XSL-FO による XML ドキュメント印刷のための スタイルシート作成方法

2008 年 5 月改訂 8 版

A Data Usability Company
アンテナハウス

アンテナハウス株式会社

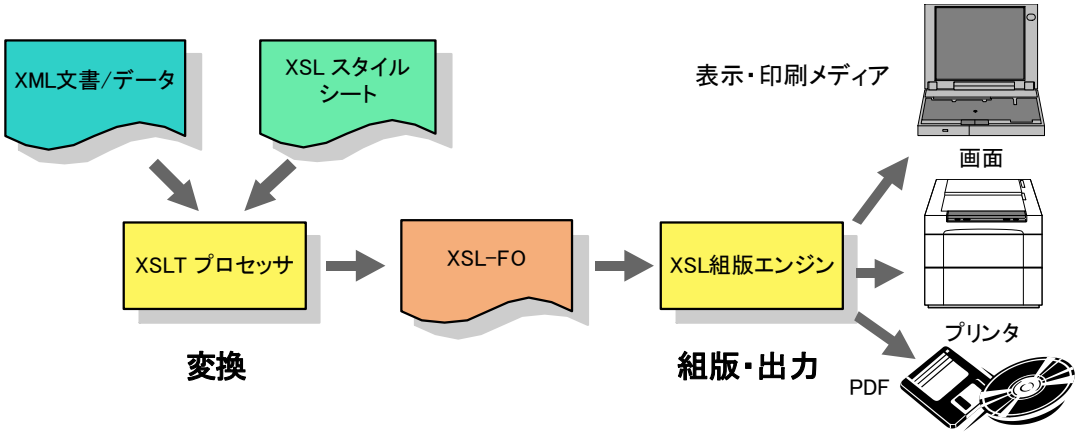
目 次

はじめに

「Extensible Stylesheet Language 仕様」（略称：XSL 仕様）は XML ドキュメントを表示・印刷するための仕様として W3C が 2001 年 10 月にバージョン 1.0 を勧告したものです。XSL 仕様は、その後改訂され、2006 年 11 月 5 日にバージョン 1.1 が勧告となりました。XSL 仕様は XML ドキュメントを表示・印刷するためのオブジェクトとプロパティを定義しています。このため、この仕様に基づいて作成された結果を XSL-FO（XSL-Formatting Object）と呼ぶのが通例です。また XSL-FO で、XSL 仕様を指す場合もあります。

さて XML ドキュメントから、この XSL-FO を作成して印刷するには、次の手順が一般的です。

1. XML ドキュメントの DTD に対して、目的の出力を実現する XSL スタイルシートを作成する。
2. XML ドキュメントと XSL スタイルシートの２つを入力として XSLT プロセッサに与え XSL-FO を作成する。
3. XSL-FO を処理する組版エンジンで、印刷や PDF 出力などの目的の結果を得る。



XSL-FO の作成とフォーマッタによる表示・印刷

XSL-FO を出力するスタイルシートを作成するためには、XSLT と XSL-FO の知識が不可欠です。

XSLT については仕様書（参考資料［3］）、その日本語訳（参考資料［4］）の他に多数の参考書がでています。XML から HTML への変換などでもしばしば使われますので、既にご存知の方も多いでしょう。XSLT はすでにバージョン 2.0 が勧告となっていますが、スタイルシートはバージョン 1.0 の範囲で記述するものとします。

一方、XSL 仕様については、英文の仕様書（参考資料［1］）JIS の日本語訳（参考資料［2］）が出されています。XSL-FO の仕様は非常に膨大な内容で、A4 サイズで 500 ページを超える量(1)になります。この仕様全体を理解するのは非常に大変です。しかし XSL 仕様は、基本的には実装処理系を作成するためのものです。処理系を利用する側では、必ずしもそのすべてを知る必要はありません。一定の知識とパターンを身に付ければ、十分スタイルシートの作成はできるでしょう。

本稿では XML ドキュメントを XSL-FO に変換するための XSL スタイルシートの作成を解説します。簡単な文書を記述するためのフォーマットとして SimpleDoc を使います。SimpleDoc のベースは浅見智晴氏が作成した PureSmartDoc （参考資料［5］）です。サンプルとするために要素の種類を減らし、文書の記述と組版に便利な機能を追加しました。

本稿ではこの SimpleDoc 文書を XML ドキュメントの例とします。本稿自体が SimpleDoc.dtd のインスタンス XML 文書であり、ここで解説している、SimpleDoc 文書から XSL-FO に変換するスタイルシートを使って、XSL Formatter によって組版できます。

(1) W3C にある XSL 1.1 の PDF 版では 514 ページでした。（http://www.w3.org/TR/2006/REC-xsl11-20061205/xsl11.pdf）

XSL スタイルシート作成のステップ

XSL-FO に変換するためのスタイルシートの作成は、どのようなステップを踏むのでしょうか？ 簡単に整理すると以下のようになります。

ステップ	内容
XML 文書の構造を知る	まず入力仕様にあたる XML 文書の構造に関する情報が必要です。XSLT プロセッサによる変換処理では DTD が存在しなくとも XSL-FO を作ることができます。(2)しかし要素やプロパティの種類・内容、出現順序など、DTD に記述された情報はスタイルシートを作成する上ではどうしても必要です。
印刷形式の仕様を作成する	最終結果として得られる印刷物の形式で、いわば出力仕様にあたります。XSL 仕様は組版のための仕様です。印刷形式の仕様は用紙のサイズとレイアウト、見出しや本文の体裁設定、目次や索引の有無など、多岐にわたります。
印刷形式を XSL-FO にあてはめる	印刷形式の仕様が決定されれば、その形式で印刷するためには、どのような XSL 仕様のオブジェクトとプロパティを適用するのかを知らなければなりません。これはできあいのスタイルシートを手がかりに指定方法に習熟してゆくのがよいでしょう。
XSL スタイルシートを作成する	入力の XML 文書を目的の印刷形式に変換するための処理を XSL スタイルシートで記述します。入力 XML 文書を、出力仕様を実現する XSL-FO にマッピングします。スタイルシートの記述は、一般のプログラミング言語と同じ側面もありますが、XSLT の特性を知らないと難しい分野(3)もあります。

(2) 基本的に XSLT プロセッサでは入力 XML 文書に文書型宣言があっても妥当性の検証を行いません。しかし、文書型宣言で宣言された実体宣言を XML 文書中で実体参照で使用している場合には DTD が必要になります。

(3) XSLT では、変数に初期値を設定することはできますが、再度その変数に値を代入することはできません。また、ループを再帰呼び出しを使用して実現するテクニックも必要になります。

SimpleDoc の構造

最初に SimpleDoc の構造の概略を次の表に示します。詳細は SimpleDoc.dtd を参照ください。

要素	意味	定義
block 要素並び	―	p | ul | ol | dl | table | program | pre | div | hidden
inline 要素並び	―	a | note | span | figure | b | i | em | code | br | icon | index | underline | ref
doc	ルート要素	(head, body)
head	ヘッダー	(date | author | position | abstract | title)*
date, author, abstract, title	ヘッダーの構成要素：作成日、著者、要約、表題	(#PCDATA | inline 要素並び)*
body	文書本体	(part | chapter | section | appendix | (%block;) | (%inline;) | newpage)*
part	部	(title, (chapter | block 要素並び | inline 要素並び | newpage)*)
chapter	章	(title, (section | block 要素並び | inline 要素並び | newpage)*)
section	節	(title, (subsection | block 要素並び | inline 要素並び | newpage)*)
subsection	副節	(title, (subsubsection | block 要素並び | inline 要素並び | newpage)*)
subsubsection	副々節	(title, (block 要素並び | inline 要素並び | newpage)*)
appendix	付録	(title, (bib | block 要素並び | inline 要素並び | newpage)*) 付録には参考資料一覧を置くことができます。
title	タイトル	(#PCDATA | inline 要素並び)*
p	段落	(#PCDATA | block 要素並び | inline 要素並び)*
ul	番号なしリスト	(li*) type プロパティで行頭文字を指定できます。
ol	番号付リスト	(li*) type プロパティでリストのラベル部分の番号書式を指定できます。
bib	参考資料リスト	(li*) 巻末に参考資料の一覧を作成するためのリストです。
dl	定義型リスト	(dt, dd)* type プロパティで横並びのブロックにフォーマットするのか、縦並びのブロックにフォーマットするかを指定できます。
dt	定義型リストの用語部分	(#PCDATA | ブロック要素並び | インライン要素並び)*
dd	定義型リストの定義部分	(#PCDATA | ブロック要素並び | インライン要素並び)*
table	テーブル全体	(title?, col*, thead?, tfoot?, tbody) layout プロパティでテーブルを自動レイアウトするか否か(auto/fixed)を指定します。width プロパティでテーブル全体の幅を指定します。rowheight プロパティでテーブル全体にわたる行の高さを指定します。
col	列プロパティ	EMPTY number プロパティで列番号、width プロパティで列幅を指定します。
thead	テーブルヘッダ	(tr*)
tfoot	テーブルフッタ	(tr*)
tbody	テーブル本体	(tr*)
tr	テーブルの行	(th | td)* height プロパティで行の高さを指定できます。
th	ヘッダセル	(inline 要素並び)* colspan プロパティで横結合する列数, rowspan プロパティで縦結合する行数を指定できます。align, valign プロパティで横、縦方向の揃えを指定できます。

要素	意味	定義
td	データセル	(inline 要素並び)* colspan プロパティで横結合する列数, rowspan プロパティで縦結合する行数を指定できます。align, valign プロパティで横、縦方向の揃えを指定できます。
program	プログラムコード	(#PCDATA | title)*
div	汎用ブロック要素	(title, (汎用ブロック要素 | 汎用インライン要素)*) class プロパティで種類を拡張します。
a	アンカー要素（リンク）	(#PCDATA | inline 要素並び)* href プロパティでリンク先 URI を指定します。
note	注釈	(#PCDATA | inline 要素並び)*
b	太字	(#PCDATA | inline 要素並び)*
i	斜体	(#PCDATA | inline 要素並び)*
em	強調	(#PCDATA | inline 要素並び)*
code	インラインのプログラムコード	(#PCDATA | inline 要素並び)*
span	汎用インライン要素	(#PCDATA | inline 要素並び)*
figure	図	(title?) src プロパティでファイルを指定します。
br	改行	EMPTY
ref	参考資料への参照番号	EMPTY ref-id プロパティに参考資料の ID を設定します。
index	索引項目	#PCDATA key プロパティでグループ化用の文字を指定します。

特徴は次のとおりです。

- part 〜 subsubsection にいたる文書構造は PureSmartDoc と同じです。文書は part から書き始めることも、section から作成することもできます。様々な規模の文書に対応できるよう、柔軟な構造を持っています。
- ブロック要素とインライン要素は、PureSmartDoc より要素数を減らし、最低限のものとしました。汎用ブロック要素の div、汎用インライン要素の span の class プロパティにより、様々な拡張ができるように考慮してあります。
- テーブルのセルやリストの要素内で改行ができるように、また段落（p）内でも段落を終了せずに改行ができるように br 要素を追加しました。
- リストやテーブルでは、プロパティ値でその出力形式をある程度指定できるようにしました。
- 参考資料一覧の作成、索引の作成方法について説明するために特別に bib, ref, index などの要素を用意しています。

Hello! World

SimpleDoc 文書から XSL-FO への変換の最も簡単な例

まず SimpleDoc 文書から XSL-FO に変換する XSL スタイルシートの最も簡単な例を次に示します。

入力 XML 文書（Hello.xml）

```
<?xml version="1.0" encoding="Shift-JIS" ?>
<doc>
  <head>
    <title>サンプル</title>
  </head>
  <body>
    <p>Hello World!</p>
    <p>はじめての<b>SimpleDoc</b>です。</p>
  </body>
</doc>
```

XSL-FO 変換のスタイルシート（Sample.xsl）

```
<?xml version="1.0" encoding="Shift-JIS" ?>
<xsl:stylesheet version="1.0"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes" />

<xsl:template match="doc">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master page-height="297mm" page-width="210mm"
        margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
        <fo:region-body   margin="20mm 0mm 20mm 0mm" />
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="PageMaster">
      <fo:flow flow-name="xsl-region-body">
        <fo:block>
          <xsl:apply-templates select="body" />
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="body">
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="p">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="b">
  <fo:inline font-weight="bold">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
</xsl:stylesheet>
```

