特集3

Open Office XMLはこう使おう

MS Office＋XML-DBによる文書データ統合管理と検索システムの構築

WINGSプロジェクト
佐藤 治夫（著）
SATO, Haruo
山田 祥寛（監修）
YAMADA, Yoshihiro

Microsoft Officeの最新バージョンである「Microsoft 2007 Office System」がリリースされて、2年が経過しようとしている。バージョンアップによる大きな変化の1つとして、「Office Open XML」というXMLをベースとした文書の標準フォーマットを本格的に採用した点が挙げられる。これにより、散在しているOffice文書をXMLデータとして統合的に格納／管理できるようになった。本稿では、Office Open XMLとXML-DBによるシステム構築について解説する。

Open Office XMLについて

「Microsoft 2007 Office System」（以下、Office 2007）に搭載された新機能として「Office Open XML」がある。

Office Open XMLは、Word/Excel/PowerPointなどのOffice文書を電子ファイルとして保存するためのXML、およびZIP技術をベースにしたファイルフォーマットの標準仕様である。仕様の策定はマイクロソフトを中心に進められ、2006年12月にECMAでECMA-376として、2008年4月にはISOでISO/IEC 29500として標準化された。また、2008年12月にはECMAでいくつかの修正が施されたSecond Editionが公開されている。

Open Office XMLの構成

Office Open XMLは、主に以下のMarkup Language（以下、ML）から構成される。

- Word文書に対応したWordProcessing ML
- Excelファイルに対応したSpreadSheet ML
- PowerPoint文書に対応したPresentation ML
- 図形、画像、テキストへのグラフィック効果などの描画機能に対応したDrawing ML

MLとは、文書構造を表現するための言語のことである。Office Open XMLには、そのほかにも数式を表現するOffice Math、ベクター画像のためのVML（Vector Markup Language）といったMLがある。

パッケージとパーツ

Office 2007で作成されたファイルは一見すると1つのファイルに見えるが、実は複数のファイル群がZIP形式で圧縮されパッケージングされている。Office Open XMLではこのファイル群を格納するためのコンテナを「パッケージ」、パッケージ内に格納されているファイルを「パーツ」と呼んでいる。**図1**に、Wordファイルのパッケージ内の構造を示す。

パーツは、必要最低限のものを分類すると**表1**のようになる。表に示した以外にも、オプションで画像用のフォルダ（mediaフォルダ）などがある。これは、文書内に画像が存在する場合に作成される。

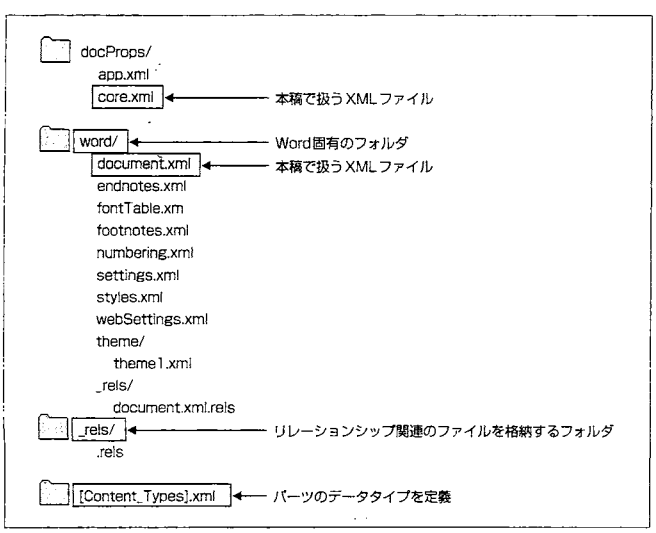


図1：Wordprocessing MLのパッケージ構造

表1：パーツの分類

パーツ	フォルダ名（ファイル名）	説明	Word/Excel/PowerPointで共通
コンテンツタイプ	[Content_Types].xml	パーツのデータタイプを定義する	○
リレーションシップ	_rels	パーツ間の関連を定義する	○
ドキュメントプロパティ	docProps	アプリケーションやドキュメントのプロパティを定義する	○
ドキュメントパーツ	word	Wordの場合「word」、Excelの場合「xl」、PowerPointの場合「ppt」というフォルダ名になる	×

表2：主要なネイティブXML-DB製品

製品名	発売元
Cyber Luxeon	サイバーテック
NeoCoreXMS	
EsTerra XML Storage Server	メディアフュージョン
TXT	東芝ソリューション

表3：XML-DBが活用されている分野

XML-DBが活用されている分野	例
ドキュメントデータの管理	業務マニュアル、約款契約書、通販カタログ、情報誌、ISO文書、論文、医薬品添付文書
メタデータの管理	製品情報管理、部品表、Webカタログ、電子カルテ、CMS
業界標準データの管理	XBRL、ロゼッタネット、BMS、NewsML、RSS、MML

Office文書にOffice Open XMLが採用されたメリット

Office文書がマイクロソフト独自のバイナリ形式から、Open Office XMLというXMLベースのオープンなフォーマットで保存されるようになったメリットを以下に列挙する。

透明性／アクセス容易性

バイナリ形式が標準であったOffice 2003までは、Office文書のデータに外部プログラムからアクセスするにはバイナリ形式の解読から始めるか、もしくはJakarta POI（Java）などのライブラリを使用する必要があった。

Office 2007ではXML形式のためデータ構造がすべて明らかになっており、自由にアクセスして利用できる。また、XML関連仕様のDOMやSAXなど標準のデータ操作方法で、Office文書のデータにアクセスするプログラムを開発することも可能だ。

文書をデータとして利用しやすくなった

XMLによって、文書がタグ付けされ半構造化されているため、データとしての2次利用が可能になった。Office文書のアウトライン機能で階層化された文書データもXMLによってタグ付けされるので、見出しレベルによる検索機能なども容易に開発できる。

パーツ化によるメリット

データが複数のXMLと画像ファイルに分けられてパッケージングされるので、ファイル破損のリスクが分散された。また、フッターやヘッダーなどパーツごとの再利用が容易となり、プログラムにより一斉にフッターやヘッダーを差し替えられるようになっている。

データサイズの縮小

ZIP形式で圧縮されているので、ファイルサイズが縮小された。

マイクロソフト以外のオフィス製品との相互運用性

XMLのフォーマットが公開されているので、Office Open XMLをサポートしているオフィス製品であれば、MS Officeで作成された文書の読み書きができるようになる。ただし、執筆時点（2008年12月）ではOffice Open XMLをサポートしている製品はMS Officeのみである。

本稿で使用するXML-DB

ここで、今回のサンプルプログラムで利用するXML-DBについて簡単に紹介しよう。

まずは、主要なネイティブXML-DB製品を**表2**に示す。このうち、本稿ではNeoCoreXMSの評価版を採用する。

NeoCoreXMSの大きな特徴は、DPP（Digital Pattern Processing）という独自インデックス方式による高速で大容量データにも対応した検索機能である。NeoCoreXMSは、データ保管時にXMLの全要素に対して自動でインデックス付けを行なう「フルインデックス」方式を採用しているので、XMLデータの設計者はXMLに対するインデックスを意識する必要がない。DPPが設定するインデックスは、「アイコン」と呼ばれる64ビットのデータで表現される。そのサイズは常に一定（64ビット）なので、検索時はデータ量に依存せずに一定で高速な検索スピードを実現している。

XML-DBの活用事例

XML-DBの活用は、項目が多くかつデータ構造が階層構造で複雑なデータを扱う分野で着実に広がっている。例を挙げると、**表3**に示す分野である。

表3で挙げた分野のデータを表形式のRDBで管理しようとすると、データ構造が複雑になり、テーブルの結合処理などでパフォーマンスが劣

Column

OpenOfficeとMS Officeの相互運用

MS Officeに対抗する代表的なオフィススイート製品として、「OpenOffice」がある。OpenOfficeは、オープンソースプロジェクトとしてOpenOffice.org（http://www.openoffice.org/）で開発が進められており、ファイルフォーマットとしてOpenDocument Format（以下、ODF）というXMLをベースとした標準仕様が採用されている。

ODFは、構造化情報標準促進協会（OASIS）、および国際標準化機構（ISO）と国際電気標準会議（IEC）の合同技術委員会ISO/IEC JTC 1/SC 34によって、Office Open XMLよりも一足早く、2006年5月にISO/IEC 26300として標準規格に認定された。

マイクロソフトは、ODFと同じ目的のOpenXMLという仕様を策定する一方、2008年にはOffice製品群で幅広く文書フォーマットをサポートする方針を打ち出し、Office 2007のService Pack 2ではODF version1.1などに対応することを発表している。

