データベース	概論		2010
12	平成22年度	在宅支援	データベース	実践	論理設計	2010
13	平成22年度	在宅支援	データベース	実践	物理設計	2010

●リスト 6-21 検索対象ノードの内容に，検索キーを代入した変数が完全一致する場合の処理（VisualBasic の場合）

取得する親ノード[検索対象ノード="" & 検索キーを代入した変数 & ""]

●リスト 6-22 検索対象ノードの内容に，検索キーを代入した変数が部分一致する場合の処理（VisualBasic の場合）

取得する親ノード[検索対象ノード=contains(.," & "" & 検索キーを代入した変数 & "")]

XML はインターネット対応であるために，顧客側やユーザ側には，「ブログのように簡単に，データの編集や削除ができるはずである」という意識が働く。だが，運用開始後の想定外データの追加は，設計者があらかじめそれを予測して対応策を講じておかない限り，問題を引き起こすことになる。だからこそ，正確な予測をするためにも，前章「設計前の作業～仕様調査～」で述べたように，「設計に必要な最低限のデータ」を揃えることが重要なのである。

データ管理における意識

企画段階から実運用までに作成された仕様書や文書ファイルは，運用期間中，安全に保管しておかなければならない。

実運用に用いる XML 文書ファイルについては，データの破損やディスク障害に備えるためにも，定期バックアップだけでなく，随時バックアップが必要である。データ更新時に，手動で異なるメディアにコピーするか，あるいは，XML 文書生成アプリケーションにファイルのバックアップ機能を実装しておき，これを用いる。ファイル名は，タイムスタンプを連結したものとし，特定のフォルダに自動的に保存していくようにするとよい。XML 文書ファイルを一度読み込み，更新日時と担当者 ID の連結した値を，ルート要素の属性値として付加して保存する方法でもよいだろう。

最も重要なのは，Excel レガシーデータと XML 文書生成アプリケーションのプログラムファイルである。この 2 つさえ管理できていれば，万が一の場合でもデータを復旧することができるので，最悪の事態は避けられるだろう。

必要ファイルの保存だけでなく，逆に，不要ファイルを蓄積しないよう注意する必要もある。特に，管理ファイルと複数データファイルによる構成では，管理ファイルに記録していなければ処理対象とはならないため，活用方法が不透明なデータファイルまでもが，手当たり次第に蓄積される恐れがある。

XML 設計者は，データ爆発への理解と自制心を持ち，使わず，管理できず，削除もできないデータが蓄積されないよう，設計書の添付資料として，データ作成・管理の標準化方法について明記しておくべきである。

XML 生成アプリケーションや処理アプリケーションで DOM を用いて木を生成する場合は，メモリ上に展開するだけなのか，あるいは一時的に自動保存するのか，永久に蓄積するのかも，検討しておく必要がある。アプリケーション開発者にとっては，すべての木を背後で保存していく方が，処理の途中経過を確認しながらプログラミングでき，間違いがない。だが，一連の操作が終了した時点で，一時的に生成したファイルは削除する処理を記述しておかなければ，操作の都度，無制限にファイルが蓄積されてしまうことになる。XML 設計者とアプリケーション開発者の間での刷り合わせが必要である。

コードのレビューとテスト

XML 設計者が最善の構造を考えたつもりであっても，XML データ処理アプリケーションのテスト時に，バグの発生に遭遇することがある。

テスト時に気付くボトルネックは 4 つ。XML の構造，指定したパス式，プログラムの構造，ノード数である。それらを 1 個ずつ詰めていけば，障害の原因は自ずと明らかとなる。

多くの場合，設計用のサンプル・データを用いてプロトタイピングを行い，正式データに置き換えてテストを行った段階で，バグが生じる。

特に不定形多階層の文書では，サンプル・データの存在する階層の処理しか記述していなかったために，すべての階層にデータが追加された実データを用いると，途中までしか処理されないということが起こりうる。設計者は設計書を作成するだけではなく，実装担当者が異なる解釈をしないよう，口頭でも十分な打ち合わせをしなければならない。

また，サンプル・データと実データのデータ量（処理対象ノード数）が圧倒的に異なるために，正式データに置き換えた途端，レスポンスが劣化するということがある。

これを防ぐには，あらゆるパターンのサンプル・データを作成し，データ件数に不足があれば，コピー＆ペーストして実データと同じ件数まで増やし，これを使って設計を検討したり，プロトタイピングを行うとよい。