生成された XSL-FO

```
<?xml version="1.0" encoding="UTF-16"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master page-height="297mm" page-width="210mm"
      margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
      <fo:region-body margin="20mm 0mm 20mm 0mm" />
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="PageMaster">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>
        <fo:block>Hello World!</fo:block>
        <fo:block>はじめての
          <fo:inline font-weight="bold">SimpleDoc</fo:inline>です。
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

上の XSL-FO は、次のように組版/表示されます。

Hello World!
はじめての **SimpleDoc** です。

スタイルシートの構造

Sample.xsl と生成された XSL-FO を見ると次のことがわかります。

- スタイルシートはテンプレートの集合です。ルート要素の xsl:stylesheet の下位は xsl:template 要素から構成されています。各テンプレート xsl:template は match="xxx"で入力 XML 文書の xxx タグを処理するよう対応付けられています。
- 各テンプレートでは、必要な XSL-FO のオブジェクトと入力要素のテキストが出力されます。そして xsl:apply-templates 命令により、子要素とテキストに対応するテンプレートが呼び出されます。(4)

XSLT プロセッサは、入力 XML 文書を読み込み、そのルートノードから処理を開始します。要素を処理するテンプレートを探し、テンプレートに記述された処理を行います。そして再帰的に次々と子要素を処理して、ルート要素に戻って処理対象がなくなったら終了します。

ブロック要素とインライン要素

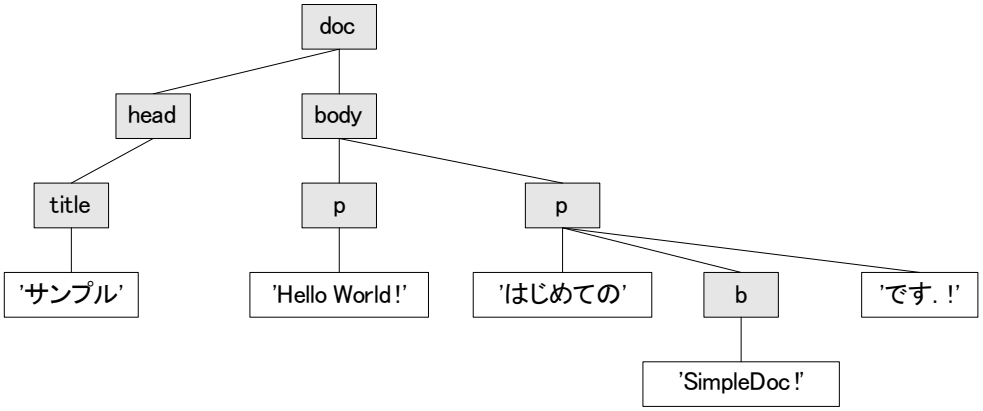
次に注目していただきたい点は、ブロック要素とインライン要素の対応付けです。

- スタイルシートを見ると、p 要素は fo:block オブジェクトに、b 要素は fo:inline オブジェクトに変換しています。XSL-FO への変換の基本は、入力 XML 文書の要素をレイアウト意図によりブロック・オブジェクトかインライン・オブジェクトに変換することです。
- 一般的に終了タグで改行したい要素は、fo:block オブジェクトにマッピングします。終了タグで改行しない要素は fo:inline オブジェクトにマッピングします。fo:inline オブジェクトには、何らかの修飾プロパティを指定します。ここでは b 要素は太字を意味しているので、書体をボールドに設定しました。

(4) テキストに対応するテンプレートは記述されていませんが、この場合は XSLT のビルトインテンプレート規則が適用され、テキストノードは結果にコピーされます。

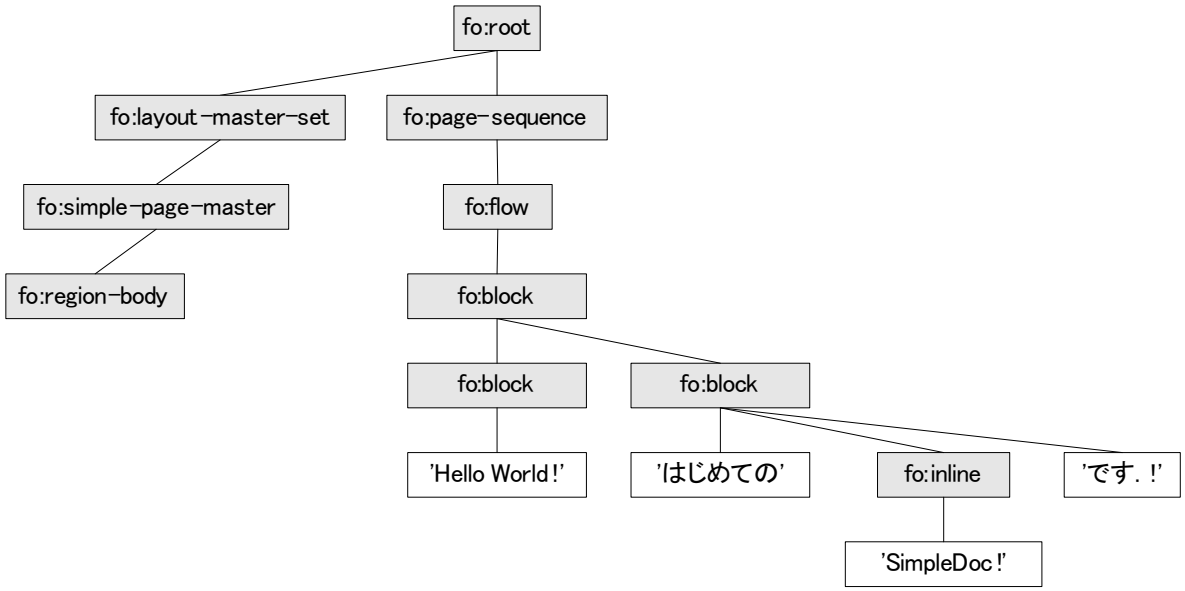
FO ツリーの構造

次に注目していただきたい点は FO ツリーの構造ですが、まず XML 文書のツリー構造を見てみましょう。



Hello.xml のツリー構造

それに対し、XSL-FO のツリー構造は以下のようになっています。FO ツリーはルートが fo:root で、その子供に fo:layout-master-set と fo:page-sequence があります。fo:layout-master-set は、ページ書式の定義部で、fo:page-sequence はページに配置する実データです。



XSLT 処理後の XSL-FO ツリー

ページ書式を定義する fo:layout-master-set は実際の組版データの fo:page-sequence より前(preceding-sibling) でなければなりません。XSL プロセッサは、入力の XML 文書をルート要素からたどり、対応するテンプレート (xsl:template) を探して処理をはじめます。したがって、一般的に**fo:layout-master-set は、入力 XML 文書のルート要素を処理するテンプレートで出力**する必要があります。この例では<xsl:template match="doc">がこの処理を行っています。

fo:flow 以降は要素名が変わっただけで、元の文書と同じツリー構造です。元の文書に**存在するもの**を<xsl:template match="xxx">～<xsl:apply-templates />で、そのまま引き写すだけなら、このような結果になります。また Sample.xsl では、XML 文書中の<head>～</head>の情報が出力に現れません。これは<xsl:stylesheet match="doc">テンプレート中で、<xsl:apply-templates select="body" />として処理対象の子要素を<body>とし、<head>を除外しているからです。スタイルシートでは、このように処理対象を意図的にコントロールすることができます。

実用的な XSL スタイルシートの設計

印刷形式の仕様

前章の Sample.xsl では実用的な出力結果を得ることはできません。次に、実用的な組版を行うためのスタイルシートの作成方法を説明します。全体の構成は次のようにします。

【文書書式】

項目	仕様
用紙サイズ	A4 用紙（210mm×297mm）
用紙方向	縦置き
書字方向	すべて lr-tb（文字は左から右、行は上から下へ）
構成	先頭から順に表紙、目次、本文、索引の順とする。
ヘッダ・フッタ	表紙、目次、索引にはヘッダ、フッタは使わない。 本文のみヘッダとフッタを付ける。

【表紙・目次】

項目	仕様
用紙のマージン	上：25mm、下：25mm、左：25mm、右：25mm

【本文 】

項目	仕様
用紙のマージン	上：10mm、下：10mm、左：0mm、右：0mm
内容	part 〜 subsubsection に対応した見出し、表、箇条書き、段落、画像から構成。
書字方向	lr-tb
段数	1
基本文字サイズ	10pt
文字配置	両端揃え
その他の条件	ヘッダ領域とフッタ領域を配置する。フッタ領域の内容は小口寄りとして左右で切り替える。また、脚注領域と本文の間に境界線を配置する。境界線種は実線。本文領域の 1/3 の長さで、左寄りに配置。

【ヘッダ領域 】

項目	仕様
エクステント	10mm
書字方向	lr-tb
内容	文書の表題を印字する。 文字サイズ 9pt、文字送り方向は中央揃え、行送り方向は下揃え。ページ上部に爪インデクスを作成する。

【フッタ領域 】

項目	仕様
エクステント	10mm
書字方向	lr-tb
内容	ページ番号および現在ページの節タイトルを小口側に印字する。

【索引】

項目	仕様
用紙のマージン	上：25mm、下：25mm、左：25mm、右：25mm
段数	2
段間	20mm

XSL スタイルシートの構成

XSL スタイルシートは、次の５つのファイルから構成されます。

ファイル名	内容・用途
SD2FO-DOC.xsl	XSL スタイルシート本体
attribute.xsl	XSL-FO のプロパティをまとめて定義したファイル
param.xsl	用紙サイズなどの値をパラメータとして定義したファイル
index.xsl	索引を作成する処理をまとめたファイル
article.xsl	表紙、目次、索引の無い論文型書式の組版用スタイルシート

SD2FO-DOC.xsl は大別すると次のトップレベル XSLT 要素から構成されます。

XSLT 要素	内容・用途
xsl:include	機能別に分割されたスタイルシートをインクルードします。
xsl:param	スタイルシート全体で使用する用紙サイズなどの値をパラメータとして定義します。
xsl:attribute-set	ブロックやインラインなど、出力する XSL-FO のオブジェクトごとのプロパティをグループ化して定義したものです。
xsl:template match="xxx"	入力 XML 文書のタグ（"xxx"）ごとに記述した変換テンプレート定義です。 <xsl:apply-templates />で呼び出されます。
xsl:template name="yyy"	<xsl:call-template name="yyy" />で明示的に呼び出される、いわばテンプレートのサブルーチンです。
xsl:key	索引のための key を生成します。索引の作り方は後述します。

xsl:param、xsl:attribute-set はそれぞれ param.xsl、attribute.xsl の中で定義され、SD2FO-DOC.xsl においてインクルードされています。

スタイルシートを作成する際に、必ずしも xsl:param、xsl:attribute-set を使う必要はありません。しかし、以下の利点があります。

- xsl:attribute-set は XSL-FO のプロパティ、xsl:template は変換処理本体と役割分担させることによりスタイルシートを見やすくでき、メンテナンスが容易になります。
- xsl:param は、XSLT プロセッサの呼び出し側から値を渡すことができます。スタイルシートで xsl:param の値によって処理を分岐させれば、スタイルシートの処理を外部から制御することができます。

xsl:param の使用例

```
<!-- 目次を作成するか否かを決定します。 -->
<xsl:param name="toc-make" select="false()" />
<!-- 用紙サイズを定義します。 -->
<!-- 値は$paper-width, $paper-height で参照できます。 -->
<xsl:param name="paper-width">210mm</xsl:param>
<xsl:param name="paper-height">297mm</xsl:param>
```

xsl:attribute-set の使用例

```
<!-- 段落（p 要素）に対応する XSL-FO のプロパティを定義します。 -->
<!-- xsl:use-attribute-sets="p"で参照できます。 -->
```

```
<xsl:attribute-set name="p">
  <xsl:attribute name="text-indent">1em</xsl:attribute>
  <xsl:attribute name="space-before">0.6em</xsl:attribute>
  <xsl:attribute name="space-after">0.6em</xsl:attribute>
  <xsl:attribute name="text-align">justify</xsl:attribute>
</xsl:attribute-set>
```

以降ではこの SD2FO-DOC.xsl に沿って、スタイルシートを説明します。

ページ書式の設定

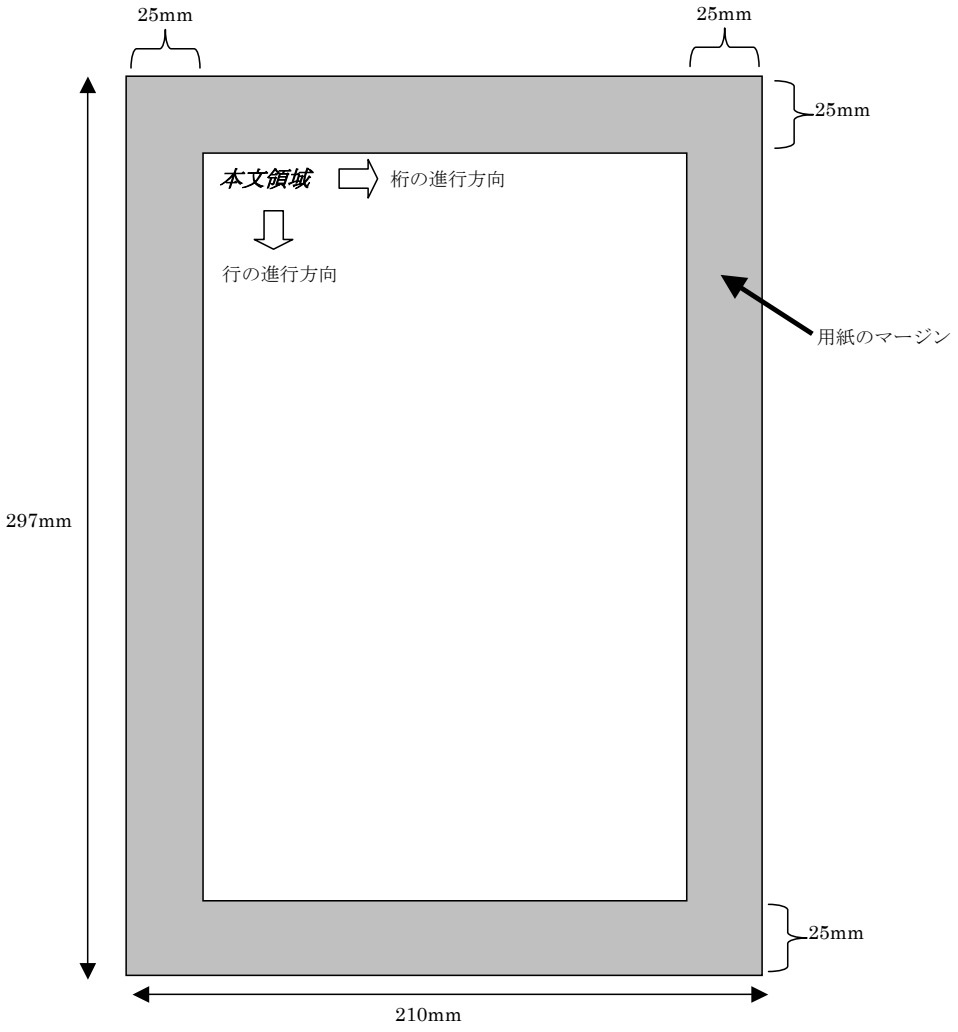
SD2FO-DOC.xsl のページ書式は次のような特徴を持ちます。

- 表紙、目次、本文、索引用のページ書式を持つ。
- 表紙、目次、索引用のページ書式はページ番号や文書名を入れないものとする。したがってヘッダ・フッタ領域は持たない。
- 本文用のページ書式は、左右で異なるページ書式とし、フッタ内容の印刷位置を小口寄りに配置する。
- ページ番号は本文の先頭を１ページとする。
- 索引ページのみ二段組とする。

したがって、表紙、目次、本文（左)、本文（右)、索引の５種類のページ書式が必要になります。以降で各ページ書式の定義方法を記述します。

表紙、目次のページ書式

表紙、目次のページ書式は次の図のようになります。



表紙、目次のページ書式

ページの書式はページマスタとして定義します。具体的には、fo:simple-page-master 要素を用いて、以下のように記述します。

スタイルシートのページ書式設定部分

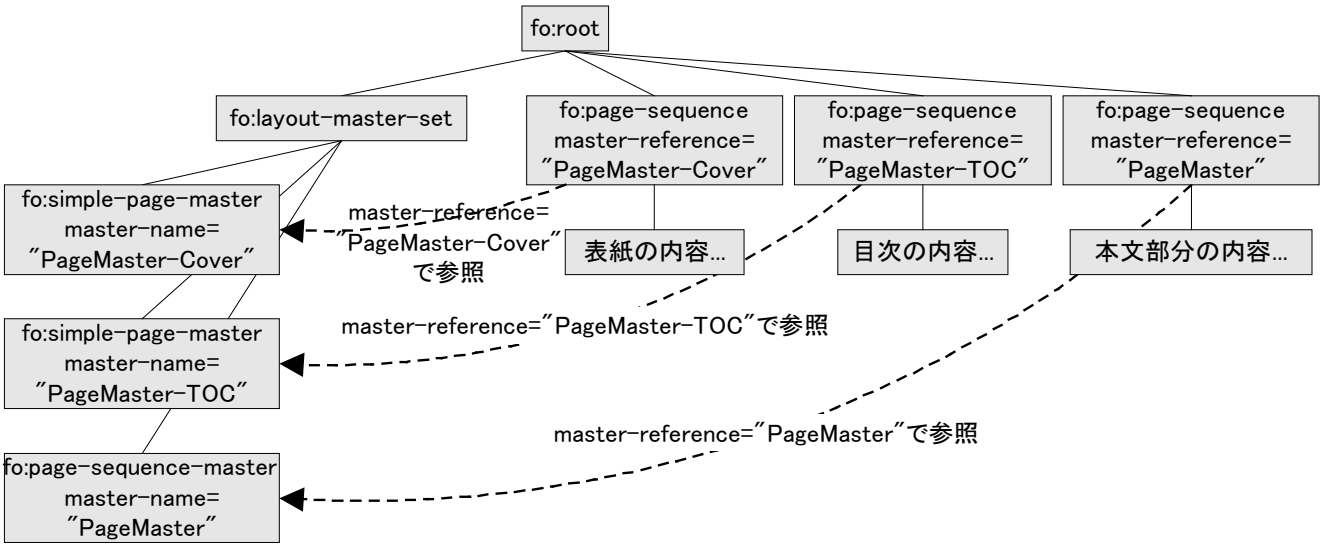
```
<fo:simple-page-master  margin="25mm 25mm 25mm 25mm"
  master-name="PageMaster-Cover">
  <xsl:attribute name="page-height">
    <xsl:value-of select="$paper-height" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
    <xsl:value-of select="$paper-width" />
  </xsl:attribute>
  <fo:region-body  margin="0mm 0mm 0mm 0mm" />
</fo:simple-page-master>

<fo:simple-page-master margin="25mm 25mm 25mm 25mm"
  master-name="PageMaster-TOC">
  <xsl:attribute name="page-height">
    <xsl:value-of select="$paper-height" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
    <xsl:value-of select="$paper-width" />
  </xsl:attribute>
  <fo:region-body  margin="0mm 0mm 0mm 0mm" />
</fo:simple-page-master>
```

設定している値は同じですが、変更の可能性も考えて、それぞれページマスタを用意します。

fo:simple-page-master の master-name との対応関係は次のとおりです。

master-name	用途	参照しているテンプレート
PageMaster-Cover	表紙用	<xsl:template match="doc/head">
PageMaster-TOC	目次用	<xsl:template name="toc">

これらのページマスタをどこで定義し、どこで参照するかという観点で XSL-FO ツリーの構造を示すと、次の図のようになります。



ページ書式から見た FO ツリーの構造

本文のページ書式 ― 左右ページ書式の切り替え

本文では左右でのページ書式切り替えを行います。XSL-FO では、偶数ページ書式と奇数ページ書式をグループにして交互に切り替えることで左右ページの書式切替ができます。

左ページ用と右ページ用のふたつの fo:simple-page-master を作成し、fo:page-sequence-master で 2 つをグループ化します。2 つのページ書式を交互に繰返すには fo:repeatable-page-master-alternatives を使います。偶数ページ用か奇数ページ用かは fo:conditional-page-master-reference の odd-or-even プロパティに odd（奇数）または even（偶数）を指定します。

スタイルシートでの記述は以下のようになります。

スタイルシートのページ書式設定部分

```
<fo:simple-page-master margin="10mm 00mm 10mm 00mm"
  master-name="PageMaster-Left">
  <xsl:attribute name="page-height">
    <xsl:value-of select="$paper-height-default" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
    <xsl:value-of select="$paper-width-default" />
  </xsl:attribute>
  <fo:region-body margin="15mm 25mm 15mm 25mm" />
  <fo:region-before region-name="Left-header"
    extent="10mm" display-align="after" />
  <fo:region-after region-name="Left-footer"
    extent="10mm" display-align="before" />
  <fo:region-start region-name="Left-start"
    extent="20mm" />
  <fo:region-end region-name="Left-end"
    extent="20mm" />
</fo:simple-page-master>

<fo:simple-page-master margin="10mm 00mm 10mm 00mm"
  master-name="PageMaster-Right">
  <xsl:attribute name="page-height">
    <xsl:value-of select="$paper-height-default" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
    <xsl:value-of select="$paper-width-default" />
  </xsl:attribute>
  <fo:region-body margin="15mm 25mm 15mm 25mm" />
  <fo:region-before region-name="Right-header"
    extent="10mm" display-align="after" />
  <fo:region-after region-name="Right-footer"
    extent="10mm" display-align="before" />
  <fo:region-start region-name="Right-start"
    extent="20mm" />
  <fo:region-end region-name="Right-end"
    extent="20mm" />
</fo:simple-page-master>

<fo:page-sequence-master master-name="PageMaster">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference
      master-reference="PageMaster-Left" odd-or-even="even" />
    <fo:conditional-page-master-reference
      master-reference="PageMaster-Right" odd-or-even="odd" />
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

横組みなので奇数ページを右にします。

索引のページ書式 ― 二段組

索引のページ書式は二段組です。XSL-FO では fo:region-body のプロパティ column-count に段数を指定します。つまりページ単位で段組の設定が可能ですが、ページの途中で段数を変更することはできません。ただし、fo:block にプロパティ span="all"を指定することでブロック・オブジェクトを全段抜きで配置することができます。段間の空き量は fo:region-body のプロパティ column-gap で指定します。スタイルシートでは以下のように記述します。