また、マイクロソフトはMS OfficeでODFファイルを読み書きできるアドイン「OpenXML/ODF Translator Add-in for Office」を開発するオープンソースプロジェクトも支援していることなどから、今後はオフィススイート間の相互運用／協調路線の方向に進んでいくものと思われる。

化するという問題が発生しがちである。ここでは、XML-DBを活用した事例としてECサイトの商品カテゴリを編集／表示するシステムを説明する。**図2**にシステムの構成を示す。

図2では、ECサイトの店舗担当者が商品カテゴリを画面から登録する。商品カテゴリは階層構造を成すデータなのでそのままXMLで表現し、アプリケーションはデータを加工することなく、そのままXML-DBに保存できる。その商品カテゴリ情報を一般消費者が参照する際も、XML-DBからXMLを抽出し、スタイルシートによってHTMLなどに変換して表示することが可能だ。

ただし、商品カテゴリ情報をRDBに格納しようとした場合、テーブルの構造が問題となる。**図3**にテーブルの設計案を示す。

案1の方法は、データの構造上、無制限にデータを階層構造として表現できる。しかし、データ量が多くなるとテーブルのJOINによるパフォーマンスの悪化が懸念される。案2の方法は、階層構造を増やそうとするとカラムの追加とアプリケーションの変更が必要となる。

いずれの設計方法にせよ、階層が深くなったりデータ量が多くなったりすると極端なパフォーマンスの悪化が予想され、階層のレベル、カテゴリ数に上限を設定せざるを得なくなる。そうなるとビジネスサイドとの調整が必要になり、プロジェクト期間が延びてしまう恐れも出てくる。また、データを抽出するためのSQLも、データを操作するアプリケーションも複雑になるので、開発コストも増えることになるだろう。

上記のジャンルに限らず、今後は特にWebシステムに求められる機能も、データ構造も複雑化していくことが予想される。このことから、XML-DBが活躍する場面はさらに増えていくだろう。

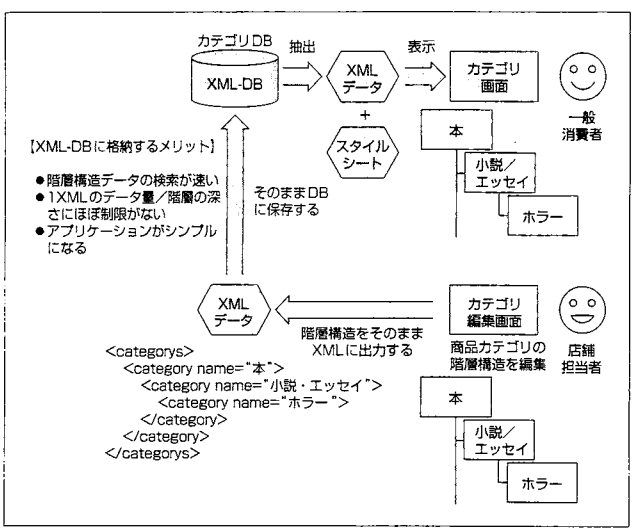


図2：XML-DBの活用事例

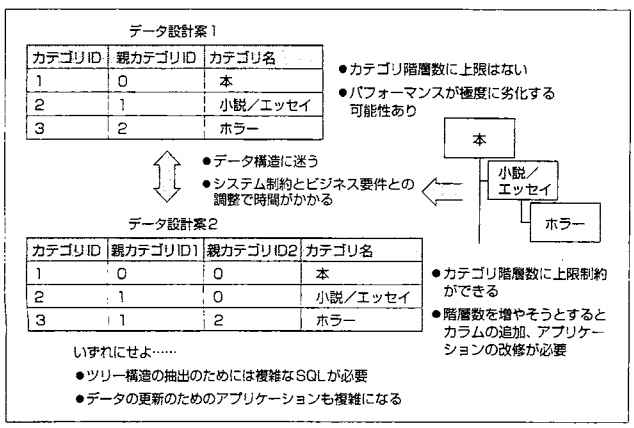


図3：階層構造のデータをRDBに保存

サンプルアプリケーションの構築

Office Open XMLとXML-DBについての説明が終わったところで、実際にOffice Open XMLを活用したアプリケーションを構築していく。

本稿で構築するのは、WordファイルのXMLをXML-DBに格納し、格納したXMLデータをWebブラウザから検索するというアプリケーションである。構築するアプリケーションの全体構成を**図4**に示す。

サンプルアプリケーションの全体構成

図4に示した処理の構成は、大きく分けて次の2つである。

①XML-DBにデータを格納するためのバッチ処理
②格納されたXMLデータを検索するWebアプリケーション

①XML-DBにデータを格納するためのバッチ処理

ファイルサーバー上のあるフォルダをルートとして、その配下に存在するWord文書を探索して見つけた場合、一時フォルダにコピーしてZIP解凍する。ZIP解凍されたファイルの中から、document.xmlとcore.xmlのデータをXML-DBに保存／修正／削除する処理を実行する。

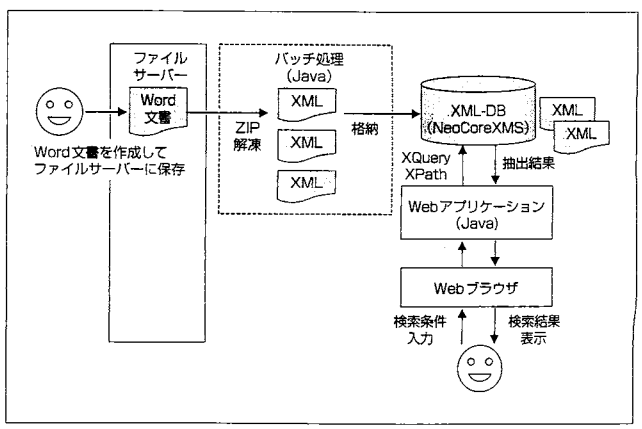
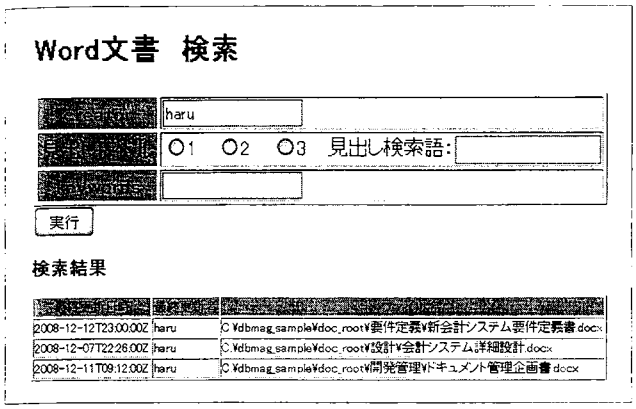


図4：サンプルアプリケーションの全体構成



画面1：検索条件入力画面と検索結果表示画面

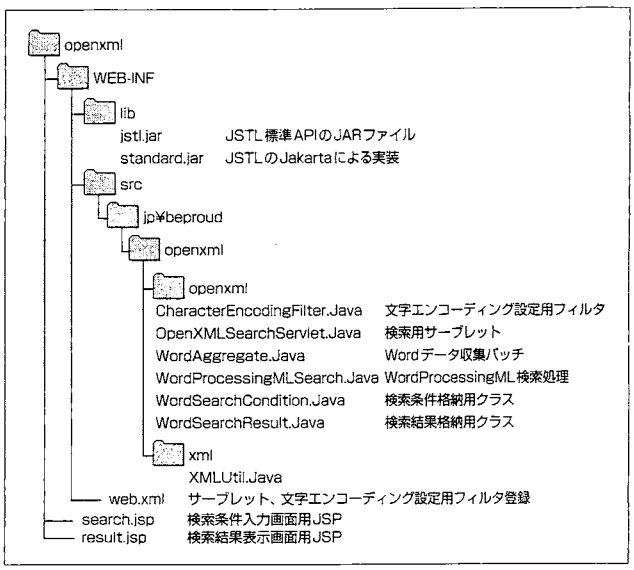


図5：サンプルアプリケーションのファイル構成

なお、document.xmlはWord文書の内容を表わすXMLファイル、core.xmlはWord文書のプロパティ（作成者／日時、最終更新者／日時、説明書きなど）が出力されるXMLファイルである。

②格納されたXMLデータを検索するWebアプリケーション

バッチ処理で格納したXMLデータをWeb画面から検索し、一覧表示する。画面イメージを**画面1**に示す。

XML-DBとOpen Office XMLを活用したその他の事例

本稿のサンプルアプリケーションは、Wordファイルの XMLをXML-DBに格納する事例である。しかし、XML-DBに格納したWord文書のXMLを元データとして、Excelに帳票として出力することもできる。その際には、SpreadSheet MLで定められたXML仕様／パッケージ構造に従った形式でXMLファイルを生成し、ZIP圧縮でパッケージングしてExcelファイルを作成する。