ただし，もっとも望ましいのは，実データを用いたプロトタイピングである。

構造があまりにも複雑な XML 文書を扱う場合は，XML 設計者とアプリケーション開発者がペアプログラミングを行ったり，あるいは設計者がコードレビューを引き受けるのも，バグを防ぐための一方法である。

報連相と文書化で防ぐデータの不整合

一文書の作成に複数のスタッフが関わるケース，とくに遠隔地のスタッフがプロジェクト・チームを結成して開発にのぞむ場合は，整合性の維持をはかるために，文書による報連相が不可欠となる。

たとえばマニュアルや論文の XML 化では，ライタ（レガシーデータの元となる原稿作成），エディタ（構成および校正），デザイナ（図表やイラストの作成），更新担当者（レガシーデータからの XML 文書作成）が関わる。それら複数の担当者が連携しないまま作業を行うと，トランザクションにも似た問題が発生する。つまり，最新の文書の所在が曖昧になるのである。また，メールの添付ファイルに修正依頼内容を

書いて送受信し合い，別のフォルダや，別のパソコンにコピーして用いて，最新の依頼を失念するといった事態も起こりうる。

こういった問題を未然に防ぐには，データ更新作業のリーダーが（最も望ましいのは設計者が兼務することである），作業手順書を作成して，一か所から指示を発するとよい。短期間に修正依頼が相次ぐ場合は，逐一指示を発するのではなく，一定期間保留してから連絡する。

作業指示書には，作業の記録（作業日時，担当者名，作業内容，作業時間，作業後の結果と確認）および，更新の記録（更新予定日，更新日時，更新内容，担当者，バックアップファイルの保管場所及びバックアップ方法，最新レガシーデータおよび XML 文書ファイル名，タイムスタンプ，ファイルサイズ）を明記しておく。また，XML 設計の変更，XML データ作成方法の変更，データ処理アプリケーションの機能変更や機能追加，アプリケーションの GUI のリデザインを行う場合も，その詳細を文書として残しておく。

これらの文書作成は一見手間ではあるが，必要な作業である。もし，設計者が，5 年 10 年分の作業記録をすべて記憶できるとしても，チーム開発では，一人の人間の記憶に依存することは避けなければならない。そして，文書を作成しても，その文書を理解する者が設計者一人であってはならない。

また，更新担当者には，示唆呼称をしながら作業を進め，完了した手順にはチェックを付ける（手順書をプリントアウトしない場合は画面上で色を変えるなどを実践する）ように義務付けることが望ましい。

重要なのは，話し言葉と以心伝心だけに頼らず，仕様書を基本とした共通認識に基づいて，一手順ずつ確実に作業を進めていく姿勢である。

完璧でないからこそ，完璧を目指す

XML 化を成功させるには，XML 設計者が万全の設計を行うだけでなく，その後の工程の作業も万全でなければならない。

だが，アプリケーション開発に限らず，機械であれ建造物であれ，理論が完璧であり，全工程に携わる者が「100%」誠心誠意を尽くして作り上げたとしても，100%完璧であるという保証はない。100%の保証がないことを肝に銘じて，完璧をめざして努力するのが，プロであろう。

企画段階でのヒアリング漏れ，設計段階での予測ミス，実装段階でのバグといった問題が一切なく，完璧なシステムを開発できたとしても，データ更新時の入力ミスや，ユーザの操作ミスに対するサポートの困難なケースがあるかもしれない。また，木構造に影響を及ぼす社会問題や，災害などによるハードウェアやネットワークの障害が生じるかもしれない。開発したシステムが正常に動作しても，それで完璧とはいえない。何の問題もなく運用期間が終了して初めて，完璧であったといえるのだ。

筆者の場合手がけた案件で問題の発生したことは一度もないが，もし仮に，万が一何か問題が発生したとしても，設計者に必要なのは，発生しつつある問題を正確に評価して，将来的なトラブルを予測し，論理的思考で，すみやかに解決することであろうと思われる。そして，どの段階で，どのような誤りが発生しやすいかを，文書化して後進に伝えることであろう。

6.6 アルゴリズム的手法を超えて

6.6.1 本書を超えて，手順なき実践の世界へ

入門者は手順に従い，経験者は俯瞰する

おそらく皆さんがたずさわる案件の多くは顧客あっての受託開発であろうから，広義の設計における企画や調査の工程は，手順にしたがって確実に進めることが必要となる。だが，狭義の設計は，設計者の脳内でなされるものである。そのような脳内作業については，じつのところ，手順にしたがうことが最短最良の方法であるとはいえない（広義の設計，狭義の設計については，第 5 章●図 5-1 を参照）。