スタイルシートのページ書式設定部分

```
<fo:simple-page-master margin="25mm 25mm 25mm 25mm"
  master-name="PageMaster-index">
  <xsl:attribute name="page-height">
    <xsl:value-of select="$paper-height-default" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
    <xsl:value-of select="$paper-width-default" />
  </xsl:attribute>
  <fo:region-body margin="00mm 00mm 00mm 00mm"
    column-count="2" column-gap="20mm" />
</fo:simple-page-master>
```

スタイルシート全体の出力制御

- スタイルシートは、FO ツリーをページ書式部分（fo:layout-master-set）、表紙の内容、目次の内容、本文の内容、索引の内容（これらは fo:page-sequence）の順で生成する。
- 表紙と目次、索引は、入力の XML データの順に沿ったスタイルシートからは作れないので、独自に作成するサブルーチンのテンプレートを作る。
- これらの制御をルート要素 doc を処理するテンプレートで行う。

doc 要素を処理するテンプレートは以下のようになります。要件どおり fo:layout-master-set の出力、表紙の作成、目次の作成、本文の処理、索引の作成という順になっています。doc 要素のプロパティまたは外部パラメータを指定することで表紙、目次、索引の出力をそれぞれ抑制できるようにしました。例えば、<doc cover="false">とすれば表紙は出力されません。また、外部パラメータとして toc-make の値を false としておけば目次は出力されません。

doc 要素を処理するテンプレート

```
<xsl:template match="doc">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <!-- ページ書式の設定（fo:simple-page-master） →省略 -->
    </fo:layout-master-set>

    <!-- head 要素を処理させ表紙を作成します。 -->
    <xsl:if test="$cover-make or @cover!='false'">
      <xsl:apply-templates select="head" />
    </xsl:if>

    <!-- 目次を作成するテンプレートを呼び出します。 -->
    <xsl:if test="$toc-make or @toc!='false'">
      <xsl:call-template name="toc" />
    </xsl:if>

    <!-- 本文（body 要素以降）を処理します。 -->
    <xsl:apply-templates select="body" />

    <!-- 索引を作成するテンプレートを呼び出します。 -->
    <xsl:if test="$index-make or @index!='false'">
      <xsl:if test="//index">
        <xsl:call-template name="index.create" />
      </xsl:if>
    </xsl:if>
  </fo:root>
</xsl:template>
```

表紙の作成

- 表紙には head 要素の title（表題）、date（作成日）、author（著者）を出力する。abstract は出力しない。
- 表題を格納するブロックは幅 130mm、高さ 20mm で左右中央に配置する。背景はグレーでボーダーにはそれより濃い目のグレーを使用する。上マージンから 25mm の位置に配置し、次に配置する作成日との間に 150mm の距離を確保する。フォントサイズは 24pt とし、フォントは「ＭＳ ゴシック」を使用する。文字の配置はブロック内でセンタリングする。
- 作成日を格納するブロックは、幅 160mm で左右中央に配置する。背景色、ボーダーはなし。フォントサイズは 14pt、フォントは「ＭＳ 明朝」を使用する。著者との間に 5mm の空きを確保する。
- 著者を格納するブロックは、幅 160mm で左右中央に配置する。背景色、ボーダーはなし。フォントサイズは 14pt、フォントは「ＭＳ 明朝」を使用する。author にロゴマークの画像が指定された場合はそれを文字の上側に印字する。

表紙の作成は head を処理するテンプレートで行います。

表題部分のレイアウト指定は xsl:attribute-set の中で name="cover.title"の部分に整理されています。

表紙の表題・作成日・著者の書式指定

```
<!-- cover -->
<xsl:attribute-set name="cover.title">
  <xsl:attribute name="space-before">25mm</xsl:attribute>
  <xsl:attribute name="space-before.conditionality">retain</xsl:attribute>
  <xsl:attribute name="space-after">150mm</xsl:attribute>
  <xsl:attribute name="font-size">24pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ ゴシック"</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="text-align-last">center</xsl:attribute>
  <xsl:attribute name="start-indent">15mm</xsl:attribute>
  <xsl:attribute name="width">130mm</xsl:attribute>
  <xsl:attribute name="height">20mm</xsl:attribute>
  <xsl:attribute name="background-color">#EEEEEE</xsl:attribute>
  <xsl:attribute name="border-style">outset</xsl:attribute>
  <xsl:attribute name="border-color">#888888</xsl:attribute>
  <xsl:attribute name="padding-top">5pt</xsl:attribute>
  <xsl:attribute name="padding-bottom">5pt</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="cover.date">
  <xsl:attribute name="space-after">5mm</xsl:attribute>
  <xsl:attribute name="font-size">14pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ 明朝"</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="text-align-last">center</xsl:attribute>
  <xsl:attribute name="width">160mm</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="cover.author">
  <xsl:attribute name="font-size">14pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ 明朝"</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="text-align-last">center</xsl:attribute>
  <xsl:attribute name="width">160mm</xsl:attribute>
</xsl:attribute-set>
```

注意すべき点は次の通りです。

- 表題をレイアウトする手段として fo:block-container を使用します。fo:block-container には width, height が指定できます。本文領域の幅は 210mm - 25mm - 25mm = 160mm です。この幅の中にセンタリングして配置できればよいのですが、そのような機能はないので、ここからブロックの幅 130mm を引き結果の 30mm を等分して、start-indent=15mm と指定しています。
- fo:block-container にはプロパティ space-before="25mm"を指定しています。この表題は本文領域内の最初のブロックになります。しかし、既定値のままでは、本文領域の先頭ブロックの space-before は無視されて、表題が本文領域の上端に配置されてしまいます。space-before.conditionality="retain" とすることにより、強制的に空きを確保することができます。

author に logo プロパティが指定されていた場合、それを画像として表示します。これを処理するのが author. logo. img テンプレートです。pos プロパティにより、配置位置も著者の左か上かを選択できます。画像付き著者名の例は本稿の表紙をご覧ください。

head 要素を処理するテンプレート

```
<xsl:template match="doc/head">
  <fo:page-sequence master-reference="PageMaster-Cover">
    <fo:flow flow-name="xsl-region-body">
      <fo:block-container xsl:use-attribute-sets="cover.title">
        <xsl:apply-templates select="/doc/head/title" />
      </fo:block-container>
      <fo:block-container xsl:use-attribute-sets="cover.date">
        <xsl:apply-templates select="/doc/head/date" />
      </fo:block-container>
      <fo:block-container xsl:use-attribute-sets="cover.author">
        <xsl:apply-templates select="/doc/head/author" />
      </fo:block-container>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>

<xsl:template match="doc/head/title">
  <fo:block start-indent="0mm">
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="doc/head/date">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="doc/head/author">
  <fo:block>
    <xsl:if test="@logo">
      <xsl:call-template name="author.logo.img" />
    </xsl:if>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template name="author.logo.img">
  <xsl:choose>
    <xsl:when test="@pos='side'">
      <fo:inline space-end="1em">
        <fo:external-graphic src="{@logo}">
          <xsl:if test="@width and @height">
            <xsl:attribute name="content-width">
              <xsl:value-of select="@width" />
            </xsl:attribute>
            <xsl:attribute name="content-height">
```

```
              <xsl:value-of select="@height" />
            </xsl:attribute>
          </xsl:if>
        </fo:external-graphic>
      </fo:inline>
    </xsl:when>
    <xsl:otherwise>
      <fo:block space-after="1em">
        <fo:external-graphic src="{@logo}">
          <xsl:if test="@width and @height">
            <xsl:attribute name="content-width">
              <xsl:value-of select="@width" />
            </xsl:attribute>
            <xsl:attribute name="content-height">
              <xsl:value-of select="@height" />
            </xsl:attribute>
          </xsl:if>
        </fo:external-graphic>
      </fo:block>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

テンプレートはきわめて単純な構造です。title, date, author のそれぞれに対応した fo:block-container に、xsl:attribute-set 要素で定義したプロパティの組を、xsl:use-attribute-sets で呼び出して適用させています。それぞれの fo:block-container の中では、title, date, author の各要素にそれぞれのテンプレートを適用していきます。

目次の作成

- 目次は表紙の次に改ページして配置する。表題は「目次」。背景は薄い灰色。
- 入力 XML 文書中の part（部）, chapter（章）, section（節）, subsection（副節）, subsubsection（副々節）の title 要素の内容を集めて目次を作成する。
- 目次の各行は、part ～ subsection の各 title、リーダ（罫）、ページ番号で構成。
- 目次の各行は文書中の part ～ subsection のネストレベル（入れ子の深さ）に応じて、前スペース、左インデント、フォントサイズ、フォントウェイトを設定する。
- PDF 出力のために目次の各行から本文中の見出しへの内部リンクを設定する。

目次作成テンプレート

目次は toc テンプレートで作成します。toc テンプレートは、ルート要素 doc を処理するテンプレートから、<xsl:call-template name="toc">で呼び出されます。

toc テンプレート

```
<xsl:template name="toc">
  <!-- fo:page-sequence を生成します。-->
  <fo:page-sequence master-reference="PageMaster-TOC">
    <!-- region-body に対する flow を生成します。-->
    <fo:flow flow-name="xsl-region-body">
      <!--目次全体のブロックを生成します。-->
      <fo:block xsl:use-attribute-sets="div.toc">
        <!--目次のタイトル「Table of Contents」を生成します。-->
        <fo:block xsl:use-attribute-sets="h2">Table of Contents</fo:block>
        <!-- XML 文書全体から part, chapter, section, subsection,
          subsubsection 要素を抽出し-->
        <xsl:for-each select="//part |
                              //chapter |
                              //section |
                              //subsection |
                              //subsubsection">
          <!-- 各々に対して目次の各行を生成するテンプレートを適用します。-->
          <xsl:call-template name="toc.line" />
        </xsl:for-each>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

toc テンプレートでは次の順で処理を行います。

1. 新しい page-sequence を生成します。この page-sequence はページ書式として master-name="PageMaster-TOC"の fo:simple-page-master を参照します。新しい page-sequence が生成されるため、印刷時には改ページが発生します。
2. 次に本文領域の xsl:flow オブジェクトを生成します。目次全体を蔽うブロックを、"div.toc"という名の attribute-set を適用して作成します。この attribute-set では、背景の薄い灰色を設定しています。そして目次のタイトルの「目次」を作成します。
3. xsl:for-each select="..."で、文書全体の part ～ subsubsection の要素集合を生成し、個々の要素を目次の一行を処理する toc.line テンプレートに渡します。目次の行の並びは XML 文書ツリーでの該当ノードの出現順になります。

このテンプレートは doc 要素を処理するテンプレートから呼び出されますので、カレントノードは、doc 要素ノードです。xsl:for-each は、このカレントノードを select で指定したノード集合のひとつひとつに一時的に変更します。したがって呼び出される toc.line テンプレートでは、カレントノードは part ～ subsection のいずれかの要素ノードになります。xsl:for-each の処理が終了するとカレントノードは元の doc 要素ノードに復帰します。

目次行の作成テンプレート

toc.line テンプレートでは、目次の一行を編集します。

目次の各行を生成する toc.line テンプレート

```
<!-- 目次行の編集で使用するグローバルパラメータと変数です。 -->
<xsl:param name="toc-level-default" select="3" />
<!-- 目次行の編集テンプレート本体 -->
<xsl:variable name="toc-level-max">
  <xsl:choose>
    <xsl:when test="not (doc/@toclevel)">
      <xsl:value-of select="$toc-level-default" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="number(doc/@toclevel)" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>

<xsl:template name="toc.line">
  <!-- カレントノードのネストレベルを計算し "level" ローカル変数にセットします。 -->
  <xsl:variable name="level"
    select="count(ancestor-or-self::part | ancestor-or-self::chapter |
    ancestor-or-self::section | ancestor-or-self::subsection |
    ancestor-or-self::subsubsection )" />
  <!-- ネストレベルが対象内かチェックします。 -->
  <xsl:if test="$level &lt;= $toc-level-max">
    <!-- 目次の一行ごとに fo:block を生成します。 -->
    <fo:block text-align-last="justify">
      <!-- 左マージンはネストレベルに比例させて深くします。-->
      <xsl:attribute name="margin-left">
        <xsl:value-of select="$level - 1" />
        <xsl:text>em</xsl:text>
      </xsl:attribute>

      <!-- space-before は上位の項目であるほど大きく取ります。-->
      <xsl:attribute name="space-before">
          <xsl:choose>
          <xsl:when test="$level=1">5pt</xsl:when>
          <xsl:when test="$level=2">3pt</xsl:when>
          <xsl:when test="$level=3">1pt</xsl:when>
          <xsl:otherwise>1pt</xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <!-- font-size も同様です。-->
      <xsl:attribute name="font-size">
        <xsl:choose>
          <xsl:when test="$level=1">1em</xsl:when>
          <xsl:otherwise>0.9em</xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <!-- font-weight も同様です。-->
      <xsl:attribute name="font-weight">
        <xsl:value-of select="800 - $level * 100" />
      </xsl:attribute>
```

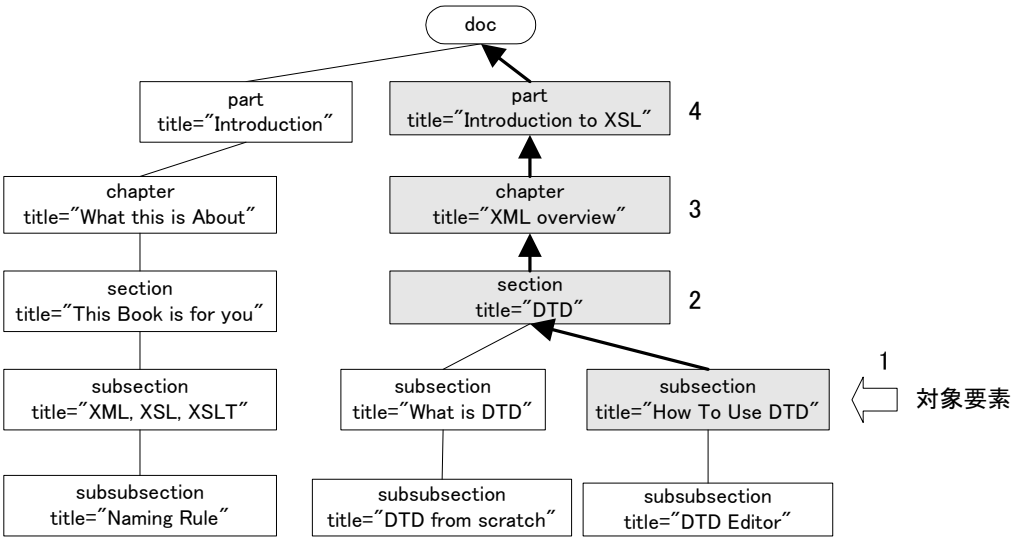
```
      <!-- 以降が目次のデータです。 -->
      <fo:basic-link internal-destination="{generate-id()}">
         <xsl:value-of select="title" />
      </fo:basic-link>
      <fo:leader leader-pattern="dots" />
      <!-- fo:page-number-citation を生成します。印刷時はページ番号で置き換えられます。-->
      <fo:page-number-citation ref-id="{generate-id()}" />
    </fo:block>
  </xsl:if>
</xsl:template>
```

toc.line テンプレートでは次の順で処理を行います。

1. 処理する要素ノードのルート要素からの入れ子の深さ（ネストレベル）を計算し、level 変数に設定します。
2. ネストレベルが「目次を設定するレベル」以下であれば処理を進めます。そうでなければ読み飛ばします。目次を設定するレベルは、doc 要素の toclevel プロパティで指定します。指定がない場合は 3 としています。
3. 目次の各行に対して fo:block を生成します。
4. ルート要素からの深さに応じて、インデント・フォントサイズ・フォントウェイトのプロパティ値を決定します。
5. 目次行の実データであるその要素の title、リーダ、ページ番号を出力します。目次の見出しを fo:basic-link で囲み、見出しから本文へのリンクを設定します。生成した PDF に内部リンクとして設定されます。（fo:basic-link については、「PDF 生成に関する機能」の項で説明します。）

ネストレベルの計算

ネストレベルは、level というローカル変数を設けて、count(ancestor-or-self::part|ancestor-or-self::chapter|ancestor-or-self::section|ancestor-or-self::subsection||ancestor-or-self::subsubsection)という値を計算させることで得ています。doc 要素の下にある自分もしくは先祖の part ～ subsubsection の数を数えるわけです。図解すると次のようになります。



※count(ancestor-or-self::part|ancestor-or-self::chapter|ancestor-or-self::section|ancestor-or-self::subsection)は、
自分自身を含む先祖のpart, chapter, section, subsection, subsubsection要素の数をカウントして返します.
例えば、対象の要素がtitel="How To Use DTD"のsubsectionの場合、count(...)関数の返す値は4になります.