そのほか、Excelに出力するだけでなくスタイルシートを適用してHTMLやPDFに出力するなど、ワンソースマルチユースでのデータ利用も考えられるだろう。

環境構築

今回のサンプルアプリケーションでは、次のソフト／ミドルウェアを使用する。

- Windows XP Professional SP3
- MS Office 2007 SP1MSO
- NeoCoreXMS ver 3.1.3
- Java 6 Update 11
- Tomcat 6.0.18
- Eclipse 3.4

Java、Tomcat、Eclipseの入手／インストール方法については、誌面の都合により本稿では割愛する。詳細は、WINGSプロジェクトのサイト「サーバサイド技術の学び舎 - WINGS」（http://www.wings.msn.to/）から「サーバサイド環境構築設定」を参照してほしい。

サンプルアプリケーションの入手とファイル構成

サンプルアプリケーション用のWARファイルが本誌付録CD-ROMに収録されている。このWARファイルを「%CATLINA_HOME%/webapps」にコピーして利用する。Tomcatを起動すると、**図5**に示すフォルダ構成でファイルが展開される。

NeoCoreXMS評価版のインストール

NeoCoreXMSの評価版は、サイバーテック社のサイトからダウンロードできる。

https://www.cybertech.co.jp/

評価版は、NeoCoreXMSのすべての機能を30日間使用できる。サイバーテック社のサイトのフォームから評価ライセンスを申し込むと、ダウンロードURL付きのメールが送られてくる。ファイルをダウンロードし、インストーラに従って進めていくとインストールが完了する。NeoCoreXMSの評価版をダウンロードすると、サイバーテック社からライセンスファイルがメールで送信されてくるので、NeoCoreXMSをインストールしたフォルダ配下の「neoxml ¥config」フォルダに「license.xml」という名前でライセンスファイルを保存する。

NeoCoreXMSは、スタートメニューから［すべてのプログラム］-［NeoCoreXMS］-［Start Neo Server］を選択することで起動できる。

また、NeoCoreXMSによる開発を行なうために「NeoCoreXMSのインストールフォルダ/API/Java/lib」フォルダにあるxmsclient.jarを、アプリケーションルート配下の「WEB-INF/lib」フォルダにコピーしよう。その後、Eclipseの参照ライブラリに追加することで、プログラムからAPIを使用できるようになる。

JSTLのインストール

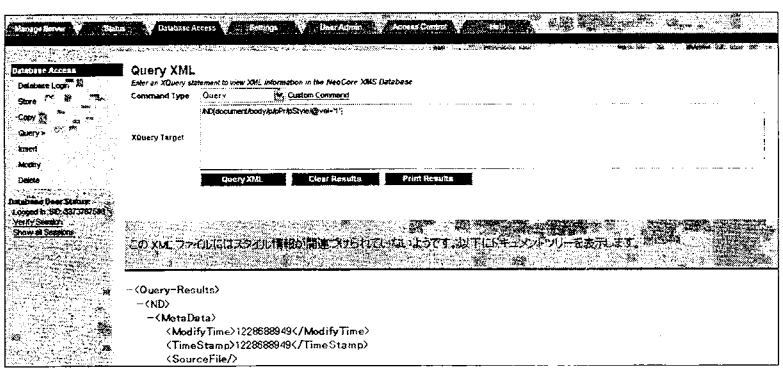
JSPで検索結果の表示を制御するために、JSTLという標準タグライブラリを使用する。JSTLをインストールするには、Apache JakartaのTaglibのサイトにアクセスする。

http://jakarta.apache.org/taglibs/index.html

このサイトから「jstl.jarとstandard.jar」を入手し、アプリケーションルート配下の「WEB-INF/lib」フォルダにコピーする。「% TOMCAT_HOME% /webapps/examples/WEB-INF/lib」フォルダからコピーすることも可能だ。

Column
NeoCoreXMS管理コンソール

NeoCoreXMSには、NeoCore管理コンソールというブラウザベースのインターフェイスが用意されている(画面A)。Windowsの場合、スタートメニューから[すべてのプログラム]－[NeoCoreXMS]－[XMS管理コンソール]を選択することにより起動できる。また、ブラウザで直接管理コンソールのURL(http://localhost:7001/console)を指定しても起動可能だ。



画面A：NeoCoreXMS管理コンソールの画面イメージ

NeoCoreXMSの基本

具体的なソースコードの説明に入る前に、NeoCoreXMS上のデータのアクセス方法について説明する。**図6**に、NeoCoreXMSに各種クライアントがアクセスする方法について示す。

HTTPインターフェイスとAPI

NeoCoreXMSに用意されているHTTPインターフェイスには、GET、POSTによるシンプルなインターフェイスがある。クライアントはこれらのインターフェイスを使用して、NeoCoreXMS上のデ

表4：NeoCoreXMS JavaAPIの基本用例

戻り値型	戻り値説明	メソッド名	引数	メソッド説明
String	ログイン結果。ログインに成功した場合セッションID	login	String loginid, String password	NeoCoreXMSにログインする
String	検索結果XML	queryXML	String query	XQueryまたはXPathで検索する
String	挿入結果XML	insertXML	String query String xmlString	queryで指定した箇所に第2引数のXMLを挿入する
String	修正結果XML	modifyXML	String query String xmlString	queryで指定した箇所のXMLを第2引数のXMLで修正する
String	削除結果XML	deleteXML	String query	queryで指定した箇所のXMLを削除する
void	なし	startTransaction	なし	トランザクションを開始する
void	なし	commitTransaction	なし	トランザクションをコミットする
void	なし	rollbackTransaction	なし	トランザクションをロールバックする

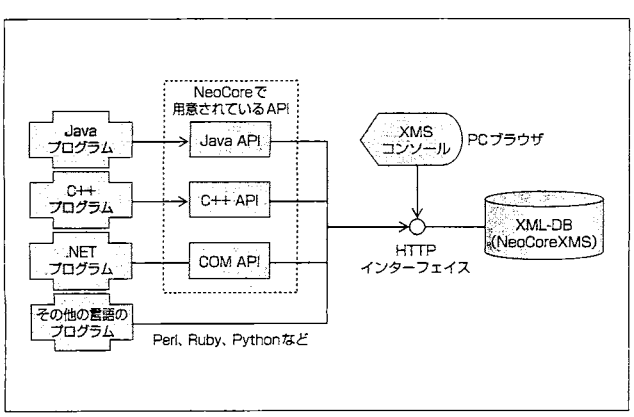


図6：NeoCore APIへのアクセス方法

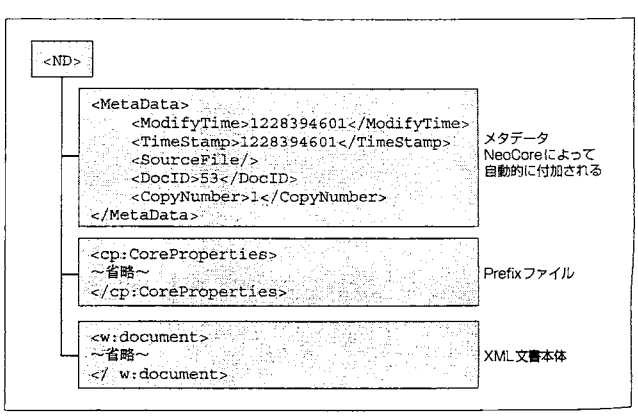
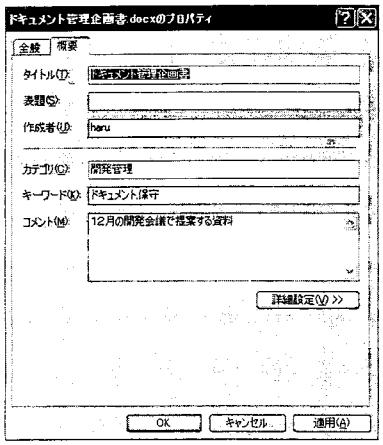


図7：NeoCoreXMSに格納されるXMLデータの形式



画面2：Word文書へのプロパティ設定

ータにアクセスする。クライアントプログラムがJava、C++、.NET（COM）の場合は、より使いやすいHTTPインターフェイスをラップしたAPIが用意されているので、そちらの使用をお勧めする。

本稿では、Java APIのSessionManagedConnectionに用意しているメソッドを使用している。**表4**に、本稿で使用するNeoCoreXMSのJava APIを示す。

NeoCoreXMSに保存されるXMLの形式

次に、NeoCoreXMS上に保存されるXMLデータの形式について説明する。

NeoCoreXMSにXMLデータが格納（Store）されると、必ず<ND>という要素が作成される。格納されるXMLデータは<ND>要素の子要素として格納される。**図7**に、NeoCoreXMSに格納されたXML文書の形式を示す。