狭義の設計に初めて取り組む場合は，まずは，本書を参考に，一手順ずつ確実に設計作業を進めてみるとよい。きっと，その過程で，手順に沿って各作業を進めることの難しさに気付いて困惑するに違いない。階層構造を決定し，タグ名を検討していると，階層構造の問題に立ち返る必要が生じる。ファイル構成を決定したが，今度は各ノード間の整合性がとれなくなった———。行きつ戻りつの作業を延々と続け，どの段階をもって設計完了とすべきか判断に苦しむことになるだろう。そして，「あらゆる問題を横断的に検討しながら進める必要があるのなら，手順なんぞあってないようなものではないか！」「XML 設計もデータ処理アプリケーション設計も，どの作業から始めてもよいではないか！」と思うに違いない。

しかし，設計作業を終え，仕様書作成にいたったとき，眼の前には，設計結果の木が立ち現れていることに気付くだろう。そうしたら，もう次回の設計からは，手順にこだわる必要などない。「各々の手順を順番にクリアしていかなければならない」という考えは捨ててよい。

実際，全てのステップをクリアして初めて，設計結果の木が立ち現れるわけではないのである。設計を始めるとき，構築すべき木は，既にある。つまり，設計者が行うべきは，設計手順を 1 段階ずつクリアしながら到達点を目指すことではない。第 4 章でも述べたように，真っ先に，上空から行く手にある光景を俯瞰して，設計結果の木に気付いてしまうことである。

そして木の形を脳裏に焼き付けたなら，その木が「良い設計である」という裏付けを，XML 設計とアプリケーション設計の両面から検討し，言葉や図を用いて仕様書に定着させていく。それは，顧客やデータ入力担当者やアプリケーション開発者に対して，見い出した木の合理性と正当性を説明する必要があるからだ。

言語は思考の道具としては不完全なので，言語で思考したところで，設計結果の木を俯瞰できる可能性は（可能性がないとは言わないが）高いとはいえない。「これこそ最良の木である」という感覚は，非言語的なものである。したがって，そのままでは他者には伝わらない。思考を具体化して伝達するには，言葉に置き換え，言葉を尽くして説明するしかないのである。

本書で，設計の手順を解説しているのは，書籍が「シーケンス」を表現する媒体だからであって，決して，設計に手慣れてきた後でも本書の手順を踏むよう勧めているわけではない。手順通りに進めれば答えにたどり着けるような，一方向の計算にもとづく作業は，いずれコンピュータが取って代わる。複数の連携する木の構造を併行して考えたところで，それも単なる並列処理に過ぎない。設計者の価値は，ヒトだからこそできる思考にこそある。

昨今の科学の進歩はすさまじい。とりわけ，脳科学は，恐ろしい勢いで発展している。だからであろうか，「ヒトによる設計業務は不要になると思うか？」という質問を，しばしば受ける。その懸念は，もっともである。事実，いまや記憶すべき情報が多すぎて，大量の情報の迅速な処理が必要であり，コンピュータの方がヒトより遂行能力の高い仕事も多い。AI の進歩により，ヒトでなくとも，こなせる作業はどんどん増えていくだろう。

習得すべき技術は，これからもおそろしい勢いで変わり続ける。だからといって，第 4 章でも述べたように，さしあたり皆さんが現役である間，ヒトによる設計業務のすべてをコンピュータがとって代わるわけではない。需要は減っていくであろうし，作業内容も変化していくが，ヒトの能力が不要になることはない。

適切な判断に基づく木を植えよう

その昔，データベース化の対象は文字列と数値だった。それからベクトルデータを扱えるようになり，ラスタデータを扱えるようになり，今では動画像も扱うことができる。近い将来，図形はもちろん，テキストや音も，イメージそのものとして扱われるようになっていく。そして，イメージ記憶のデータベース化が進み，脳内イメージを検索キーとして指定し，情報を得られるようになっていく。

このリアル世界では，年々，テキストよりも，イメージを喚起する視覚情報に重きが置かれるようになっている。この傾向が覆されることはない。ヒトの外部にあるデータをトリガとして，個々のヒトの記憶を検索したり，逆に，個々のヒトの中のイメージをトリガとして，外部にあるデータを検索する処理も一般化していく。