ルート要素からのネストレベルの計算

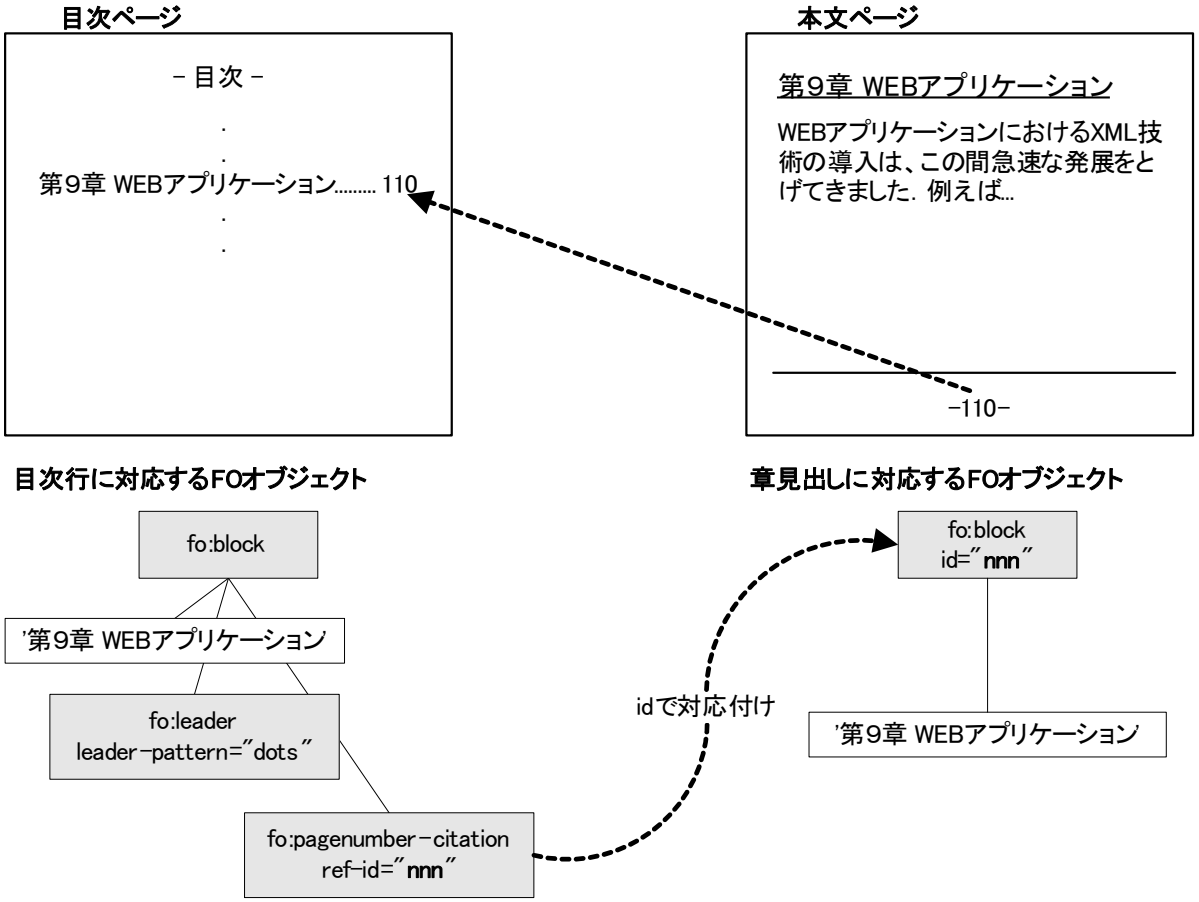
ネストレベルに応じたプロパティ設定

取得したカレントノードのネストレベルに応じて、fo:block のプロパティを設定します。ここでは、part ～ subsubsection という要素名に応じて設定しているのではない点にご注意ください。ネストレベルに応じてプロパティ設定を行うことで、使用する要素に依存せず、同じフォーマットで目次を生成できます。次の表がスタイルシートが設定しているプロパティです。

プロパティ	ネストレベル				
	1	2	3	4	5
margin-left	0em	1em	2em	3em	4em
space-before	5pt	3pt	1pt	1pt	1pt
font-size	1em	0.9em	0.9em	0.9em	0.9em
font-weight	700	600	500	400	300

ページ番号の取得

次に各 part ～ subsection の出現するページ番号を取得する必要があります。ところが、XSL 変換で FO を作る段階ではページ番号の値を知りたくても、まだ**組版していないので値は未確定**です。これを XSL-FO では fo:page-number-citation という機構で解決します。fo:page-number-citation は、組版の過程でフォーマッタがページ番号に置き換える FO です。どのページ番号で置き換えるのかを指定するのが ref-id プロパティです。フォーマッタは ref-id で指定された値と同じ値を id プロパティで持った FO を探します。そしてその FO が属しているページの番号を取得してきてくれます。したがって part ～ subsection 要素から生成する fo:block には、かならず id プロパティを作らねばなりません。この仕組みを図で表すと次のようになります。



fo:page-number-citation の仕組み

テンプレート中では ref-id プロパティの値として generate-id( )関数を使用しています。generate-id( )関数は XSLT プロセッサが、カレントノードを識別するユニークな文字列を生成してくれます。

fo:leader

目次行のタイトルとページ番号の間には、fo:leader を使用しています。fo:leader は、インラインエリアを生成する特殊なオブジェクトです。ここでは leader-pattern="dots"を指定しました。タイトルとページ番号の間を dot（ピリオド）で埋める役割を果たします。

ここで重要な点は、目次行を生成する fo:block で、text-align-last="justify"で両端揃えを指定している点です。これにより、部～節のタイトルは左に、ページは右に配置され、その間をリーダパターンが埋める結果を得ることができます。

fo:leader のプロパティで様々なパターンを指定することができます。次に例を示します。左側が fo:leader のプロパティです。

leader-pattern="dots".. 99 ページ
leader-pattern="rule" rule-style="dotted".. 99 ページ
leader-pattern="rule" rule-style="dashed"- 99 ページ
leader-pattern="rule" rule-style="solid"______________________________ 99 ページ
leader-pattern="rule" rule-style="double"______________________________ 99 ページ
leader-pattern="rule" rule-style="groove"______________________________ 99 ページ
leader-pattern="rule" rule-style="ridge"______________________________ 99 ページ

また次のようにした場合

```
<fo:leader leader-pattern="use-content">+</fo:leader>
```

パターンの任意指定+++ 99 ページ

生成された目次行の例

今までの手続きを経て作成された目次行の FO の例を示します。

生成された目次行

```
<fo:block text-align-last="justify" margin-left="0em"
  space-before="5pt" font-size="1em" font-weight="700">
  <fo:basic-link internal-destination="IDA0UU3B">
    はじめに
  </fo:basic-link>
  <fo:leader leader-pattern="dots" />
  <fo:page-number-citation ref-id="IDA4AIOB" />
</fo:block>
```

実際の印刷例は本稿の目次を参照ください。

 本文の処理

- 入力 XML 文書中の body 要素以下の要素の内容を本文に出力する。
- 本文部分の各ページ書式は、ページヘッダ、ページフッタと本文領域から構成する。ページヘッダ、ページフッタの内容は本文の偶数ページと奇数ページで対称の位置に配置する。
- 脚注があるページには、本文領域と脚注領域を区切る線を引く。

本文を処理するテンプレートの枠組み

入力 XML 文書の本文部分は body 要素以下に格納されます。body 要素を処理するテンプレートを次に示します。

body 要素を処理するテンプレート

```
<!-- 本文の処理 -->
<xsl:template match="body">
<!-- 開始ページ番号は1 -->
  <fo:page-sequence master-reference="PageMaster"
    initial-page-number="1">
    <!-- 左ページのヘッダ -->
    <fo:static-content flow-name="Left-header">
      <!-- （詳細は、「ページヘッダとページフッタの作成」の項を参照）-->
    </fo:static-content>
    <!-- 右ページのヘッダ -->
    <fo:static-content flow-name="Right-header">
      <!-- （詳細は、「ページヘッダとページフッタの作成」の項を参照）-->
    </fo:static-content>

    <!-- 左ページのフッタ -->
    <fo:static-content flow-name="Left-footer">
      <!-- （詳細は、「ページヘッダとページフッタの作成」の項を参照）-->
    </fo:static-content>
    <!-- 右ページのフッタ -->
    <fo:static-content flow-name="Right-footer">
      <!-- （詳細は、「ページヘッダとページフッタの作成」の項を参照）-->
    </fo:static-content>

    <!-- 本文と脚注の間に短い線を引く -->
    <fo:static-content flow-name="xsl-footnote-separator">
      <fo:block>
        <fo:leader leader-pattern="rule" rule-thickness="0.5pt"
          leader-length="33%" />
      </fo:block>
    </fo:static-content>

    <!-- 本文領域 -->
    <fo:flow flow-name="xsl-region-body">
      <fo:block>
        <xsl:apply-templates />
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

このテンプレートでは、以下の処理を行います。

1. 新しい "PageMaster" に基づいた fo:page-sequence を作成します。これで直前の目次からページ書式が切り替わります。
2. 新しいページ書式に基づいて、ページヘッダ、ページフッタの内容を出力します。
3. 本文と脚注の間の境界領域をリーダで作成します。
4. 本文領域のフローオブジェクトを生成します。
5. xsl:apply-templates で下位の要素を処理します。

ページヘッダやページフッタは fo:static-content の中に記述します。本文ページでは、左右でページ書式の切り替えを行うため、左右のページヘッダと左右のページフッタの 4 つの fo:static-content を作成します。また、本文と脚注の境界も fo:static-content を使って作成します。

4 つの fo:static-content をページ内の領域への対応付けは次のようになります。「本文のページ書式 - 左右ページ書式の切り替え」で用意した本文の fo:simple-page-master は右ページ用と左ページ用が定義されていて、各ページのヘッダ領域とフッタ領域にはそれぞれ名前が付けられています。一方、fo:static-content には flow-name をつけ、それが region-name と一致する領域に fo:static-content の内容が流し込まれます。

ページ	領域の名前	static-content の名前
右ページヘッダ	fo:region-before region-name="Right-header"	fo:static-content flow-name="Right-header"
右ページフッタ	fo:region-after region-name="Right-footer"	fo:static-content flow-name="Right-footer"
左ページヘッダ	fo:region-before region-name="Left-header"	fo:static-content flow-name="Left-header"
左ページフッタ	fo:region-after region-name="Left-footer"	fo:static-content flow-name="Left-footer"

脚注と本文との境界線に、xsl-footnote-separator という flow-name を持つ fo:static-content で作成します。線の描画には、fo:leader オブジェクトを使用します。実線で本文領域の 1/3 の幅を確保します。

本文の内容は fo:flow の子供として出力します。

ページ番号の設定

fo:page-sequence に initial-page-number プロパティを使ってページ番号の初期値を設定することが可能です。SD2FO-DOC.xsl では、本文の fo:page-sequence に initial-page-number="1"を設定して、本文から 1 ページが開始するようにしています。

ページフッタ／ページヘッダ内容の作成

- ページフッタにはページ番号とページ中の節タイトルを配置する。
- ページヘッダには文書タイトルを出力する。さらに、入力文書中の part（部）, chapter（章）, section（節）のいずれかのうち、文書内の最上位レベル要素の出現に応じて番号つきの爪を配置する

ページフッタの出力

fo:static-content にページフッタの内容を出力します。左右ページで配置が異なりますが内容は同じです。

- ページ番号
- 本文 section の見出し（柱）

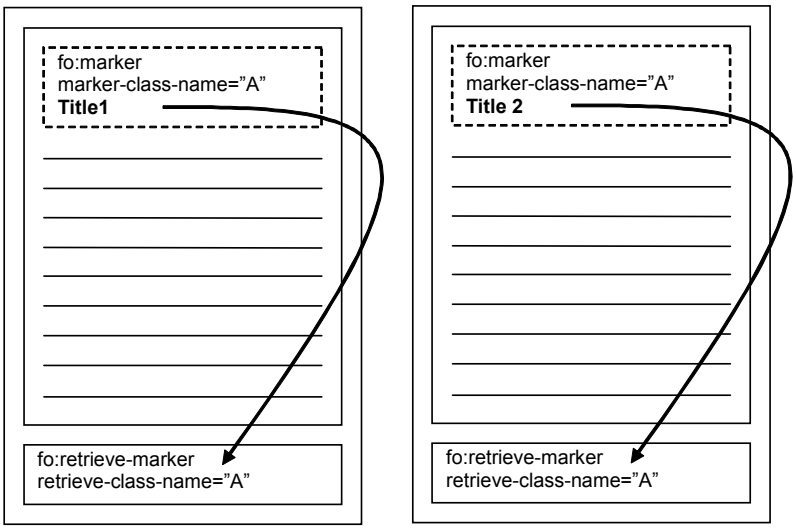
ページ番号の出力

ページ番号を表すには fo:page-number オブジェクトを使用します。fo:page-number オブジェクトは特殊なインラインエリアを生成し、組版時にフォーマッタがページ番号に置換します。ページ番号は小口側に配置するため fo:block のプロパティに text-align="outside"を設定します。

```
<!-- フッタ領域にページ番号を配置する -->
  <fo:block font-size="9pt" text-align="outside">
    <fo:inline font-size="17pt">
      <fo:page-number />
      <xsl:text> | </xsl:text>
    </fo:inline>
   (ランニングフッタの出力（後述))
  </fo:block>
```

ランニングフッタの作成

文書中の section のタイトルをページフッタに出力します。section のタイトルは節ごとに変わっていくのでランニングフッタとなります。このため fo:marker と fo:retrieve-marker を使います。本文中のタイトル要素に対して fo:marker を作成し、ページフッタの fo:static-content には fo:retrieve-marker を置きます。組版時にフォーマッタが fo:retrieve-marker の部分を、該当する fo:marker の内容に置換します。



fo:retrieve-marker の retrieve-class-name プロパティで、置換したい fo:marker のクラス名を指定します。retrieve-boundary は適用範囲を指定します。retrieve-position はそのページの中のどの fo:marker を選択するか（最初に現れるものか、最後のものか等）を指定します。

```
<!-- フッタに section 名を出力するために retrieve-marker を置く -->
<fo:retrieve-marker
  retrieve-boundary="page-sequence"
  retrieve-position="first-starting-within-page"
  retrieve-class-name="section-title" />
```

本文中のタイトル要素に対して、fo:marker を生成します。″part | chapter | section | subsection | subsubsection | appendix ″を処理するテンプレートの中で以下のように記述します。

```
<xsl:if test="local-name() = 'section'">
  <xsl:element name="fo:marker">
    <xsl:attribute name="marker-class-name">section-title</xsl:attribute>
    <xsl:value-of select="title" />
  </xsl:element>
</xsl:if>
```

これによって文書中に section が出現するたび、以下のように fo:marker が生成されます。

```
<fo:flow flow-name="xsl-region-body">
  <fo:marker marker-class-name="section-title">はじめに</fo:marker>
  <!-- セクションの内容 -->
  ....................................
  ....................................
  ....................................
  ....................................
  <fo:marker marker-class-name="section-title">
    XSL-FO 変換のステップ
  </fo:marker>
  <!-- セクションの内容 -->
  ....................................
  ....................................
  ....................................
  ....................................
  <fo:marker marker-class-name="section-title">
    SimpleDoc の構造
  </fo:marker>
  <!-- セクションの内容 -->
```

```
  ....................................
  ....................................
```

ページヘッダの出力

ページヘッダの内容は次の 2 つです。

・文書名
・爪

文書名の出力

文書名をページヘッダに出力する処理は、以下のように文書名要素を fo:block に出力するだけです。

```
<fo:block font-size="7pt" text-align="center"
  border-after-width="thin" border-after-style="solid">
  <xsl:value-of select="/doc/head/title" />
</fo:block>
```

爪の出力

ページヘッダに出力する爪もページフッタの節タイトルと同様に fo:marker と fo:retrieve-marker を使うことで可能です。SD2FO-DOC.xsl スタイルシートでは、15 種類のクラス名（thumb1, thumb2, ..., thumb14, thumb0）を用意し、文書内に出現する対象（part/chapter/section のうち最上位レベル）へ順に設定していきます。ページヘッダである fo:static-content には、それぞれの fo:retrieve-marker を表セルを使って設定します。ページ内に該当する fo:marker が存在しなければ、fo:retrieve-marker は置換されません。これにより、爪が節の切り替えに応じて移動しているように見せることができます。

スタイルシートのページヘッダの表組は以下のように生成されています。

爪を出力するための表の生成

```
<!-- 絶対位置指定のテーブル -->
<fo:block-container absolute-position="fixed"
  top="0mm" left="20mm" height="15mm">
  <fo:table>
    <fo:table-column column-width="12mm" number-columns-repeated="15" />
    <fo:table-body>
      <fo:table-row>
        <fo:table-cell>
          <fo:block font-size="24pt"
            color="white" background-color="black"
            text-align-last="center" display-align="center">
            <!-- thumb1 のクラスを持つ marker があれば置換される -->
            <fo:retrieve-marker
              xsl:use-attribute-sets="thumb-class"
              retrieve-class-name="thumb1" />
          </fo:block>
        </fo:table-cell>
        <!-- 同じように残り１４個のセルを作る（最後は retrieve-class-name="thumb0"）。-->
      </fo:table-row>
    </fo:table-body>
  </fo:table>
</fo:block-container>
```

本文中の"part | chapter | section | …"の処理で fo:marker の生成は以下のようになっています。変数 thumb($thumb)には、爪を切り替える要素名が設定されています。複数の要素を共通で処理するテンプレートなので、カレントノードを判定しなければなりません。カレントノードの要素名は、local-name( )を使って調べてられています。

爪のための fo:marker の生成