データベースに格納されるXMLデータの構造

続いて、サンプルアプリケーションでXML-DBに格納されるdocument.xmlと、core.xmlの構造について説明する。

document.xmlの構造

ここでは、「こんにちは」という簡単な文章が書かれたWordファイルが出力したdocument.xmlを図8に示す。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>                    ルート要素
<w:document                                                                名前空間宣言
    xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml">
  <w:body>                                                                 文書本体
    <w:p w:rsidR="00BF2D49" w:rsidRDefault="00F96A5B">                     段落
      <w:pPr>
        <w:rPr>
          <w:rFonts w:hint="eastAsia"/>
        </w:rPr>                                                          段落プロパティ
      </w:pPr>
      <w:r>                                                              ラン
        <w:rPr>
          <w:rFonts w:hint="eastAsia"/>
        </w:rPr>                                                          ランプロパティ
        <w:t>こんにちは</w:t>                                              テキストデータ
      </w:r>
    </w:p>
    <w:sectPr w:rsidR="00BF2D49" w:rsidRDefault="00BF2D49">                 セクション
      <w:pgSz w:w="11906" w:h="16838"/>
      <w:pgMar w:top="1985" w:right="1701" w:bottom="1701" w:left="1701" w:header="851"
        w:footer="992" w:gutter="0"/>
      <w:cols w:space="425"/>
      <w:docGrid w:type="lines" w:linePitch="360"/>
    </w:sectPr>
  </w:body>
</w:document>
```

図8：document.xmlの構造

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties
    xmlns:cp="http://schemas.openxmlformats.org/
              package/2006/metadata/core-properties"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:dcmitype="http://purl.org/dc/dcmitype/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <dc:title>ドキュメント管理企画書</dc:title>
    <dc:subject></dc:subject>
    <dc:creator>haru</dc:creator>
    <cp:keywords>ドキュメント,保守</cp:keywords>
    <dc:description>12月の開発会議で提案する資料</dc:description>
    <cp:lastModifiedBy>haru</cp:lastModifiedBy>
    <cp:revision>2</cp:revision>
    <dcterms:created xsi:type="dcterms:W3CDTF">2008-12-07T17:04:00Z</dcterms:created>
    <dcterms:modified xsi:type="dcterms:W3CDTF">2008-12-07T17:04:00Z</dcterms:modified>
    <cp:category>カテゴリは設計文書です</cp:category>
</cp:coreProperties>
```

LIST1：Wordファイルに含まれるcore.xml

document.xmlのルート要素はw:document要素で、その子要素にはw:body要素が1つだけ付加される。そして、w:bodyの子要素にはw:p要素（段落）やw:sectPr（セクション要素）が付加される。

w:p要素（段落）は、1つのw:pPr要素（段落プロパティ）と複数のw:r要素（ラン）で構成される。w:pPr要素には、文字の行端揃えなど段落のプロパティ設定が保存される。ランには、テキストデータとそのプロパティ情報（フォントサイズ、フォントなど）が保存される。w:bodyの子要素の構造は、Word文書の文章構造によってさまざまに変化する。

core.xmlのXML構造

画面2のようにプロパティをWordファイルに設定した場合、**LIST1**のcore.xmlとして保存される。

サンプルアプリの構築

以上の前提を理解したところで、以降では実際のサンプルアプリケーションの構築について説明していこう。

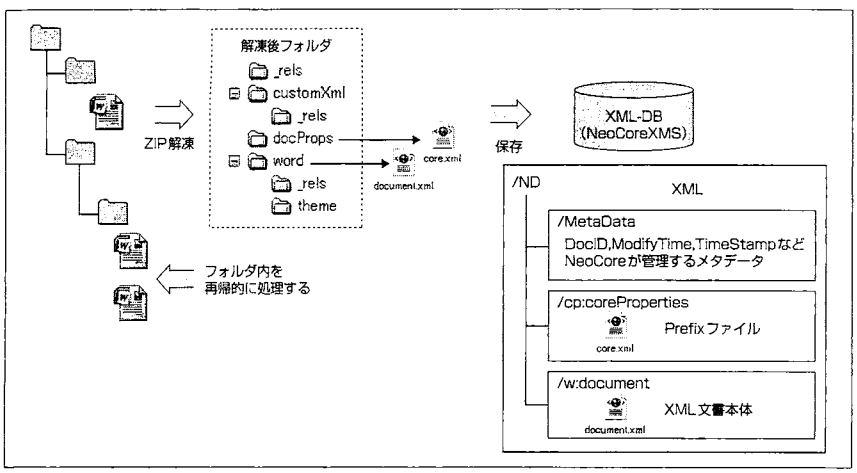


図9：Wordデータ収集／格納バッチ処理概要

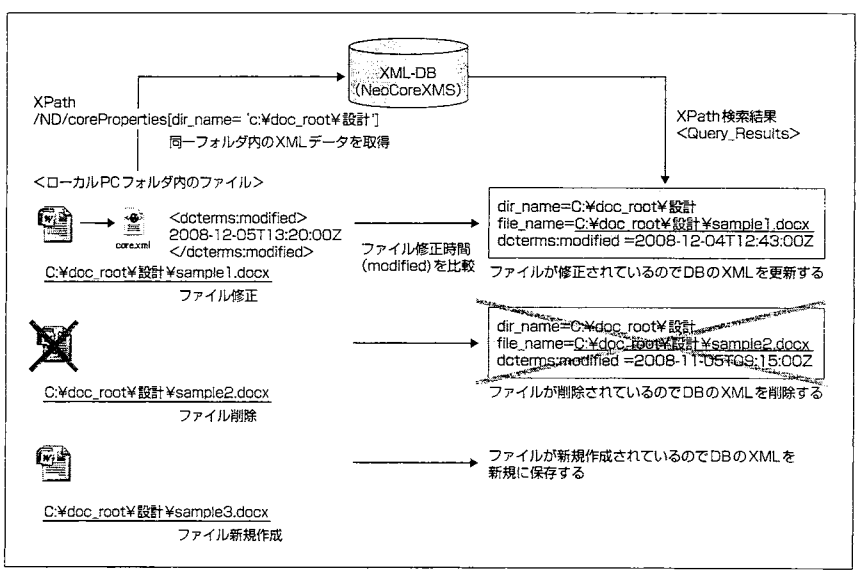
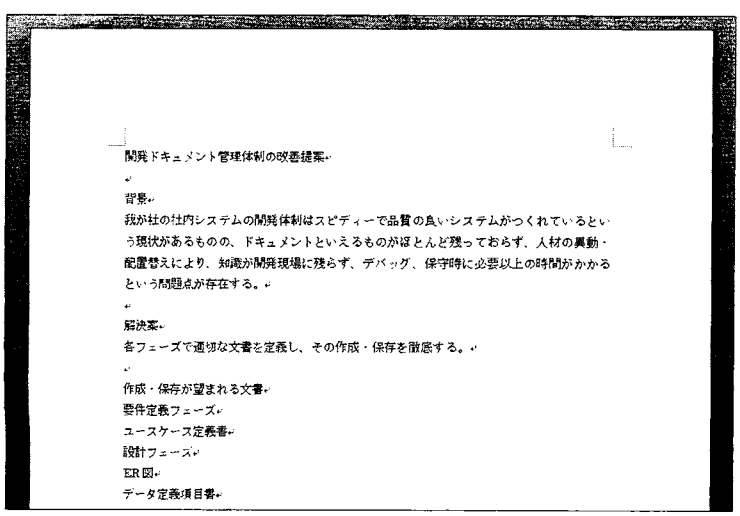
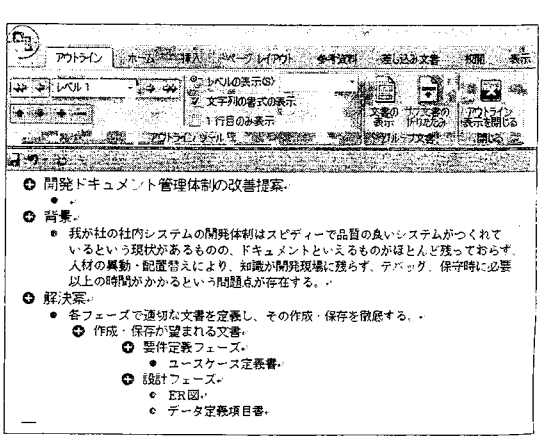


図10：Wordファイル収集バッチファイル制御処理の仕様詳細



画面3：Wordファイルへの文章の入力



画面4：Wordのアウトライン表示

Word文書の作成

まずは、Word文書を作成する。通常のように、文章を次々と入力していけば良い。その際に、見出しのスタイル設定、強調表示などは意識せず、テキストファイルに入力していく感覚で文章を打ち込んでいく（**画面3**）。