こういった世界の変化につれ，標準フォーマットもまた変化する。現在の XML が，永遠に最上の標準フォーマットであり続けることはない。新しいフォーマットが登場すれば，それまで蓄積されていた XML データは，そのときどきの時代の標準フォーマットに変換され，新しい形の資産として利用されていく。そして，現在の処理対象 XML 文書は，遺された始原データ，すなわちレガシーデータとして生まれ変わる。我々は，未来のレガシーデータの設計者なのである。

さいごに。いま一度，第 4 章で述べたことを思い出してほしい。

XML 設計とは，「的確な予測と判断に基づいて，その構造をデザインすること」である。設計者がこの社会に対してできることは，ヒトでなければできない的確な予測と判断に基づいて，確実に根を張るような構造を持つ苗木を，1 本ずつ植えていくことだけなのである。

エピローグ

ここ数年，筆者は，既存 XML アプリケーションのリメイクについて，しばしば相談を受けてきた。そして，作成済みの XML 文書を見ては当惑することがあった。「整形式の文書を作成できること＝XML 設計をできるということ」という誤解があるように思えてならないのである。

筆者の技術宣伝に対する考えは，「敷居は低く，壁は高く」―――技術に触れるための敷居はできるだけ低くして「面白く！楽しく！賑やかに！」裾野を拡げた方がよいが，成果物の品質を保つためには壁を高く保つべき―――なので，実務者になるための壁が低くなりつつあるように見える状況を危惧する。

もし，XML 設計が，趣味のコンテンツ作成や学習，所属するサークルや団体や自社 Web サイトへの利用を目的とするものならば，まずは仕様書に目を通さなければ，などと気負う必要は全くない。入門書を斜め読みし，サンプル・コードを真似して，見よう見まねで気軽に取り組む方がよい。「楽譜を読めるようになってからでなければ，歌を歌って楽しんではいけない」，などというのは無粋というものである。

しかし，それが顧客から依頼された仕事となると，話は別である。

自らの技術の多少上のレベルの案件を受注して努力することはスキルアップのためにも必要である。

だが，努力しても追い付けないほどの仕事を請けるのは「無謀」ではないだろうか。それが社会基盤に関わるものであるならば，一般市民の生活に甚大な被害を与える恐れすらある。正確な情報を得られなかったために，ユーザの人生が変わってしまうことだってあるのだ。

気軽な設計に基づく文書が膨大に蓄積された暁には，再利用は困難を極める。設計者ひとりひとりが責任を自覚し，挑戦と無謀を見極めなければならない（もっとも，逆に筆者は，周りからは慎重すぎると言われているのだが）。

設計者は，短期報酬型よりも長期報酬型，正常性バイアスが働き過ぎるぐらいで，ちょうどいいのではないかと思う。

XML の関わる問題は，現段階ではまだ，データ爆発によるエネルギー消費や，情報の伝達速度といった点にとどまっている。だからといって，問題が軽微なわけではない。XML は，コンピュータによる計算だけではなく，ヒトによる理解をも目標とする，きわめて人間寄りの仕様である。第 4 章の「哲学的問題～最小単位を定義する～」で述べたことが現実味を帯びてきたときには，XML 設計の良し悪しが，ヒトの精神に深刻な影響を及ぼしたり，多数の人々の生死を左右する事態も懸念される。

この社会で役立つ設計をするには，コンピュータに向かい，終日データや開発ツールの画面と対峙しているだけでは不完全なのではないだろうか。

自分以外の誰かが考えた言葉で語りうる知識を吸収する以上に，自らの頭ひとつで，人生について，この世界について悩み考え抜き，行学一致の生活を心がけることでしか，気付きえないことがあるのではないだろうか。

筆者紹介

PROJECT KySS （プロジェクト･キッス）
薬師寺国安と，聖による，コラボレーション･ユニット。1997 年，ActiveX と Dynamic HTML の普及を目的として結成。各々が技術専門誌に連載記事を執筆後，PROJECT KySS 名義で，XSL ワーキングドラフト，XML マスターベーシック資格試験，Microsoft InfoPath，Expression Design など，日本語の初発本や記事を多数執筆。また，日経 IT Pro にコラボレーションや Web プランニングなどの人気コラムを執筆。現在は，Silverlight の記事を連載中。
両名とも，Microsoft MVP for Data Platform Development (Oct 2003-Sep 2010)
実務では，科学技術･医療･福祉分野の情報を扱う案件を手掛けている。
http://www.PROJECTKySS.NET/

2009 年 10 月 10 日 初版発行
著者 PROJECT KySS
編集･装丁 薬師寺聖
出版元 セイザインデザイン 代表 薬師寺聖
http://www.SeinDesign.net/