```
<xsl:template
  match="part | chapter | section | subsection | subsubsection | appendix">
  <fo:block>
    <xsl:choose>
      <xsl:when test="(local-name() = 'part')
        and ($thumb = 'part')">
        <xsl:element name="fo:marker">
          <xsl:variable name="num">
            <xsl:number format="1" />
          </xsl:variable>
          <xsl:attribute name="marker-class-name">
            thumb<xsl:value-of select="$num mod 15" />
          </xsl:attribute>
          <xsl:number format="1" />
        </xsl:element>
      </xsl:when>
      <xsl:when test="(local-name() = 'chapter')
        and ($thumb = 'chapter')">
        <xsl:element name="fo:marker">
          <xsl:variable name="num">
            <xsl:number format="1" />
          </xsl:variable>
          <xsl:attribute name="marker-class-name">
            thumb<xsl:value-of select="$num mod 15" />
          </xsl:attribute>
          <xsl:number format="1" />
        </xsl:element>
      </xsl:when>
      <xsl:when test="(local-name() = 'section')
        and ($thumb = 'section')">
        <xsl:element name="fo:marker">
          <xsl:variable name="num">
            <xsl:number format="1" />
          </xsl:variable>
          <xsl:attribute name="marker-class-name">
            thumb<xsl:value-of select="$num mod 15" />
          </xsl:attribute>
          <xsl:number format="1" />
        </xsl:element>
      </xsl:when>
    </xsl:choose>
    <!-- 以降省略 -->
```

設定するクラス名（marker-class-name）は、'thumb'までは固定文字列で、その後は順に数字をつけます。1 から始まって 15 個単位で繰り返すために、最初に xsl:number を使ってその要素が文書内の同じ要素と比較して何番目であるかを得ます。次にその数値を 15 で割った余りをクラス名として用いることによって実現しています。

見出しの作成

- 入力 XML 文書中の part（部）, chapter（章）, section（節）, subsection（副節）, subsubsection（副々節）の title から見出しを作成する。
- 見出しの書式は、part 〜 subsection の要素ごとに割り当てるのではなく、文書中のネストレベルに応じて割り当てる。
- 最上位のネストレベルの見出しは直前で改ページする。
- 見出しに画像を配置できるようにする。

見出しの書式条件

一般的には部〜副節の見出しに設定する書式は、その要素にあわせて作成しますが、ここではネストレベルに応じて設定するものとします。設定する条件は次のとおりです。

ネストレベル	attribute-set	書式の条件
1	h1	フォント：サイズ 24pt, ＭＳ ゴシック, ボールド 後スペース：14pt, 次のブロックとのページ内の継続：always ボトムボーダー：実線 2pt　ブレーク条件：直前で改ページ
2	h2	フォント：サイズ 16pt, ＭＳ ゴシック, ボールド 前スペース：19pt　後スペース：5pt　次のブロックとの継続：always
3	h3	フォント：サイズ 13pt, ＭＳ ゴシック, ボールド 前スペース：14pt　後スペース：5pt　次のブロックとの継続：always
4	h4	フォント：サイズ 12pt, ＭＳ ゴシック, ボールド 前スペース：5pt　後スペース：5pt　次のブロックとの継続：always
5	h5	フォント：サイズ 10pt, ＭＳ ゴシック, ボールド 前スペース：3pt　後スペース：3pt　次のブロックとの継続：always

この条件を記述したスタイルシートの書式定義部は以下のとおりです。

見出しの書式定義

```
<!-- titles -->
<xsl:attribute-set name="h1">
  <xsl:attribute name="font-size">24pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ ゴシック"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-after">14pt</xsl:attribute>
  <xsl:attribute name="break-before">page</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
  <xsl:attribute name="border-after-style">solid</xsl:attribute>
  <xsl:attribute name="border-after-width">2pt</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h2">
  <xsl:attribute name="font-size">16pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ ゴシック"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">19pt</xsl:attribute>
  <xsl:attribute name="space-after">5pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h3">
  <xsl:attribute name="font-size">13pt</xsl:attribute>
```

```
  <xsl:attribute name="font-family">"ＭＳ ゴシック"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">14pt</xsl:attribute>
  <xsl:attribute name="space-after">5pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h4">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ ゴシック"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">5pt</xsl:attribute>
  <xsl:attribute name="space-after">5pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h5">
  <xsl:attribute name="font-size">10pt</xsl:attribute>
  <xsl:attribute name="font-family">"ＭＳ ゴシック"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="space-after">3pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>
```

見出しは、次のブロックとの間で自然改ページが発生しないように、keep-with-next.within-page="always"を指定します。h1 には、keep-with-next.within-page="always"は指定してありませんが、break-before="page"が指定されているので、直前に改ページが挿入され、ブロックは必ずページの先頭に配置されます。したがって次のブロックと連続します。

見出しを処理するテンプレート

見出しを処理するテンプレートを次に示します。ネストレベルで書式を選択するため、ひとつのテンプレートで集中的に処理できます。

見出しを処理するテンプレート

```
<xsl:template match="part |
                     chapter |
                     section |
                     subsection |
                     subsubsection">
  <xsl:call-template name="title.out" />
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="part/title |
                     chapter/title |
                     section/title |
                     subsection/title |
                     subsubsection/title">
</xsl:template>

<xsl:template name="title.out">
  <xsl:variable name="level" select="count(ancestor-or-self::part |
                                           ancestor-or-self::chapter |
                                           ancestor-or-self::section |
                                           ancestor-or-self::subsection |
                                           ancestor-or-self::subsubsection )" />
  <xsl:choose>
    <xsl:when test="$level=1">
      <fo:block xsl:use-attribute-sets="h1" id="{generate-id()}">
        <xsl:call-template name="title.out.sub" />
```

```
        <xsl:value-of select="title" />
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=2">
      <fo:block xsl:use-attribute-sets="h2" id="{generate-id()}">
        <xsl:call-template name="title.out.sub" />
        <xsl:value-of select="title" />
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=3">
      <fo:block xsl:use-attribute-sets="h3" id="{generate-id()}">
        <xsl:call-template name="title.out.sub" />
        <xsl:value-of select="title" />
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=4">
      <fo:block xsl:use-attribute-sets="h4" id="{generate-id()}">
        <xsl:call-template name="title.out.sub" />
        <xsl:value-of select="title" />
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=5">
      <fo:block xsl:use-attribute-sets="h5" id="{generate-id()}">
        <xsl:call-template name="title.out.sub" />
        <xsl:value-of select="title" />
      </fo:block>
    </xsl:when>
    <xsl:otherwise>
      <fo:block xsl:use-attribute-sets="h5" id="{generate-id()}">
        <xsl:call-template name="title.out.sub" />
        <xsl:value-of select="title" />
      </fo:block>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="title.out.sub">
  <xsl:if test="@logo">
    <fo:inline space-end="5pt">
      <fo:external-graphic src="{@logo}">
        <xsl:if test="@width and @height">
          <xsl:attribute name="content-width">
            <xsl:value-of select="@width" />
          </xsl:attribute>
          <xsl:attribute name="content-height">
            <xsl:value-of select="@height" />
          </xsl:attribute>
        </xsl:if>
      </fo:external-graphic>
    </fo:inline>
  </xsl:if>
</xsl:template>
```

見出しの処理は４つのテンプレートから構成されます。実際の見出し行の生成は title.out テンプレートを呼び出して行います。title.out では、

1. level というローカル変数に、処理中の要素のネストレベルを計算して格納します。
2. この値に応じて h1 ～ h5 の見出し書式を選択して見出しの fo:block に適用します。
3. 同様に id プロパティを generate-id( )関数で生成して見出しの fo:block に適用します(7)。
4. 画像を処理する title.out.sub テンプレートを呼び出す。

(7) 目次行の fo:page-number-citation から参照します。

5. title 要素で指定された文字列を出力する。

という順で処理されます。

２番目の part/title ～ subsubsection/title を処理するテンプレートは「空の処理」（何も出力されない）になっていますが、これはタイトルを二重に出力してしまわないための常套手段です。タイトル行は先に見たように、title.out の処理で出力されますが、part ～ subsection 要素の xsl:apply-templates を処理しているときに title 要素が現れますから、再びタイトル文字列が出力されてしまいます。しかし、part/title ～ subsubsection/title にマッチするテンプレートを用意して中身を空にしておけば、余計なタイトル文字列の出力が抑制できるというわけです。 これは title.out テンプレートで先行して処理してしまっているので、1 番目のテンプレートで xsl:apply-templates で再び title が処理対象になったときに、title のテキストが出力に現れないようにするためです。

生成された見出しの例

今までの手続きを経て作成された見出しの FO の例を示します。id プロパティに generate-id( )関数の結果が格納されていることにご注目ください。

生成された見出し

```
<fo:block font-size="24pt"
          font-family="&amp;quot;ＭＳ ゴシック&amp;quot;"
          font-weight="bold"
          space-after="14pt"
          break-before="page"
          keep-with-next.within-page="always"
          border-after-style="solid"
          border-after-width="2pt"
          id="IDA4AIOB">
  <fo:inline space-end="5pt">
    <fo:external-graphic src="img/ico4.png" />
  </fo:inline>
  はじめに
</fo:block>
```

インライン要素の処理

- b（太字）、i（斜体）、em（強調）、code（インラインのプログラムコード）要素は、文字のプロパティを設定した fo:inline オブジェクトに変換する。
- a（アンカー）は、テキストを出力するのみとし、リンク先（href プロパティ）の内容を続けて出力する。
- note（注釈）は脚注に変換する。脚注参照文字列は「(*n*)」のフォーマットとし、*n* には文書全体で一意な通し番号を割り当てる。
- br（改行）は、見かけ上行を切り替える。
- span（汎用インライン要素）は、fo:inline を生成するのみとする。

b, i, em, code 要素を処理するテンプレート

b（太字），i（斜体），em（強調），code（インラインのプログラムコード）要素の変換はきわめて簡単です。fo:inline を生成して、そこに該当するプロパティを設定するだけです。太字は font-weight="bold"、斜体は font-style="italic"となります。em も太字を適用します。記述方法が b と違いますが同じ結果です。code は等幅フォントを表す monospace を font-family プロパティに設定します（実際には、Courier などのフォントで組まれます）。

b, i, em, code 要素を処理するテンプレート

```
<xsl:template match="b">
  <fo:inline font-weight="bold">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="i">
  <fo:inline font-style="italic">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="em">
  <fo:inline xsl:use-attribute-sets="em">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="code">
  <fo:inline font-family="monospace">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
```

fo:inline を適用するので、要素テキストの終端は、改行があるとはみなされません。例を次に示します。

インライン要素 i（italic）は「*斜体*」になります。
インライン要素 b（bold）は「**ボールド書体**」になります。
インライン要素 em（emphasis）もやはり「**ボールド書体**」になります。

インライン要素 code（インラインプログラムコード）は「等幅フォント（monospace font)」になります。

a 要素

a（アンカー）は、href プロパティでリンク先 URI を持っています。組版でリンク先をどう扱うかが問題になります。ここでは、リンク先を a 要素で囲まれたテキストの後に括弧で囲んでリンク先を出力することにします。ただし、両者が同じ内容の場合は出力しません。

a 要素を処理するテンプレート

```
<xsl:template match="a">
  <xsl:variable name="anchor-texts">
    <xsl:value-of select="." />
  </xsl:variable>
  <xsl:apply-templates />
  <xsl:if test="@href!=$anchor-texts">
    <fo:inline>
      <xsl:text>(</xsl:text>
      <xsl:value-of select="@href" />
      <xsl:text>)</xsl:text>
    </fo:inline>
  </xsl:if>
</xsl:template>
```

このスタイルシートを使った例を示します。

「この例はW3C のサイトで公開されています。」は
「この例は W3C のサイト(http://www.w3.org/)で公開されています。」と表示されます。

「この例はhttp://www.w3.org/で公開されています。」は
「この例は http://www.w3.org/で公開されています。」と表示されます。

「この例はhttp://www.w3.org/で公開されています。」は
「この例は http://***www.w3.org***/で公開されています。」と表示されます。

ここではアンカーを単純なテキストに変換しましたが、SD2FO-DOC.xsl では XSL の fo:basic-link に変換し、XSL Formatter で組版することで PDF のリンクを作成することができます。詳細は「PDF 生成に関する機能」の「リンクの設定」を参照してください。

note 要素

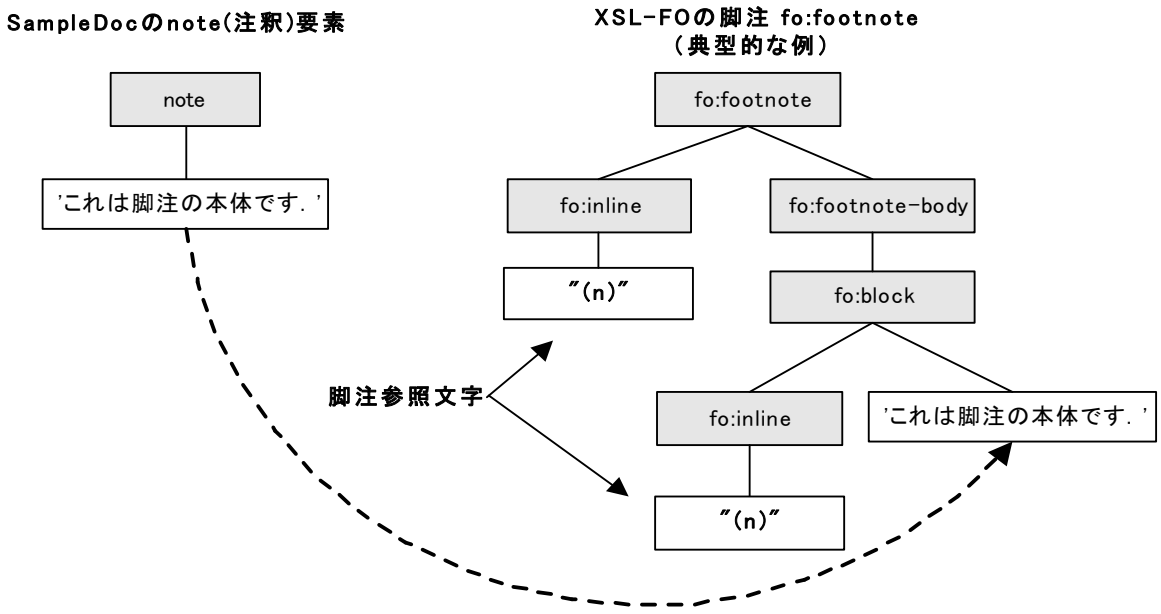
note（注釈）は、XSL-FO の脚注用オブジェクト fo:footnote に変換します。

note 要素を処理するテンプレート

```
<xsl:template match="note">
  <fo:footnote>
    <fo:inline baseline-shift="super" font-size="75%">
      <xsl:number level="any" count="//note" format="(1)" />
    </fo:inline>
    <fo:footnote-body>
      <fo:block xsl:use-attribute-sets="note">
        <fo:inline baseline-shift="super" font-size="75%">
          <xsl:number level="any" count="//note" format="(1)" />
        </fo:inline>
        <xsl:apply-templates />
      </fo:block>
```

```
    </fo:footnote-body>
  </fo:footnote>
</xsl:template>
```

XSL-FO で脚注とその参照ラベルを作成するのが fo:footnote オブジェクトです。構造は、(inline, fo:footnote-body)となっています。最初の子の inline が本文中に配置される脚注参照文字となります。次の fo:footnote-body が脚注の本体で、これは fo:block などのブロック要素から構成されます。典型的な fo:footnote オブジェクトの例を以下に示します。



note 要素と fo:footnote の典型的な例

一般的には本文側に配置する脚注参照文字と同じラベルを脚注の本体側にも置きます。また脚注参照文字は文書中で全体にわたる通し番号をふります。これらはスタイルシート側で行う必要があります。FO には実現する機構はありません。スタイルシートの処理は次のようになります。

1. fo:footnote を出力します。
2. fo:inline で脚注参照文字を xsl:number を使用して作成します。
3. note 要素の子は、同じ脚注参照文字を加えて fo:footnote 要素で囲んだ block 要素に格納します。

xsl:number は、XSLT の命令です。<xsl:number level="any" count="//note" format="(1)" />で、ルート要素の下のすべての note 要素を出現順に検索し現在の note 要素と一致するものを探します。一致したノードの出現番号を"(1)"の書式にしたがってフォーマットします。

また baseline-shift="super"は、上付き文字用に、フォントサイズに合わせてベースラインをシフトさせる指定です。以下に脚注の例を示します。

「脚注の例です。ここに脚注番号が置かれます<note>脚注の本文です。本文領域下端に本文とは離れて配置されます。</note>。」は、
「脚注の例です。ここに脚注番号が置かれます[8]。 」と表示されます。

[8] 脚注の本文です。本文領域下端に本文とは離れて配置されます。

br 要素

br 要素は空要素ですから、空のブロック要素（fo:block）に置き換えてやります。こうすることで、印刷結果では改行が発生します。

br 要素を処理するテンプレート

```
<xsl:template match="br">
  <fo:block>
  </fo:block>
</xsl:template>
```

次に例を示します

「 段落中に br により強制的に
改行を入れてみます。段落が終了するわけではないので、改行前に設定されたプロパティは保持されます。 」は、
「段落中に br により***強制的に***
改行を入れてみます。段落が終了するわけではないので、改行前に設定されたプロパティは保持されます。 」と表示されます。

span 要素

span 要素（汎用インライン要素）は、現在のところ fo:inline に変換するのみです。class プロパティに意味をもたせれば拡張できます。

span 要素を処理するテンプレート