ひととおり文章を打ち終えたところで、アウトライン表示に切り替えて見出しにスタイルを設定し、文書を階層化させていく（**画面4**）。

必要に応じて、エクスプローラからWordファイルを右クリックし、プロパティの設定を行なう（画面2）。

Wordデータ収集／格納バッチ処理の実装

Word文書を作成したところで、いよいよファイルサーバー上のWord文書を探索し、XML-DBに格納するバッチ処理を実装する。

バッチ処理の処理仕様を**図9**に示す。Wordデータ収集／格納バッチ処理は、ファイルサーバー内にあらかじめ指定したフォルダを起点として、ツリー構造から再帰的にWordファイルを探索する。Wordファイルを見つけた場合、そのファイルをZIP解凍してパッケージの中からdocument.xmlとcore.xmlを取り出し、1つのXML文書としてXML-DB上に保存する。

バッチ処理により、XML-DBに対して文書を新

規作成／修正／削除する際の仕様を図10に示す。探索したWord文書のデータが存在しない場合、バッチ処理は新規にデータをXML-DBに保存する。

また、ローカルPCで既存のWord文書が修正された場合、XML-DB上のデータも修正される。修正の有無は、ローカルファイルとXML-DBのcore.xmlのcp:coreProperties/dcterm:modified要素値を比較することにより判定する。そして、ローカルPCにすでにWordファイルが存在しない場合は、XML-DB上のデータを削除する。

バッチプログラムの実装

バッチ処理の全体像をつかんだところで、バッチ処理のJavaプログラムの説明に入る。バッチ処理は、WordAggregate.javaのmainメソッドから実行する(LIST2)。

クラス定数には、ファイル探索の起点となるルートのフォルダを定義したDOC_ROOT変数と、Wordファイルを解凍するにあたり一時的にファイルを移動するためのTMP_DIR_NAME変数が定義されている。mainメソッドからは、実処理が書かれているprocessDirを呼び出しているだけである。

フォルダ内のWordファイルを探索し、XML-DBに格納するロジックが書かれているのがprocessDirメソッドである。このメソッドがXML-DBに格納するデータの新規作成／修正／削除を制御するメインのロジックとなる(LIST3)。

なお、本稿のサンプルプログラムでは、XMLの処理に共通で使用する処理をXMLUtilというクラスに実装している。表5に、XMLUtilクラスに実装したメソッドを示す。実際のソースコードは、本誌付録CD-ROMに収録しているサンプルアプリケーションを参照してほしい。

processDirメソッドの処理フロー

LIST3のprocessDirメソッドでは、図11のと

```
public class WordAggregate {

    // ドキュメントを格納するルートのフォルダ
    private static final String DOC_ROOT = "C:/dbmag_sample/doc_root";

    // Wordファイルをコピーして解凍するための一時フォルダ
    private static final String TMP_DIR_NAME = "C:/dbmag_sample";

    // バッチ処理起動用mainメソッド
    public static void main(String[] args) throws Exception{
        // ドキュメントのルートから階層構造をたどり、ファイル名を取得する
        File docRootDir = new File( DOC_ROOT );
        processDir( docRootDir );
    }
```

LIST2：ファイル収集バッチ処理の起動(WordAggregate.java)

```
private static void processDir(File dir) throws Exception{
    // ステップ1：フォルダ内のファイル集合を取得する
    File[] subFiles = dir.listFiles();

    // ステップ2：XML-DBから、該当フォルダの文書一覧をMapで取得する（core.xmlの内容）
    HashMap<String, String> docMap = getCorePropertiesDataMap(dir);

    // ステップ3：ファイル集合すべてに対して繰り返す
    for (int i = 0; i < subFiles.length; i++) {
        File subFile = subFiles[i];

        // ディレクトリの場合、再帰実行する
        if ( subFile.isDirectory() ) {
            processDir( subFile );
            continue;
        }

        // 以下のファイルは、対象外なので読み飛ばす
        // ・ファイル名の末尾が「.docx」で終わらないファイル
        // ・「~$」が先頭についたファイル
        if (!subFile.getName().endsWith( ".docx" ) || subFile.getName().startsWith("~$") ){
            continue;
        }

        // Wordファイルの場合、XMLDBへの保存処理を実行する
        // ステップ3-1：一時フォルダにファイルをコピーする
        String dstFileName = TMP_DIR_NAME + File.separator + subFile.getName();
        copyTransfer( subFile.getAbsolutePath() , dstFileName );

        // ステップ3-2：コピーしたファイルをzip解凍する
        unZip(dstFileName);

        // ローカルファイル(core.xml)とサーバに保存されたXMLのタイムスタンプを比較する
        String valueModified = docMap.get(subFile.getAbsolutePath());

        if ( valueModified == null ){
            // サーバーにデータが存在しない場合、新規に保存(Store)する
            // ステップ3-3-1：XML-DBにXMLを保存する
            String bodyFileName = TMP_DIR_NAME + File.separator + "word/document.xml";
            String prefixFileName = TMP_DIR_NAME + File.separator + "docProps/core.xml";
            String docId = storeOpenXML( bodyFileName, prefixFileName, null, subFile );

        }else{
            // 解凍されたcore.xmlから、modifiedを取得する
            Document coreXMLDoc =
                XMLUtil.createXMLDocument(TMP_DIR_NAME + "/docProps/core.xml");
            Element elemLocalModified =
                (Element)coreXMLDoc.getDocumentElement().getElementsByTagName(
                    "dcterms:modified").item(0);
            String localModified = elemLocalModified.getTextContent();

            // ステップ3-3-2：modified timeを比較して差異があれば、XMLを更新する
            if ( !valueModified.equals(localModified) ){
                Element wordBodyDocElem
                    = XMLUtil.createXMLDocument(
                        TMP_DIR_NAME + "/word/document.xml").getDocumentElement();
                modifyOpenXML(
                    subFile.getAbsolutePath() , XMLUtil.node2String( wordBodyDocElem ) ,
                    XMLUtil.node2String( coreXMLDoc.getDocumentElement() ) );
            }
            // ステップ4：処理したXMLファイルはMapから削除する
            docMap.remove(subFile.getAbsolutePath());
        }
    }

    // XML-DBに存在するが、ファイルサーバーに存在しないデータを削除する
    // (Mapに残っているファイルは、ローカルサーバーに存在しないファイル)
    Set<String> docFileNameSet = docMap.keySet();
    for (Iterator iterator = docFileNameSet.iterator(); iterator.hasNext();) {
        String delFileNm = (String) iterator.next();
        NeoCoreClient.deleteXML("/ND[coreProperties/file_name='" + delFileNm + "']");
    }
}
```

LIST3：ファイル探索とXML-DBへの格納処理(WordAggregate.java)

おりに処理フローが実行される。

以降、Wordデータ収集／格納バッチ処理が記述されたprocessDirメソッドの中で、ポイントとなる箇所を説明していく。

ステップ2 既存XML文書の修正時間情報取得

XML-DBにアクセスし、XML-DBに格納した既存のXML文書の修正時間情報を取得する処理は、getCorePropertiesDataMapメソッドに実装されている。取得した情報はMapで返却される（LIST4）。

XPathとXQuery

XML-DB上のXMLデータ検索にはXPathまたはXQueryを使用する。XPathとXQueryはW3Cで標準化されているXML検索用の仕様である。

XPathは、XML文書の中の特定の要素を指し示す記述方法を定めている。XQueryはRDBでいうところのSQLに該当し、さまざまなタイプのXMLデータソース（XMLファイル、ネイティヴXML-DB、RDBなど）が混在するマルチベンダ、かつマルチデータベース環境におけるデータの検索／抽出／結合の実現を目的として設計されたクエリ言語である。

XQueryは、XPathのサブセットとして使用できる。

表5：XML共通処理

戻り値の型	メソッド名	引数	説明
String	xmlString	String fileName	引数で渡されたファイル名のXMLデータを文字列で返す
Document	createXMLDocument	String fileName	引数で渡されたファイル名のXMLからDocumentオブジェクトを生成して返す
String	node2String	Node node	Nodeオブジェクトを文字列にして返す
Document	string2Document	String xmlStr	XML文字列からDocumentオブジェクトを生成して返す
Element	getChildElement	Element elem，String tagName	第1引数の子要素から第2引数の名前の要素を返す

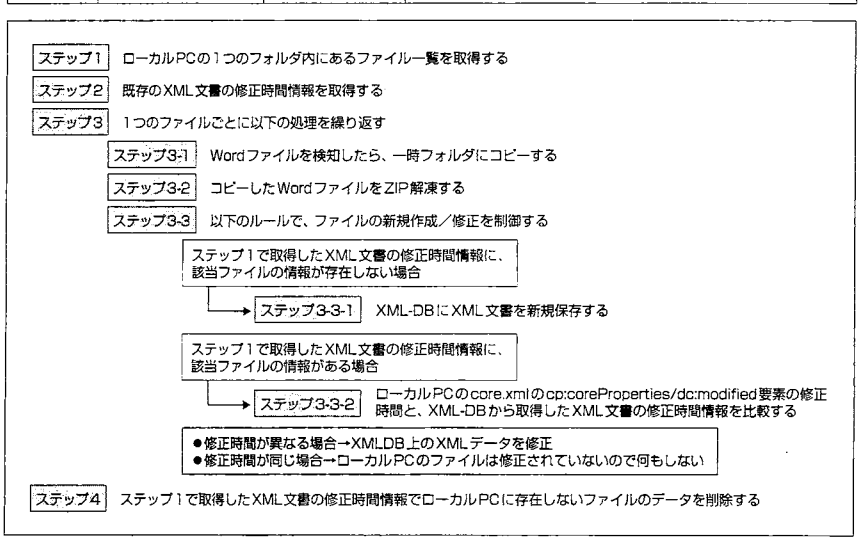


図11：prosessDirメソッドの処理フロー

```
private static HashMap<String, String> getCorePropertiesDataMap(File dir)
    throws Exception {
    Document coreProps =
      = NeoCoreClient.queryXML(
        "/ND/coreProperties[dir_name='" + dir.getAbsolutePath() + "']" );
    Element queryResult = coreProps.getDocumentElement();

    NodeList corePropsList = queryResult.getElementsByTagName("cp:coreProperties");
    // 検索結果のXMLデータをHashMapに移す（キー：ファイル名、値：修正時間）
    HashMap<String, String> docMap = new HashMap<String, String>();
    int corePropsListLength = corePropsList.getLength();

    for (int i = 0; i < corePropsListLength; i++) {
        Element elemCoreProperties = (Element)corePropsList.item(i);
        Element elemFileName
          = XMLUtil.getChildElement(elemCoreProperties, "file_name");
        Element elemModified
          = XMLUtil.getChildElement(elemCoreProperties, "dcterms:modified");
        // キー：更新時間、値：modified
        docMap.put(
          elemFileName.getTextContent(), elemModified.getTextContent());
    }
    return docMap;
}
```

LIST4：XML-DBから既存XML文書の修正時間を取得（WordAggregate.java）

XPathによる検索で取得するXML

XPathによる検索で取得した結果のXMLをLIST5に示す。この検索結果を取得するXPathの構文は、次のとおりである。

```
/ND/coreProperties[dir_name='フォルダ名']
```

このXPathでは、述語として/ND/coreProperties要素の子要素のdir_nameを指定している。XPathの述語とは、ノードテストで選出されたノード集合をさらに振り分ける条件を記述する部分を指す。LIST5では、ノードテストで選出されたノード集合（/ND/coreProperties）をさらにdir_name（フォルダ名）で絞り込み、同一フォルダ内の要素集合を抽出している。

NeoCoreXMSで検索した結果は、必ず<Query-Results>というルート要素の下に格納される。<cp:coreProperties>要素は、Wordファイルに格納されているcore.xmlのルート要素である。

LIST5では「/ND/coreProperties[dir_name='フォルダ名']」というXQueryで、同一フォルダ内のcp:coreProperties要素の集合を取得し、その要素の子要素である<dcterms:modified>要素と<file_name>要素をMapに格納している。ただし、<file_name>要素と<dir_name>要素は本稿のサンプルアプリケーションが独自に加えた要素である。これらの要素については後述する。

NeoCoreXMSへのコネクション取得

次に、NeoCoreXMSへのコネクションを取得する処理を**LIST6**に示す。NeoCoreXMSへのコネクションは、NeoCoreXMSがAPIとして用意しているSessionManagedNeoConnectionを使用する。コンストラクタの引数に接続先ホスト名とポート番号を指定し、インスタンスを生成後にloginメソッドを呼び出すと、ログイン処理が実行される。

XPathの発行

DBに対し、クエリを発行する共通処理のソースを**LIST7**に示す。NeoCoreXMSへの検索は、SessionManagedNeoConnectionクラスのqueryXMLメソッドを呼び出す。引数には、XQuery式、またはXPath式を指定する。

ファイルの一時フォルダへのコピー／解凍

サンプルのバッチ処理では、Wordファイルを検知した後にファイルを一時フォルダにコピーしてZIPで解凍する。その処理は、WordAggregateクラスのcopyTransferメソッドとunZipメソッドに記述されている。ここでは、誌面の都合により詳細は割愛する。実際のソースはサンプルプログラムを参照してほしい。

ステップ3-3-1

XML文書をXML-DBに新規に保存する

ローカルPCのWordファイルがXML-DB上に存在しない場合、XML-DBにデータを新規に保存（Store）する。**LIST8**に、XML-DBにデータを保存するstoreOpenXMLメソッドを示す。XML文書は、document.xmlの内容をXML本体、core.xmlの内容をXML文書本文以外で補足的なデータを格納するためのPrefix XMLとして保存する。

本稿では、core.xmlのルート要素<cp:coreProperties>の子要素として、<dir_name>要素と<file_name>要素を独自に追加している。dir_name要素は1つのフォルダ内にあるファイル一覧を取得するため、<file_name>要素はローカルPC内のファイル名を検索一覧画面に表示する際に使用する。WordAggregateクラスのstoreOpenXMLメソッドで、XML文書を新規に保存する際に、NeoCoreClientクラスのstoreXMLメソッドを呼び出している（**LIST9**）。

storeXMLメソッドはNeoCoreClientクラスに2つ用意されており、違いは第1引数にSessionManagedNeoConnectionのインスタンスがあるかないかである。dbが引数にないメソッドは、メソッド内でトランザクションの開始／コミットが完結する。もう一方のメソッドは、storeXML処理を実行するだけでトランザクションはその呼び出し元で制御する。

```
<Query-Results>
  <cp:coreProperties>
    <dc:title/>
    <dc:subject/>
    <dc:creator>haru</dc:creator>
    <cp:keywords/>
    <dc:description/>
    <cp:lastModifiedBy>haru</cp:lastModifiedBy>
    <cp:revision>35</cp:revision>
    <dcterms:created xsi:type="dcterms:W3CDTF">2008-11-06T13:15:00Z</dcterms:created>
    <dcterms:modified xsi:type="dcterms:W3CDTF">2008-12-04T12:37:00Z</dcterms:modified>
    <file_name>C:¥XMLDB¥doc_root¥設計¥sample3.docx</file_name>
    <dir_name>C:¥XMLDB¥doc_root¥設計</dir_name>
  </cp:coreProperties>
  // 以降、複数件ある場合は、cp:coreProperties要素の繰り返し
</Query-Results>
```

LIST5：XML-DB検索結果のXML

```
public static SessionManagedNeoConnection getConnection() throws Exception{
    SessionManagedNeoConnection db = new SessionManagedNeoConnection("localhost", 7701);
    System.out.println("Start LOGIN");
    String sid = db.login("Administrator", "admin");
    System.out.println("End LOGIN session id:" + sid);
    return db;
}
```

LIST6：NeoCoreXMSのコネクション取得処理（NeoCoreDBConnectionFactory.java）

```
public static Document queryXML(String query) throws Exception{
    // ログイン済みコネクションの取得
    SessionManagedNeoConnection db = NeoCoreDBConnectionFactory.getConnection();
    String queryResult = db.queryXML( query );
    // 文字列のXMLをDocumentオブジェクトに変換する
    return XMLUtil.string2Document(queryResult);
}
```

LIST7：NeoCoreXMSへのクエリ発行処理（NeoCoreClient.java）

```
private static String storeOpenXML(
    String documentXMLFileName,String coreXMLFileName,String schemaFileURL,File srcFile)
    throws Exception{

    // 引数で渡されたファイル名のXMLデータを文字列で取得する
    String xmlStr = XMLUtil.xmlString( documentXMLFileName );

    // prefixファイル docProps/core.xml
    Document coreXMLDoc = XMLUtil.createXMLDocument( coreXMLFileName );
    Element coreXMLRootElem = coreXMLDoc.getDocumentElement();

    // XMLのノードにWord文書の物理ファイル名を要素としてappendする
    Element fileNameElem = coreXMLDoc.createElement( "file_name" );
    fileNameElem.appendChild(coreXMLDoc.createTextNode( srcFile.getAbsolutePath() ));
    coreXMLRootElem.appendChild(fileNameElem);
    // XMLのノードにWord文書のフォルダ名を要素としてappendする
    Element dirNameElem = coreXMLDoc.createElement( "dir_name" );
    dirNameElem.appendChild(coreXMLDoc.createTextNode( srcFile.getParentFile().getAbsolutePath() ) );
    coreXMLRootElem.appendChild( dirNameElem );

    //core.xmlの文字列を取得
    String coreXMLStr = XMLUtil.node2String(coreXMLRootElem);
    // XMLDBへの保存処理
    return NeoCoreClient.storeXML( xmlStr , schemaFileURL , coreXMLStr );
```