```
<xsl:template match="span">
  <fo:inline>
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
```

ブロック要素の処理

ここでは、表（table）、リスト（ol，ul，dl）以外のブロック要素を扱います。

- p（段落）は、ブロック要素に変換する。段落の一行目は一文字分だけインデントする。体裁は両端揃え。ただし最終行は左揃え。前後に文字サイズ×0.6 のスペースを確保する。段落はページにまたがらないようにする。
- figure（図）は、出現位置で改行してセンタリングして画像を配置する。画像のサイズ指定があった場合はそれを適用する。キャプションがあった場合は画像の下に配置する。
- program（プログラムコード）は、ブロック要素に変換する。等幅フォントを使用し、改行やスペースなどをそのまま無視しないで扱う。背景は灰色。キャプションをプログラムコードの前に出力する。
- div（汎用ブロック要素）は、ブロック要素の fo:block に変換する。

p 要素

p 要素（段落）は、使用頻度が高い要素です。p 要素を処理するテンプレートを次に示します。

p 要素を処理するテンプレート

```
<xsl:attribute-set name="p">
  <xsl:attribute name="text-indent">1em</xsl:attribute>
  <xsl:attribute name="space-before">0.6em</xsl:attribute>
  <xsl:attribute name="space-after">0.6em</xsl:attribute>
  <xsl:attribute name="text-align">justify</xsl:attribute>
  <xsl:attribute name="keep-together.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="p">
  <fo:block xsl:use-attribute-sets="p">
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

テンプレートは簡単で、fo:block に変換するのみです。先頭行のインデントは text-indent で、行揃えは text-align で指定します。また段落がページにまたがらないようにするのは keep-together で指定しています。次に組版した例を示します。

1 行を 21 文字とした場合

　これは段落のサンプルです。段落は fo:block に変換されます。text-indent="1em"を指定するので、先頭行は一文字分だけインデントされます。行揃えは「両端揃え」としています。(text-align="justify") ただし段落の最終行は自動的に左揃えとなります。これは最終行に適用される text-indent-last の初期値が text-align="justify"の場合は、自動的に start になってくれるためです。

1 行を 25 文字とした場合

これは段落のサンプルです。段落は fo:block に変換されます。text-indent="1em"を指定するので、先頭行は一文字分だけインデントされます。行揃えは「両端揃え」としています。(text-align="justify") ただし段落の最終行は自動的に左揃えとなります。これは最終行に適用される text-indent-last の初期値が text-align="justify"の場合は、自動的に start になってくれるためです。

figure 要素

figure 要素（図）は、fo:external-graphic に変換します。figure 要素を処理するテンプレートを次に示します。

figure 要素を処理するテンプレート

```
<xsl:attribute-set name="figure.title">
  <xsl:attribute name="font-family">sans-serif</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="space-after">10pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-previous.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="figure">
  <fo:block text-align="center">
    <fo:external-graphic src="{@src}">
      <xsl:if test="@width and @height">
        <xsl:attribute name="content-width">
          <xsl:value-of select="@width" />
        </xsl:attribute>
        <xsl:attribute name="content-height">
          <xsl:value-of select="@height" />
        </xsl:attribute>
      </xsl:if>
    </fo:external-graphic>
  </fo:block>
  <fo:block xsl:use-attribute-sets="figure.title">
    <xsl:value-of select="title" />
  </fo:block>
</xsl:template>
```

テンプレートでは fo:external-graphic を生成し、画像のパスを src プロパティから設定します。width, height プロパティで画像のサイズ指定があった場合は、content-width, content-height に引き写します。弊社の XSL-Formatter では、BMP, EMF, WMF, JPEG, TIFF, GIF, PNG, EPS, SVG などの画像形式を扱うことができます。

program 要素

program 要素（プログラムコード）は、fo:block に変換し等幅フォントで表示させます。テンプレートを次に示します。

program 要素を処理するテンプレート

```
<xsl:attribute-set name="program">
  <xsl:attribute name="white-space">pre</xsl:attribute>
  <xsl:attribute name="wrap-option">wrap</xsl:attribute>
  <xsl:attribute name="background-color">gainsboro</xsl:attribute>
```

```
  <xsl:attribute name="font-family">monospace</xsl:attribute>
  <xsl:attribute name="font-size">9pt</xsl:attribute>
  <xsl:attribute name="padding">0.5em</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="program.title">
  <xsl:attribute name="font-family">sans-serif</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
  <xsl:attribute name="space-before">0.5em</xsl:attribute>
  <xsl:attribute name="space-after">0.5em</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="program">
  <xsl:apply-templates select="title" />
  <fo:block xsl:use-attribute-sets="program">
    <xsl:apply-templates select="text()" />
  </fo:block>
</xsl:template>
```

テンプレートで fo:block を生成している点は p 要素と変わりません。違いは下位の処理対象を text( )としている点です。これにより program 要素のテキストノードのみが処理対象となります。テキストノードを処理するテンプレート（match="program/text()"）は準備されていません。このため XSLT のビルトインテンプレートで処理されます。ビルトインテンプレートは、テキストを結果に複写します。fo:block に適用するプロパティでは次のものが重要です。

- font-family で等幅の monospace を指定している。
- white-space を pre としています。これは次の 4 つの意味を持ちます。
  1. linefeed-treatment="preserve"：復帰文字（#xA）をそのままとする。空白へ置換したり、無視したりしない。
  2. white-space-treatment="preserve"：XML 仕様で white space に分類される文字で#xA を除くものをそのままとして扱う(9)。
  3. white-space-collapse="false"：linefeed-treatment, space-treatment で処理された後の連続した white space をそのままとして扱う。無視しない。
  4. wrap-option="no-wrap"：行が印字領域からあふれても自動的に折り返さない。
- wrap-option を wrap とし、行があふれた場合には行を折り返す(10)。

これらの指定により program 要素中のテキストは整形済みの扱いとしてフォーマットされます。

div 要素

div 要素（汎用ブロック要素）は、プロパティ指定なしで単純に fo:block に変換しているのみです。テンプレートを次に示します。

div 要素を処理するテンプレート

```
<xsl:template match="div">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

div 要素の応用例を次に示します。div 要素中に FO を直接格納し出力します。

(9) XML 仕様の white space は、空白（#x20）, タブ（#x9）, 改行文字（#xD）, 復帰文字（#xA）です。このうち、#xA は linefeed-treatment で扱いが決まります。

(10) wrap-option="wrap"は既定値ですが、white-space="pre"の指定により、wrap-option="no-wrap"になってしまいます。これをオーバライドしています。

div 要素の応用例

```
<xsl:attribute-set name="div.fo">
  <xsl:attribute name="border">solid</xsl:attribute>
  <xsl:attribute name="border-width">thin</xsl:attribute>
  <xsl:attribute name="padding">1em</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="div[@class = 'fo']">
  <fo:block xsl:use-attribute-sets="div.fo">
    <xsl:copy-of select="node()" />
  </fo:block>
</xsl:template>
```

テンプレートでは、<xsl:copy-of select="node()" />により、div 要素の下の FO の要素をすべてまるごと出力側にコピーします。この機能を使用するために、doc 要素に FO ネームスペースを指定します。

```
<doc xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

使用例を以下に示します。

```
<div class="fo"><fo:block>これは文書中に<fo:inline font-size="1.5em" text-decoration="underline" font-style="italic" font-weight="bold">FO（フォーマッティングオブジェクト）</fo:inline>を直接埋め込む<fo:inline font-weight="bold">例</fo:inline>です。<fo:inline font-size="1em">ス</fo:inline><fo:inline font-size="1.2em">タ</fo:inline><fo:inline font-size="1.4em">イ</fo:inline><fo:inline font-size="1.6em">ル</fo:inline><fo:inline font-size="1.8em">シ</fo:inline><fo:inline font-size="2.0em">ー</fo:inline><fo:inline font-size="2.2em">ト</fo:inline>に制約されずに<fo:inline background-color="#DDDDDD">勝手なことができます。</fo:inline>ただし FO を直接記述するのはとても<fo:inline font-size="3em" font-weight="bold" font-family="sans-serif">大変</fo:inline>です。</fo:block></div>
```

これは次のように表示されます。

これは文書中に ***FO（フォーマッティングオブジェクト）*** を直接埋め込む**例**です。スタイルシートに制約されずに勝手なことができます。ただし FO を直接記述するのはとても**大変**です。

表要素の処理

- 入力 XML 文書中の table 要素以下は表に整形する。
- 表の見出し行は背景色を灰色にして、表の本体データ部分と区別する。また表の罫線は実線で幅は 1pt とする。
- 表のセルは、左に 0.3×フォントサイズ、右に 0.2×フォントサイズのパディングを確保してセルデータを格納する。
- その他表の書式に関するプロパティを処理し、組版結果に反映させる。
  1. table 要素の layout プロパティ、width プロパティ、rowheight プロパティ
  2. col 要素の number プロパティ、width プロパティ
  3. tr 要素の height プロパティ
  4. th, td 要素の align プロパティ、valign プロパティ、colspan プロパティ、rowspan プロパティ

表構造の比較

table 要素以下を表として整形するためには、XSL-FO の fo:table-and-caption に変換してやる必要があります。ここで SimpleDoc と XSL-FO の表の構造を比較してみましょう。まず SimpleDoc の表は次のとおりです。

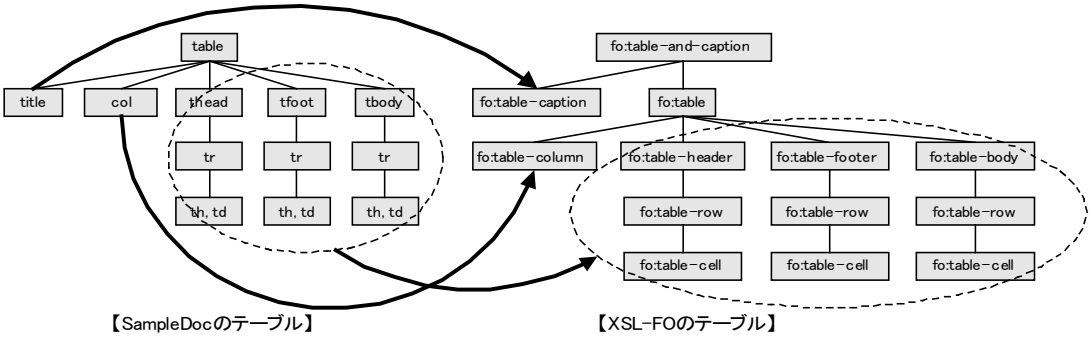
要素	意味	定義
table	表全体	(title?, col*, thead?, tfoot?, tbody) layout プロパティで表を自動レイアウトする（auto）か、レイアウト指定とするか（fixed）を指定します。width プロパティで表全体の幅を指定します。rowheight プロパティで表全体にわたる行の高さを指定します。
col	列プロパティ	EMPTY number プロパティで列番号、width プロパティで列幅を指定します。
thead	表ヘッダ	(tr*)
tfoot	表フッタ	(tr*)
tbody	表の本体（ヘッダ、フッタを除く）	(tr*)
tr	表の行	(th | td)* height プロパティで行の高さを指定できます。
th	ヘッダセル	(inline 要素並び)* colspan プロパティで横結合する列数, rowspan プロパティで縦結合する行数を指定できます。align, valign プロパティで横、縦方向の揃えを指定できます。
td	データセル	(inline 要素並び)* th と同じ。

これに対して XSL-FO の表のオブジェクトは次のような構成になっています。

要素	意味	定義
fo:table-and-caption	表とその表題の全体	(table-caption?, table)
fo:table-caption	表の表題	(%block;)[11]

[11] %block;は XSL 仕様では block, block-container, table-and-caption, table, list-block を含むものと解説されています。常識的には fo:block と fo:block-container です。

要素	意味	定義
fo:table	表自体（表題を除く）	(fo:table-column*, fo:table-header?, fo:table-footer?, fo:table-body+) table-layout：表の自動レイアウト指定(auto/fixed) table-omit-header-at-break：ページ分割時のヘッダ行有無 table-omit-footer-at-break：ページ分割時のフッタ行有無
fo:table-column	表の縦列の特性を定義	EMPTY column-number：列番号(12) column-width：列幅 number-columns-repeated：同じ設定を適用する列数 number-columns-spanned：結合する列数(13)
fo:table-header	表のヘッダ	(fo:table-row+|fo:table-cell+)
fo:table-footer	表のフッタ	(fo:table-row+|fo:table-cell+)
fo:table-body	表の本体（ヘッダ、フッタを除く）	(fo:table-row+|fo:table-cell+)
fo:table-row	表の一行	(fo:table-cell+)
fo:table-cell	表のセル	(%block;)+ number-columns-spanned：横結合する列数 number-rows-spanned : 縦結合する行数



SimpleDoc と XSL-FO の表構造

両者を見比べてみると、ほぼ同じ構造をしていることがわかります。表オブジェクトへの変換は、基本的にはこの構造の「移し替え」と考えることができます。

表を処理するテンプレート

次に表を処理するテンプレートを示します。

表のプロパティ定義部分

```
<xsl:attribute-set name="table.data">
  <xsl:attribute name="table-layout">fixed</xsl:attribute>
  <xsl:attribute name="space-before">10pt</xsl:attribute>
  <xsl:attribute name="space-after">10pt</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="table.data.caption">
  <xsl:attribute name="font-family">sans-serif</xsl:attribute>
  <xsl:attribute name="text-align">start</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="space-after">3pt</xsl:attribute>
  <xsl:attribute name="space-after.precedence">2</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>
```

(12) from-table-column( )関数のパラメータとして使用します。
(13) from-table-column( )関数で参照します。

```
<xsl:attribute-set name="table.data.th">
  <xsl:attribute name="background-color">#DDDDDD</xsl:attribute>
  <xsl:attribute name="border-style">solid</xsl:attribute>
  <xsl:attribute name="border-width">1pt</xsl:attribute>
  <xsl:attribute name="padding-start">0.3em</xsl:attribute>
  <xsl:attribute name="padding-end">0.2em</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="table.data.td">
  <xsl:attribute name="border-style">solid</xsl:attribute>
  <xsl:attribute name="border-width">1pt</xsl:attribute>
  <xsl:attribute name="padding-start">0.3em</xsl:attribute>
  <xsl:attribute name="padding-end">0.2em</xsl:attribute>
</xsl:attribute-set>
```

表を処理するテンプレート

```
<xsl:template match="table">
  <fo:table-and-caption>
    <xsl:if test="title">
      <fo:table-caption xsl:use-attribute-sets="table.data.caption">
        <fo:block start-indent="0em">
          <xsl:apply-templates select="title" mode="make-title" />
       </fo:block>
      </fo:table-caption>
    </xsl:if>
    <fo:table xsl:use-attribute-sets="table.data">
      <xsl:if test="@layout">
        <xsl:attribute name="table-layout">
          <xsl:value-of select="@layout" />
        </xsl:attribute>
      </xsl:if>
      <xsl:if test="@width">
        <xsl:attribute name="inline-progression-dimension">
          <xsl:value-of select="@width" />
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
    </fo:table>
  </fo:table-and-caption>
</xsl:template>

<xsl:template match="table/tltle" mode="make-title">
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="table/tltle">
</xsl:template>

<xsl:template match="col">
  <fo:table-column column-number="{@number}" column-width="{@width}" />
</xsl:template>

<xsl:template match="thead">
  <fo:table-header>
    <xsl:apply-templates />
  </fo:table-header>
</xsl:template>

<xsl:template match="tfoot">
  <fo:table-footer>
    <xsl:apply-templates />
  </fo:table-footer>
```

```
</xsl:template>

<xsl:template match="tbody">
  <fo:table-body>
    <xsl:apply-templates />
  </fo:table-body>
</xsl:template>

<xsl:template match="tr">
  <xsl:element name="fo:table-row">
    <xsl:choose>
      <xsl:when test="@height">
        <xsl:attribute name="block-progression-dimension">
          <xsl:value-of select="@height" />
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="(ancestor::*)[2]/@rowheight">
          <xsl:attribute name="block-progression-dimension">
            <xsl:value-of select="(ancestor::*)[2]/@rowheight" />
          </xsl:attribute>
        </xsl:if>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates />
  </xsl:element>
</xsl:template>


<xsl:template match="th">
  <fo:table-cell xsl:use-attribute-sets="table.data.th">
    <xsl:call-template name="cell-span" />
    <xsl:if test="@valign">
      <xsl:attribute name="display-align">
        <xsl:value-of select="@valign" />
      </xsl:attribute>
    </xsl:if>
    <fo:block>
      <xsl:if test="@align">
        <xsl:attribute name="text-align">
          <xsl:value-of select="@align" />
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
    </fo:block>
  </fo:table-cell>
</xsl:template>

<xsl:template match="td">
  <fo:table-cell xsl:use-attribute-sets="table.data.td">
    <xsl:call-template name="cell-span" />
    <xsl:if test="@valign">
      <xsl:attribute name="display-align">
        <xsl:value-of select="@valign" />
      </xsl:attribute>
    </xsl:if>
    <fo:block>
      <xsl:if test="@align">
        <xsl:attribute name="text-align">
          <xsl:value-of select="@align" />
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
    </fo:block>
```

```
  </fo:table-cell>
</xsl:template>

<xsl:template name="cell-span">
  <xsl:if test="@colspan">
    <xsl:attribute name="number-columns-spanned">
      <xsl:value-of select="@colspan" />
    </xsl:attribute>
  </xsl:if>
  <xsl:if test="@rowspan">
    <xsl:attribute name="number-rows-spanned">
      <xsl:value-of select="@rowspan" />
  </xsl:attribute>
  </xsl:if>
</xsl:template>
```