LIST8：Office Open XMLファイル新規保存処理（WordAggregate.java）

ステップ3-3-2

XMLデータの修正

ローカルPC内にあるWord文書の修正の有無を判断するには、ローカルPCのcore.xmlのcp:c

oreProperties/dc:modified要素の値と、XML-DBから取得したXML文書の修正時間を比較する。この2つの時間が異なる場合、XML-DB上のXMLデータを修正する(**LIST10**)。

modifyOpenXMLメソッドでは、XML-DBに文書として格納されている文書のうちのPrefix XML(core.xmlの内容を格納)と、XML文書本体(document.xmlを格納)の2箇所を修正する。

まず、Prefix XMLを修正する処理はNeoCoreClientクラスのmodifyXMLメソッド(**LIST11**)を呼び出し、更新対象のノードを指定するクエリ文字列を第1引数、更新後のXML文字列を第2引数で渡している。次に、XML文書本体を格納する処理では更新後のXMLを既存のdocument要素の兄弟要素として挿入(insert)した後、既存のXMLを削除している。

挿入にはNeoCoreClient.insertXMLのinsertXMLメソッド(**LIST12**)を、削除にはNeoCoreClient.deleteXMLメソッドをそれぞれ呼んでいる。document要素を直接更新せずにこのような方法を採用しているのは、modifyメソッドは更新前XMLのノード構造と更新後XMLのノード構造が異なるとエラーになるためである。document.xmlのルート要素であるdocument要素以下のノード構造はWord文書の内容によってさまざまに変化するため、modifyメソッドは使用することはできない[注]。

なお、modifyXMLメソッド、insertXMLメソッド、deleteXMLメソッドは、ほぼ同じソースコードとなるため、本稿ではmodifyXML/insertXMLメソッドのソースのみを示す。deleteXMLメソッドのソースは、サンプルアプリケーションを参照してほしい。

```
public static String storeXML(
    String xmlStr , String schemaFileURL , String prefixXMLStr) throws Exception{
    SessionManagedNeoConnection db = null;
    try{
        // コネクション取得
        db = NeoCoreDBConnectionFactory.getConnection();
        db.startTransaction(); // トランザクション開始
        String result = storeXML( db , xmlStr , schemaFileURL , prefixXMLStr);
        db.commitTransaction(); // トランザクションコミット
        return result;

    }catch(Exception e){
        e.printStackTrace();
        // 例外が発生した場合、ロールバックする。
        if( db!=null ) db.rollbackTransaction();
        throw e;

    }finally{
        if (db != null ){
            db.destroy();
        }
    }
}
public static String storeXML(
    SessionManagedNeoConnection db,
    String xmlStr , String schemaFileURL , Stringprefix XMLStr) throws Exception{
    System.out.println("Start STORE");
    String docId = db.storeXML( xmlStr , schemaFileURL , prefixXMLStr );
    System.out.println("End STORE doc id:"+ docId );
    return docId;
}
```

LIST9：NeoCoreXMS Store処理実行(NeoCoreClient.java)

```
private static void modifyOpenXML(String wordFileName,String bodyXmlStr, String ⇒
coreXMLStr) throws Exception{
    SessionManagedNeoConnection db = null;

    try{
        db = NeoCoreDBConnectionFactory.getConnection();

        // Prefix XML Modify
        db.startTransaction();
        String result = NeoCoreClient.modifyXML(
            db ,"/ND/coreProperties[file-name='"+ wordFileName +"']" , coreXMLStr );

        //Body XML の修正
        // //document[1]の後ろにINSERT
        NeoCoreClient.insertXML(
            db ,"/ND[coreProperties/file-name='" + wordFileName + "']/document[1]" , bodyXmlStr );
        // // document[1]を削除
        NeoCoreClient.deleteXML(
            db , "/ND[coreProperties/file-name='" + wordFileName + "']/document[1]" );
        db.commitTransaction();

    }catch(Exception e){
        e.printStackTrace();
        if ( db != null ) db.rollbackTransaction();
        throw e;

    }finally{
        if (db!= null ){
            db.destroy();
        }
    }
}
```

LIST10：Office Open XMLファイル修正処理(WordAggregate.java)

バッチ処理の実行と実行結果の確認

バッチ処理をひととおり実装し終えたら、実際に処理を実行してみよう。

ファイルは、C:¥dbmag_sample¥doc_root配下に3つのWordファイルを配置している。バッチ処理のmainメソッドは、Eclipseから起動できる。起動するには、WordAggregate.javaを開いてEclipseのツールバーのアイコンにある実行ボタンを押下する。すると、コンソールに**画面5**のように実行結果が表示される。

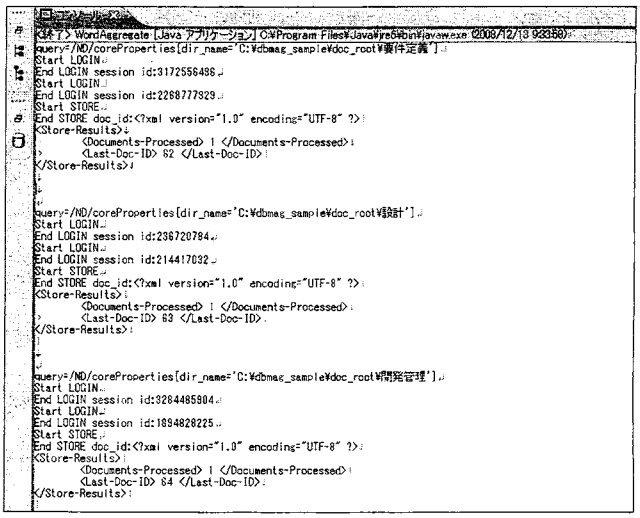
データ検索用Webアプリケーションの実装

ここでは、XML-DBに格納したOpen Office XMLを検索画面から検索するアプリケーションを実装する。検索条件として、ファイル作成者、キーワード検索(ファイルのプロパティで指定)、見出しレベル(段落スタイル)ごとのキーワード検索を用意しており、検索を行なうと結果を一覧表示する。

注：Prefix XMLに格納されるcore.xmlも内容によってノード構造が変化する。そのため、本来はmodifyメソッドを使用できないが、modifyメソッドを紹介するために使用している。

```
public static String modifyXML(
    String query , String modifyXML) throws Exception{
    SessionManagedNeoConnection db = null;

    try{
        db=NeoCoreDBConnectionFactory.getConnection();
        db.startTransaction();
        String result = modifyXML( db , query , modifyXML );
        db.commitTransaction();
        return result;

    }catch(Exception e){
        e.printStackTrace();
        if(db != null ) db.rollbackTransaction();
        throw e;

    }finally{
        if(db != null ){
            db.destroy();
        }
    }
}
public static String modifyXML(
    SessionManagedNeoConnection db,String query , String modifyXML)
    throws Exception{

    // queryで修正対象のNodeのXPathを指定する
    String result = db.modifyXML(query , modifyXML);
    return result;
}
```

LIST11：NeoCoreXMS Node修正処理（NeoCoreClient.java）


WordAggregate [Java アプリケーション] C:¥Program Files¥Java¥jre6¥bin¥javaw.exe (2008/12/13 9:33:50)
query=/ND/coreProperties[dir_name='C:¥dbmag_sample¥doc_root¥要件定義']
Start LOGIN
End LOGIN session id:3172556486
Start LOGIN
End LOGIN session id:2268777929
Start STORE
End STORE doc_id:<?xml version="1.0" encoding="UTF-8" ?>
<Store-Results>
<Documents-Processed> 1 </Documents-Processed>
<Last-Doc-ID> 82 </Last-Doc-ID>
</Store-Results>
query=/ND/coreProperties[dir_name='C:¥dbmag_sample¥doc_root¥設計']
Start LOGIN
End LOGIN session id:236720784
Start LOGIN
End LOGIN session id:214417032
Start STORE
End STORE doc_id:<?xml version="1.0" encoding="UTF-8" ?>
<Store-Results>
<Documents-Processed> 1 </Documents-Processed>
<Last-Doc-ID> 83 </Last-Doc-ID>
</Store-Results>
query=/ND/coreProperties[dir_name='C:¥dbmag_sample¥doc_root¥開発管理']
Start LOGIN
End LOGIN session id:3284485804
Start LOGIN
End LOGIN session id:1894828225
Start STORE
End STORE doc_id:<?xml version="1.0" encoding="UTF-8" ?>
<Store-Results>
<Documents-Processed> 1 </Documents-Processed>
<Last-Doc-ID> 84 </Last-Doc-ID>
</Store-Results>
```

画面5：Wordファイル収集／格納バッチの実行

以下に、各処理について説明する。

## 検索条件格納用クラス

WordSearchConditionクラスは、画面で入力された検索条件を格納するためのクラスである。画面の検索フィールドをインスタンス変数として保持している（**LIST13**）。

## 検索処理実行クラスと検索結果格納用クラス

WordProcessingMLSearchクラスは検索処理を実行するクラスであり、searchメソッドに検索処理が実装されている（**LIST14**）。

searchメソッドでは、引数で渡された検索条件（WordSearchConditionのインスタンス）をもとにXPath式を組み立て、NeoCoreClientクラスのqueryXMLメソッドを呼び出す。queryXMLメソッドは<Query-Results>要素をルートとするDocumentインスタンスを返すので、そのインスタンスを解析し、検索結果格納用のWordSearchResultクラス（**LIST15**）の集合を作成する。検索条件入力画面ですべての条件を入力すると、次のようなXPath式になる。