スタイルシートは長いようですが、基本は SimpleDoc の要素を XSL-FO の要素名にマッピングしているだけです。ポイントを以下に示します。

1. table 要素に指定された layout プロパティは、表の自動レイアウトを制御するものとして、fo:table の layout プロパティに設定します。（auto か fixed の値が有効）
2. table 要素に指定された width プロパティは、表全体の幅として fo:table オブジェクトの inline-progression-dimension プロパティに設定します。
3. col 要素に指定された number, width プロパティは、fo:table-column オブジェクト column-number, column-width プロパティに設定します。これにより列幅が指定できるようになります。列幅は固定値ではなく%値による指定も可能です。table 要素の width プロパティに絶対幅を指定し、列の幅は%指定とするなど、柔軟なパラメータ指定が可能になります(14)。
4. table 要素に指定された rowheight プロパティは、fo:table-row オブジェクト block-progression-dimension に設定します。スタイルシート中では tr 要素を処理するテンプレートから参照するため、"(ancestor::*)[2]/@rowheight"という指定方法となります。
5. tr 要素に指定された height プロパティは、fo:table-row オブジェクトの block-progression-dimension に設定します。これは、table 要素に指定された rowheight プロパティより優先するようにしてあります。
6. th, td 要素に指定された colspan, rowspan プロパティは、各々 fo:table-cell オブジェクトの number-columns-spanned, number-rows-spanned に設定します。これでセル結合が表現できます。さらに align, valign プロパティは fo:table-cell 中に配置する fo:block の text-align に、fo:table-cell の display-align プロパティに設定します。これでセル内での揃えが処理できるようになります。

(14) 更に一部の列は絶対値指定を行い、残りの列は proportional-column-width( )関数を指定して幅を比例配分させるような方法もあります。

表の整形例

以下に表の整形例を示します。

何も指定しないと列幅は列数で等分されます。

記号	読み方	意味
|	縦棒	OR 条件を示します。
?	疑問符	その要素が０回か１回出現することを示します。
,	カンマ	カンマで区切られた順であることを示します。
*	アスタリスク	その要素が０回以上出現することを示します。
+	プラス	その要素が１回以上出現することを示します。
( )	括弧	括弧内に複数の要素をグループ化します。
	記号なし	その要素は１つだけ記述できます。

col 要素で列幅を指定しました。左から 10%, 20%, 40%です。見出し行は、横方向にセンタリング。左 2 列は valign="center" align="center"で縦・横方向にセンタリングを指定しました。

記号	読み方	意味
|	縦棒	OR 条件を示します。
?	疑問符	その要素が０回か１回出現することを示します。
,	カンマ	カンマで区切られた順であることを示します。
*	アスタリスク	その要素が０回以上出現することを示します。
+	プラス	その要素が１回以上出現することを示します。
( )	括弧	括弧内に複数の要素をグループ化します。
	記号なし	その要素は１つだけ記述できます。

表のセル中の縦方向のアライメント指定には、次の点にご注意ください。HTML では valign, CSS2 では vertical-alignment というプロパティをテーブルのセルに指定しアライメントを行わせることができます。しかし XSL でこれに該当するプロパティは、vertical-align ではなく、display-align になります。ここで示したスタイルシートでも valign は display-align にマッピングしています。XSL では vertical-align はインラインレベルの要素にのみ適用されます。

リスト要素の処理

リストとは箇条書きのことです。SimpleDoc では、箇条のラベル部分の形式が異なる 3 つのリスト要素、(1)番号付リスト（ordered list）、(2)番号なしリスト（unordered list）、(3)定義型リスト[15](definition list）が定義されています。

- SimpleDoc のリスト要素（ol, ul, dl）は、リストのラベル部分と本体部分の領域が横に並んだ形式に組む。
- ただし定義型リスト要素（dl）は、リストのラベル部分と本体部分が縦に並ぶ形式も処理できるようにする[16]。
- 番号付リスト要素（ol）は、ラベル部分に編集する番号に様々な形式が適用できるようにする。
- 番号なしリスト要素（ul）は、ラベル部分の文字に様々な指定ができるようにする。

リスト形式の比較

リスト要素を横並びのラベル部と本体部に編集するには、XSL-FO の fo:list-block に編集してやる必要があります。ここで SimpleDoc のリストと XSL-FO のリストを比較してみましょう。

要素	意味	定義
ol	番号付リスト	(li)*
ul	番号なしリスト	(li)*
li	リスト項目	(inline 要素並び)*
dl	定義型リスト	(dt, dd)*
dt	定義する用語	(inline 要素並び)*
dd	用語の定義	(inline 要素並び)*

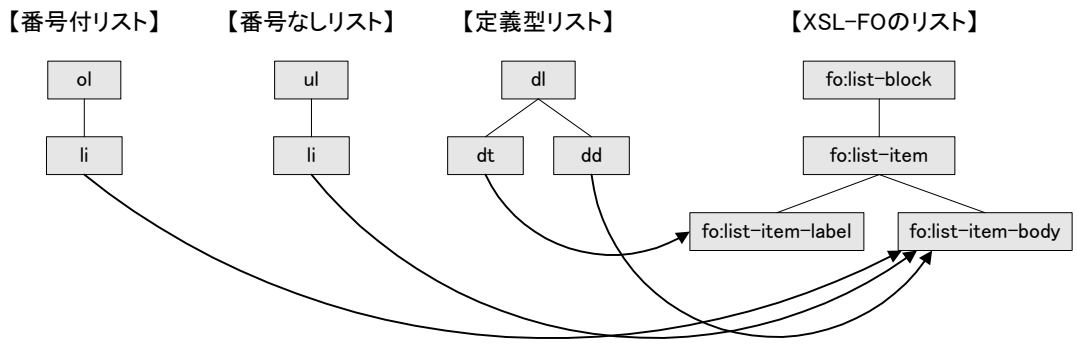
これに対して XSL-FO のリストは次の構造をしています。

要素	意味	定義
fo:list-block	リストをフォーマットするオブジェクト	(list-item+) provisional-distance-between-starts：fo:list-item-label の開始位置と fo:list-item-body の開始位置の距離を指定。 provisional-label-separation：fo:list-item-label の終了位置と fo:list-item-body の開始位置の距離を指定。
fo:list-item	リストの一項目。ラベルとリストの本体を含む。	(list-item-label, list-item-body)
fo:list-item-label	リストの一項目のラベル	(%block;)+
fo:list-item-body	リストの一項目の本体	(%block;)+

両者の対応関係を図に示します。

(15 定義型リストとは、用語集のように項目ごとにラベル文字列を指定する箇条書きのことです。
)

(16) HTML の dl のブラウザ表示と同じ表示形式です。



SimpleDoc と XSL-FO のリスト

注目すべき点として以下のことが挙げられます。

- HTML の ul, ol はブラウザが自動的にリストのラベルを生成してくれた。XSL-FO ではスタイルシートでラベル領域に作成しなければならない。その分自由度が高く、出版物で実用に耐えるラベルの形式を作ることができる。
- dl の場合はスタイルシートで dt の内容をラベル領域に配置すればよい。
- リストのラベル部分の幅を自動的に計算する機構は XSL-FO にはない。したがって、スタイルシートで適切な値を指定しなければならない。

それでは個々のテンプレートを解説します。

番号付リストを処理するテンプレート

番号付リストのテンプレート

```
<xsl:param name="list-startdist-default" select="string('2em')" />
<xsl:param name="list-gap-default" select="string('0.5em')" />

<xsl:attribute-set name="list.item">
  <xsl:attribute name="space-before">0.4em</xsl:attribute>
  <xsl:attribute name="space-after">0.4em</xsl:attribute>
  <xsl:attribute name="relative-align">baseline</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="ol">
  <!-- ラベルの先頭と本体の先頭と距離、ラベルの終了と本体の先頭との距離を決定します。 -->
  <xsl:variable name="start-dist-local">
    <xsl:choose>
      <xsl:when test="./@startdist">
        <xsl:value-of select="./@startdist" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-startdist-default" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:variable name="gap-local">
    <xsl:choose>
      <xsl:when test="./@gap">
        <xsl:value-of select="./@gap" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-gap-default" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
```

```
  <!-- fo:list-block を生成します。 -->
  <fo:list-block provisional-distance-between-starts="{$start-dist-local}"
    provisional-label-separation="{$gap-local}">
    <!-- 下位の li を処理させます。 -->
    <xsl:apply-templates />
  </fo:list-block>
</xsl:template>

<xsl:template match="ol/li">
  <fo:list-item  xsl:use-attribute-sets="list.item">
    <!-- リストのラベルを生成します。-->
    <!-- ラベルの終了位置は label-end( )関数で計算させます。-->
    <!-- ラベルのフォーマットは type プロパティで指定します。既定値は「1.」-->
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end">
        <xsl:choose>
          <xsl:when test="../@type">
            <xsl:number format="{../@type}" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:number format="1." />
          </xsl:otherwise>
        </xsl:choose>
      </fo:block>
    </fo:list-item-label>
    <!-- リストの本体部を生成します。-->
    <!-- ラベルの開始位置は body-start( )関数で計算させます。-->
    <fo:list-item-body start-indent="body-start()"  text-align="justify">
      <fo:block>
        <!-- li 以下の内容を下位のテンプレートに処理させます。-->
        <xsl:apply-templates />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

ラベルと本体部分の位置指定

リストの組版ではラベル部分と本体部分の配置が重要になります。fo:list-block では、

- provisional-distance-between-starts でラベル部分の先頭と本体部分の先頭の距離
- provisional-label-separation でラベル部分の終了と本体部分の先頭の距離

を指定します。テンプレートではこれらを、ol 要素の startdist, gap プロパティから受け取れるようにしてあります。しかし実際にリストが組版されるのは、fo:list-item 以降です。テンプレートではラベル部分の終了位置、本体部分の開始位置指定に label-end( )と body-start( )関数を使用しています[17]。2 つの関数は、fo:list-block に指定された provisional-distance-between-starts、provisional-label-separation の値を内部で参照して位置を計算してくれます[18]。

これらの関数の指定はリスト処理のテンプレートではほとんど「おまじない」に近いものです。リストのレイアウトの考え方がわかれば機械的に適用して問題ありません。

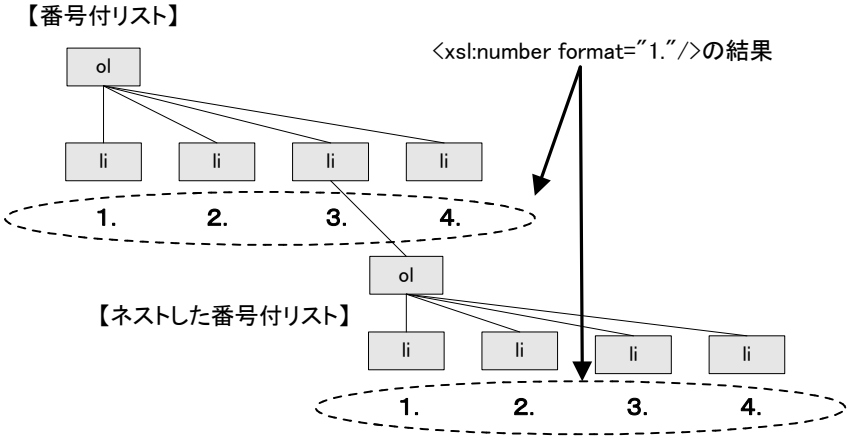
---

[17] これらは XSL 仕様で「Property Value Function」と呼ばれています。

[18] label-end() = fo:list-block の内容領域の幅 - (provisional-distance-between-starts + ラベルの start-indent - provisional-label-separation)
body-start() = ラベルの start-indent + provisional-distance-between-starts

ラベルの書式

リストのラベルは一連番号から編集します。一連番号は既定値では<xsl:number format="1." />で生成しています。xsl:number は入力 XML 文書中の同じレベル（sibling:兄弟）の li 要素をカウントし、現在の li 要素の順序番号を返します。したがって、リストがネストした場合でも正しく処理されます。



xsl:number による番号付リストのラベル作成

ラベルの書式は ol 要素の type プロパティで指定します。例えば、type="(01)"と指定すれば、(01), (02), (03), …というラベルが生成されます。番号部分には以下の指定子が使えます。

番号の書式	出力結果
1	1, 2, 3, 4...
01	01, 02, 03, 04...
a	a, b, c, d, ...x, y, z, aa, ab, ac...
A	A, B, C, D, ...X, Y, Z, AA, AB, AC...
i	i, ii, iii, iv, v...
ア	ア, イ, ウ, エ...
あ	あ, い, う, え...
イ	イ, ロ, ハ, ニ...
一	一, 二, 三, 四...
壱	壱, 弐, 参, 四...

type プロパティには Unicode でゼロと 1 を表す文字と、それ以外の括弧などの約物を指定して書式を表します。書式を入力 XML 文書の側から供給できるので文書の作成の自由度が増すでしょう。

番号付リストの例

入力 XML データ

```
<ol type="a.">
  <li>リストの種類
    <ol type="一.">
      <li>番号なしリスト</li>
      <li>番号付リスト</li>
      <li>定義型リスト</li>
    </ol>
  </li>
  <li>表の要素
    <ol>
      <li>行</li>
```

```
      <li>列</li>
      <li>セル</li>
    </ol>
  </li>
  <li>ブロック要素とインライン要素</li>
</ol>
```

組版結果は次のようになります

a. リストの種類
   一. 番号なしリスト
   二. 番号付リスト
   三. 定義型リスト
b. 表の要素
   1. 行
   2. 列
   3. セル
c. ブロック要素とインライン要素

番号なしリストを処理するテンプレート

番号なしリストのテンプレート

```
<xsl:param name="list-startdist-default" select="string('2em')" />
<xsl:param name="list-gap-default" select="string('0.5em')" />

<xsl:attribute-set name="list.item">
  <xsl:attribute name="space-before">0.4em</xsl:attribute>
  <xsl:attribute name="space-after">0.4em</xsl:attribute>
  <xsl:attribute name="relative-align">baseline</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="ul">
  <!-- ラベルの先頭と本体の先頭と距離、ラベルの終了と本体の先頭との距離を決定します。 -->
  <xsl:variable name="start-dist-local">
    <xsl:choose>
      <xsl:when test="./@startdist">
        <xsl:value-of select="./@startdist" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-startdist-default" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:variable name="gap-local">
    <xsl:choose>
      <xsl:when test="./@gap">
        <xsl:value-of select="./@gap" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-gap-default" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <!-- fo:list-block を生成します。 -->
  <fo:list-block provisional-distance-between-starts="{$start-dist-local}"
    provisional-label-separation="{$gap-local}">
    <!-- 下位の li を処理させます。 -->
    <xsl:apply-templates />
```

```
  </fo:list-block>
</xsl:template>

<xsl:template match="ul/li">
  <fo:list-item xsl:use-attribute-sets="list.item">
    <!-- リストのラベルを生成します。-->
    <!-- ラベルの終了位置は label-end( )関数で計算させます。-->
    <!-- 行頭文字は type プロパティで指定します。既定値は「・」-->
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end">
        <xsl:choose>
          <xsl:when test="../@type='disc'">
            <xsl:text>●</xsl:text>
          </xsl:when>
          <xsl:when test="../@type='circle'">
            <xsl:text>○</xsl:text>
          </xsl:when>
          <xsl:when test="../@type='square'">
            <xsl:text>□</xsl:text>
          </xsl:when>
          <xsl:when test="../@type='bsquare'">
            <xsl:text>■</xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>・</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
      </fo:block>
    </fo:list-item-label>
    <!-- リストの本体部を生成します。-->
    <!-- ラベルの開始位置は body-start( )関数で計算させます。-->
    <fo:list-item-body start-indent="body-start()" text-align="justify">
      <fo:block>
        <xsl:apply-templates />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

ラベル文字の指定

番号なしリストと番号付リストのテンプレートの違いは、リストのラベル部分の処理のみです。番号なしリストではラベルに一律の文字を配置します。ラベル文字の種類は、ul 要素の type プロパティで指定します。既定値は中黒「・」ですが、HTML と同じ disc, circle, square も指定できます。

行頭文字として画像を配置するタイプのテンプレートも作成してみました。ul 要素の type プロパティに「img:ファイル名」の形式で画像ファイルを指定します。

行頭文字として画像を使用するテンプレート

```
<!-- 行頭文字として画像を使用する場合のテンプレート-->
<xsl:template match="ul[substring(@type,1,4)='img:']/li">
  <fo:list-item xsl:use-attribute-sets="list.item">
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end">
        <fo:external-graphic content-height="1.2em" content-width="1.2em"
          src="{substring-after(../@type,substring(../@type,1,4))}" />
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()" text-align="justify">
      <fo:block>
        <xsl:apply-templates />
      </fo:block>
```

```
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

番号なしリストの例

入力 XML データ

```
<ul type="square">
  <li>リストの種類
    <ul type="disc">
      <li>番号なしリスト</li>
      <li>番号付リスト</li>
      <li>定義型リスト</li>
    </ul>
  </li>
  <li>表の要素
    <ul>
      <li>行</li>
      <li>列</li>
      <li>セル</li>
    </ul>
  </li>
  <li>ブロック要素とインライン要素</li>
</ul>
```

組版結果は次のようになります

□ リストの種類
　● 番号なしリスト
　● 番号付リスト
　● 定義型リスト
□ 表の要素
　・ 行
　・ 列
　・ セル
□ ブロック要素とインライン要素

入力 XML データ