```
/ND[coreProperties/creator='haru'
    and coreProperties/keywords='設計'
    and document/body/p/pPr/pStyle/@val='1'
    and document/body/p/r/t='レベル1のタイトル']
```

```
public static String insertXML(String query , String insertXML) throws Exception{
    SessionManagedNeoConnection db = null;

    try{
        db = NeoCoreDBConnectionFactory.getConnection();
        db.startTransaction();
        String result = insertXML( db , query , insertXML );
        db.commitTransaction();
        return result;

    }catch(Exception e){
        e.printStackTrace();
        if( db != null ) db.rollbackTransaction();
        throw e;

    }finally{
        if(db != null ){
            db.destroy();
        }
    }
}
```

LIST12：NeoCoreXMS Node挿入処理（NeoCoreClient.java）

```
public class WordSearchCondition {

    /** ファイル作成者 */
    private String creator = null;
    /** キーワード */
    private String keywords = null;
    /** 見出しレベル 段落スタイル指定 */
    private String pStyle = null;
    /** 見出しレベル内で検索する単語 */
    private String pStyleSearchWord = null;

    public String getCreator(){
        return creator;
    }

    public void setCreator(String creator){
        this.creator = creator;
    }
    // 以下、keywords、pStyle、Creatorのセッター／ゲッターメソッドは省略
}
```

LIST13：XML-DB検索条件格納用クラス（WordSearchCondition.java）

Servletの作成

最後に、ブラウザからの検索リクエストを受け付けるServletを実装する（**LIST16**）。

Servletではリクエストパラメータを WordSearchConditionに格納し、WordProcessingMLSearchのsearchメソッドを呼び出している。sear
```

```
public List<WordSearchResult> search(
    WordSearchCondition cond) throws Exception{
    StringBuffer query = new StringBuffer("");

    // Creator検索
    if ( cond.getCreator() != null && !cond.getCreator().equals("")){
        query.append(
              "/ND[coreProperties/creator='"
              + cond.getCreator() + "'" );
    }

    // キーワード検索
    if ( cond.getKeywords() !=
        null && !cond.getKeywords().equals("")){
        if ( 0 < query.toString().length() ){
            query.append( " and " );
        }else{
            query.append( "/ND[" );
        }
        query.append(
          "coreProperties/keywords='" + cond.getKeywords() + "'" );
    }

    // アウトライン　レベル指定検索
    if ( cond.getPStyle() != null ){
        if ( 0 < query.toString().length() ){
            query.append( " and " );
        }else{
            query.append( "/ND[" );
        }
        query.append(
            "document/body/p/pPr/pStyle/@val=
            '" + cond.getPStyle() + "' and document/body/p/r/t=
            '" + cond.getPStyleSearchWord() + "'")
    }

    // 検索条件が指定されなかった場合、nullを返す
    if (query.toString().length() == 0 ) return null;

    // 最後にかっこを閉じる
    query.append("]");

    List<WordSearchResult> resultList = new
        ArrayList<WordSearchResult>();
    Document queryReault = NeoCoreClient.queryXML( query.toString() );
    Element rootElem = queryReault.getDocumentElement();

    NodeList ndNodeList = rootElem.getElementsByTagName("ND");
    int resultCnt = ndNodeList.getLength();
    for (int i = 0; i < resultCnt; i++) {
        WordSearchResult result = new WordSearchResult();
        Element ndElem = (Element)ndNodeList.item( i );
        Element corePropElem
          = XMLUtil.getChildElement(ndElem, "cp:coreProperties" );
        result.setModified(
          XMLUtil.getChildElement(
              corePropElem, "dcterms:modified" ).getTextContent() );
        result.setLastModifiedBy(
          XMLUtil.getChildElement(
            corePropElem,"cp:lastModifiedBy").getTextContent() );
        result.setFileName(
          XMLUtil.getChildElement(
              corePropElem,"file_name").getTextContent());
          resultList.add(result);
        }
        return resultList;
    }
```

LIST14：XML-DB検索クラスの検索用メソッド（WordProcessingMLSearch.java）

```
public class WordSearchResult {
    /** ファイル名 */
    private String fileName = null;

    /** 最終更新者 */
    private String lastModifiedBy = null;

    /** 最終更新日時 */
    private String modified = null;

    public String getFileName() {
        return fileName;
    }
    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public String getFileName() {
        return fileName;
    }
    public void setFileName(String fileName) {
        this.fileName = fileName;
    }
    // 以下、lastModifiedByとmodifiedのセッター／ゲッターメソッドは省略
}
```

LIST15：XML-DB検索結果格納クラス（WordSearchResult.java）

```
public class OpenXMLSearchServlet extends HttpServlet {

    protected void doPost(
        HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

        // 検索条件をリクエストパラメータから格納する
        WordSearchCondition cond = new WordSearchCondition();
        cond.setCreator( req.getParameter( "creator" ) );
        cond.setKeywords( req.getParameter( "keywords" ) );
        cond.setPStyle(req.getParameter( "pstyle" ) );
        cond.setPStyleSearchWord(req.getParameter(
            "pstyle_search_word" ) );
        WordProcessingMLSearch wordMlSearch =
            new WordProcessingMLSearch();

        // 検索処理実行
        List<WordSearchResult> wordSearchResult = null;
        try {
            wordSearchResult = wordMlSearch.search(cond);
        } catch (Exception e) {
            throw new ServletException( e );
        }
        // requestに格納
        if ( wordSearchResult != null && 0 < wordSearchResult.
size() ){
            req.setAttribute(
                "searchResultList" , wordSearchResult );
        }
        req.getRequestDispatcher(
            "/result.jsp" ).forward(req, resp);
    }
```

LIST16：XML-DB検索Servlet（WordSearchResult.java）

chメソッドの戻り値は、WordSearchResultの集合を格納したListで、JSPで表示するためにServletRequestのattributeに設定している。

Servletの登録は、サンプルアプリケーションのweb.xmlを参照してほしい。

実行結果の確認

以上の処理を実装したら、Tomcatを起動してブラウザを立ち上げ、次のURLにアクセスする。

http://localhost:8080/openxml/search.jsp

すると、検索条件入力画面が表示される。検索条件を入力後、[実行]ボタンを押下すると検索結果一覧画面が表示される（画面1）。

まとめ

本稿では、Office Open XMLとXML-DBのNeoCoreXMSを連携させて、Office文書をデータとして活用するソリューションの一例を紹介した。企業の知的資産であるOffice文書は、ややもすればファイルサーバーに保存したきり眠ったままの状態になり、その価値を活かしきれなくなりがちである。Office文書のフォーマットがXML化されてオープンになることにより、データとしての活用の幅が広がっていく。本稿がそのヒントやきっかけとなれば幸いである。 DBM

本誌付録CD-ROMに、本稿のサンプルコードを収録しています。またDBマガジンのWebサイト（http://www.shoeisha.com/mag/dbm）からもダウンロードできます。

佐藤治夫（さとうはるお）
株式会社ビープラウド代表取締役。日頃は会社の経営者とエンジニアの2足の草鞋を履く。また、執筆コミュニティ「WINGSプロジェクト」に所属し、ライターとして主にWeb系の技術記事を執筆。システム開発時に使用する主なプログラミング言語は、python、Java、perlで、ITソリューションの1手段としてのXML-DBの有効性に着目し、その活用を推進している。

山田祥寛（やまだよしひろ）
執筆コミュニティ「WINGSプロジェクト」の代表。WINGSプロジェクトでは海外記事の翻訳から、主にWeb開発分野の書籍／雑誌／Web記事の執筆、講演など幅広く手がける。2008年12月時点での登録メンバーは30名。現在も一緒に執筆できる有志を募集中。興味のある方はどしどし応募してほしい。