```
<ul class="img-bullet" type="leaf">
  <li>リストの種類
    <ul class="img-bullet" type="star">
      <li>番号なしリスト</li>
      <li>番号付リスト</li>
      <li>定義型リスト</li>
    </ul>
  </li>
  <li>表の要素
    <ul class="img-bullet">
      <li>行</li>
      <li>列</li>
      <li>セル</li>
    </ul>
  </li>
  <li>ブロック要素とインライン要素</li>
</ul>
```

結果は次のようになります

- リストの種類
  - 番号なしリスト
  - 番号付リスト
  - 定義型リスト
- 表の要素
  - 行
  - 列
  - セル
- ブロック要素とインライン要素

定義型リストを処理するテンプレート

定義型リストをスタイルシートで fo:list-block に変換するには次の問題があります。

- SimpleDoc の仕様では dl の子要素の出現条件は(dt,dd)*で dl, dt の順に出現することになっている。
- dl は fo:list-block に、dt は fo:list-item-label に、dd は fo:list-item-body にマッピングできるが、fo:list-item を生成するのに対応したタグが SimpleDoc にはない。

上記の問題は、単純に入力 XML 文書のタグの流れに沿った処理を行う型のスタイルシートでは解決がつきません．入力 XML データの中から 1 つの fo:list-item として処理する dt と dd のペアを見つけ出す制御が必要になります．これを実現するテンプレートを次に示します．このテンプレートは、XSL 仕様にサンプルとして掲載されているものを改造したものです．

定義型リストのテンプレート

定義型リストのテンプレート

```
<xsl:template match="dl">
    <xsl:choose>
        <xsl:when test="@type='list'">
            <xsl:call-template name="dl.format.list"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:call-template name="dl.format.block"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template name="dl.format.block">
    <xsl:apply-templates/>
</xsl:template>
<xsl:template name="dl.format.list">
    <xsl:variable name="start-dist-local">
        <xsl:choose>
            <xsl:when test="./@startdist">
                <xsl:value-of select="./@startdist"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$dl-startdist-default"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:variable name="gap-local">
        <xsl:choose>
            <xsl:when test="./@gap">
                <xsl:value-of select="./@gap"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$dl-gap-default"/>
```

```
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <fo:list-block provisional-distance-between-starts="{$start-dist-local}"
provisional-label-separation="{$gap-local}">
        <xsl:call-template name="process.dl.list"/>
    </fo:list-block>
</xsl:template>
<xsl:template name="process.dl.list">
    <xsl:param name="dts" select="/.."/>
    <xsl:param name="dds" select="/.."/>
    <xsl:param name="nodes" select="*"/>
    <xsl:choose>
        <xsl:when test="count($nodes)=0">
            <!-- データ終了：dts, dds にため込まれた要素を処理します.  -->
            <xsl:if test="count($dts)&gt;0 or count($dds)&gt;0">
                <fo:list-item xsl:use-attribute-sets="list.item">
                    <fo:list-item-label end-indent="label-end()">
                        <xsl:apply-templates select="$dts"/>
                    </fo:list-item-label>
                    <fo:list-item-body start-indent="body-start()">
                        <xsl:apply-templates select="$dds"/>
                    </fo:list-item-body>
                </fo:list-item>
            </xsl:if>
        </xsl:when>
        <xsl:when test="name($nodes[1])='dd'">
            <!-- dd は変数 dds に記憶し、自分自身を再帰的に呼び出します. -->
            <xsl:call-template name="process.dl.list">
                <xsl:with-param name="dts" select="$dts"/>
                <xsl:with-param name="dds" select="$dds|$nodes[1]"/>
                <xsl:with-param name="nodes" select="$nodes[position()&gt;1]"/>
            </xsl:call-template>
        </xsl:when>
        <xsl:when test="name($nodes[1])='dt'">
            <!-- dts, dds にため込まれた要素を処理します. -->
            <xsl:if test="count($dts)&gt;0 or count($dds)&gt;0">
                <fo:list-item xsl:use-attribute-sets="list.item">
                    <fo:list-item-label end-indent="label-end()">
                        <xsl:apply-templates select="$dts"/>
                    </fo:list-item-label>
                    <fo:list-item-body start-indent="body-start()">
                        <xsl:apply-templates select="$dds"/>
                    </fo:list-item-body>
                </fo:list-item>
            </xsl:if>
            <!-- dt は変数 dts に記憶し、自分自身を再帰的に呼び出します. -->
            <xsl:call-template name="process.dl.list">
                <xsl:with-param name="dts" select="$nodes[1]"/>
                <xsl:with-param name="nodes" select="$nodes[position()&gt;1]"/>
            </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
            <!-- dl の子供は dt, dd しかないはずです.  -->
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template match="dt">
    <xsl:element name="fo:block" use-attribute-sets="dt">
        <xsl:apply-templates/>
    </xsl:element>
</xsl:template>
<xsl:template match="dd">
    <xsl:choose>
```

```
        <xsl:when test="../@type='list'">
            <xsl:element name="fo:block" use-attribute-sets="dd.list">
                <xsl:apply-templates/>
            </xsl:element>
        </xsl:when>
        <xsl:otherwise>
            <xsl:element name="fo:block" use-attribute-sets="dd.block">
                <xsl:apply-templates/>
            </xsl:element>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

このスタイルシートでは、まず定義型リストを

・ リストのラベル(dt)と本体(dd)が横並びになる fo:list-block にする．(リスト形式)

・ リストのラベル(dt)と本体(dd)を縦並びにする．(HTML 形式)

のどちらにするかを dl 要素の type 属性により決定します．type 属性に"list"が指定されている場合に 前者を選択します．

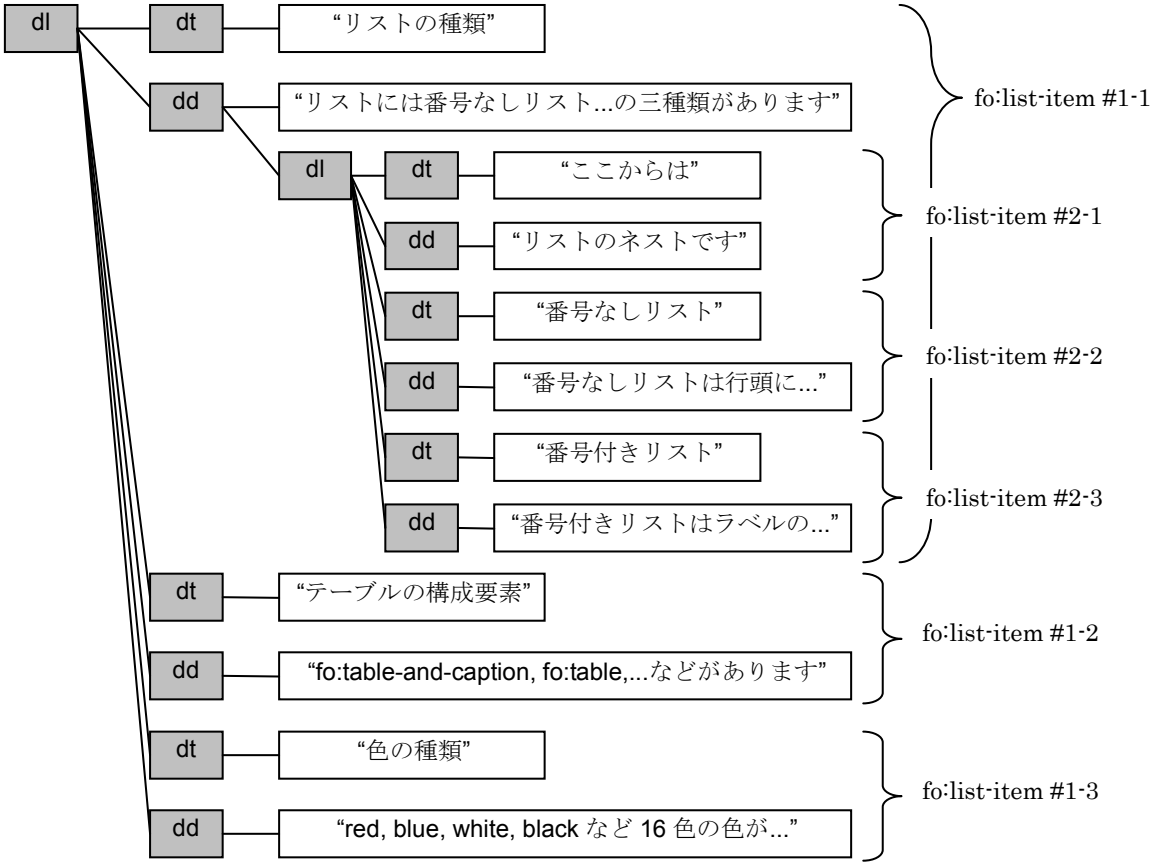
リスト形式の場合、fo:list-block を生成した後、process.dl.list テンプレートに処理を委ねます． process.dl.list テンプレートは、次の処理をしています．(19)

1. process.dl.list は、dl の下位の要素を受け取り、最初の要素から処理をはじめます．
2. dt, dd を取得したらそれぞれを順に、変数 dts, dds に追加で格納します．
3. 次の dt を取得したら、変数 dts, dls に記憶した dt, dd を取り出して fo:list-item 以下を出力します．dts, dds を初期化し、この dt を変数 dts に記憶します．
4. 上記の２つの処理を dl の下のすべての dt, dd を取り出すまで繰り返します．
5. dl の下のすべての dt, dd を処理し終わったら処理を終了します．このとき変数 dts, dds に dt, dd が残っていたら、これを取り出して fo:list-item 以下を出力します．

dt, dl から fo:list-item が生成される様子を図で示します．#1-n は最初の dl から生成されます．#2-n はネストされている dl から生成されます．この図は次の例にある XML 文書です．

(19) このテンプレートでは、HTML のように dt と dt がペアになっていないものも許容する文書モデル <!ELEMENT dl ((dt | dd)+)> のようなパターンでも処理することができます。

【定義型リストの処理】



定義型リストのリスト形式への変換

process.dl.list テンプレートは、自身を呼び出す再帰的な処理形態を取っています．一般的なプログラミング言語では、変数に「値を代入」しこれを使って処理することがあたりまえです．しかし XSLT では変数への値の代入はできません．できるのは初期値を持たせることだけです．このため再帰を使用してループを作成することになります．

HTML 型の配置はテンプレート dl.format.block が行います．dt, dd を出現順に処理するために下位テンプレートを呼び出すだけです．

定義型リストの例

入力データは次のようなものとします

定義型リストのサンプルデータ

```
<dl>
  <dt>リストの種類</dt>
  <dd>リストには番号なしリスト、番号付きリスト、定義型リストの三種類があります.
    <dl>
      <dt>ここからは</dt>
      <dd>リストのネストです. </dd>
      <dt>番号なしリスト</dt>
      <dd>番号なしリストは行頭に配置する文字により、square, circle, などに分類されます. </
dd>
      <dt>番号付きリスト</dt>
      <dd>番号付きリストは、ラベルの書式によりたくさんの種類があります. </dd>
    </dl>
  </dd>
  <dt>テーブルの構成要素</dt>
```

```
  <dd>fo:table-and-caption, fo-table-caption, fo:table, fo:table-column,
fo:tablehead,fo:table-foot, fo:table-body, fo:table-row, fo:table-cell などがあります. </dd>
  <dt>色の種類</dt>
  <dd>red, blue, white, black など 16 色の色が使用できます. また、RGB()関数による指定も可能です. </dd>
</dl>
```

リスト型で変換した結果は次のようになります. fo:list-item-label と fo:list-item-body がわかるよう に、ボーダーをつけました.

リストの種類	リストには番号なしリスト、番号付きリスト、定義型リストの三種類があります. **ここからは** リストのネストです. **番号なしリスト** 番号なしリストは行頭に配置する文字により、square, circle, などに分類されます. **番号付きリスト** 番号付きリストは、ラベルの書式によりたくさんの種類があります.
テーブルの構成要素	fo:table-and-caption, fo-table-caption, fo:table, fo:table-column, fo:tablehead,fo:table-foot, fo:table-body, fo:table-row, fo:table-cell などがあります.
色の種類	red, blue, white, black など 16 色の色が使用できます. また、RGB()関数による指定も可能です.

HTML 型で変換した場合は次のようになります.

リストの種類

リストには番号なしリスト、番号付きリスト、定義型リストの三種類があります.

ここからは

リストのネストです.

番号なしリスト

番号なしリストは行頭に配置する文字により、square, circle, などに分類されます.

番号付きリスト

番号付きリストは、ラベルの書式によりたくさんの種類があります.

テーブルの構成要素

fo:table-and-caption, fo-table-caption, fo:table, fo:table-column, fo:tablehead,fo:table-foot, fo:table-body, fo:table-row, fo:table-cell などがあります.

色の種類

red, blue, white, black など 16 色の色が使用できます. また、RGB()関数による指定も可能です.

PDF 生成に関する機能

XSL Formatter では PDF 生成に関して以下のようなことが可能です。

- 組版結果の PDF に文書情報を設定する。
- PDF のしおりを作成する。
- PDF 文書内部へのリンク（目次から本文へ、等）を設定する。
- 外部へのリンクを設定する。

ここで挙げた PDF 作成時の機能には XSL バージョン 1.0 仕様にはない項目もありますが、アンテナハウス拡張仕様 （参考資料［6］）を使うことで可能になります。

PDF 文書情報

XSL バージョン 1.0 仕様では、組版結果を PDF に出力する際に文書情報を設定することはできませんが、アンテナハウス拡張仕様 （参考資料［6］）によって文書情報の設定が可能です。axf:document-info を使って定義します。axf:document-info は、fo:root の直下で、かつ fo:page-sequence が出現するよりも前にあれば PDF に反映されます。スタイルシートでは、以下のように、SimpleDoc の/doc/head から title、subtitle、auther を PDF 文書情報として設定しています。

PDF 文書情報の出力

```
<xsl:template match="doc">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <axf:document-info name="title" value="{/doc/head/title}" />
    <axf:document-info name="subject" value="{/doc/head/subtitle}" />
    <axf:document-info name="author" value="{/doc/head/author}" />
```

しおりの作成

PDF のしおりも同様にアンテナハウス拡張仕様 （参考資料［6］）を使います。この場合、しおりに出力したい fo:block に axf:outline-level や axf:outline-title を指定することで可能になります。axf:outline-title がない場合、自動的に fo:block の内容がしおりとなります。SD2FO-DOC.xsl では、part ｜ chapter ｜ section ｜ subsection ｜ subsubsection ｜ appendix のタイトルを処理する時点で axf:outline-level を設定してしおりを出力させています。テンプレートは、〈xsl:template name="title.out"〉（タイトル出力用のサブテンプレート）です。

```
<xsl:template name="title.out">
  <xsl:variable name="level"
    select="count(ancestor-or-self::part | ancestor-or-self::chapter |
    ancestor-or-self::section | ancestor-or-self::subsection |
    ancestor-or-self::subsubsection | ancestor-or-self::appendix )" />
    <xsl:choose>
      <xsl:when test="$level=1">
        <fo:block xsl:use-attribute-sets="h1"
          id="{generate-id()}" axf:outline-level="{$level}">
          <xsl:call-template name="title.out.sub" />
          <xsl:value-of select="title" />
        </fo:block>
      </xsl:when>
      <xsl:when test="$level=2">
        <fo:block xsl:use-attribute-sets="h2"
          id="{generate-id()}" axf:outline-level="{$level}">
          <xsl:call-template name="title.out.sub" />
          <xsl:value-of select="title" />
```

```
        </fo:block>
      </xsl:when>
      <xsl:when test="$level=3">
        <fo:block xsl:use-attribute-sets="h3"
          id="{generate-id()}" axf:outline-level="{$level}">
          <xsl:call-template name="title.out.sub" />
          <xsl:value-of select="title" />
        </fo:block>
      </xsl:when>
      <xsl:when test="$level=4">
        <fo:block xsl:use-attribute-sets="h4"
          id="{generate-id()}" axf:outline-level="{$level}">
          <xsl:call-template name="title.out.sub" />
          <xsl:value-of select="title" />
        </fo:block>
      </xsl:when>
      <xsl:when test="$level=5">
        <fo:block xsl:use-attribute-sets="h5"
          id="{generate-id()}" axf:outline-level="{$level}">
          <xsl:call-template name="title.out.sub" />
          <xsl:value-of select="title" />
        </fo:block>
      </xsl:when>
      <xsl:otherwise>
        <fo:block xsl:use-attribute-sets="h5" id="{generate-id()}">
          <xsl:call-template name="title.out.sub" />
          <xsl:value-of select="title" />
        </fo:block>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

リンクの設定

リンクの設定 をするには、fo:basic-link を使います。同一文書内へのリンクは internal-destination プロパティで参照先 id を指定します。このとき、参照先の XSL-FO のオブジェクトは同じ値の id を持っている必要があります。外部へのリンクの場合、external-destination プロパティで参照先を指定します。SimpleDoc では内部/外部共に a 要素を使って表します。参照先は href プロパティで指定しますが、その値が"#"で始まっている場合は内部、そうでなければ外部としてリンクを設定します。

```
<xsl:template match="a[@href]">
  <fo:basic-link>
    <xsl:if test="starts-with(@href,'#')">
      <xsl:attribute name="internal-destination">
        <xsl:value-of select="substring-after(@href,'#')" />
      </xsl:attribute>
      <fo:inline xsl:use-attribute-sets="a">
        <xsl:apply-templates />
      </fo:inline>
    </xsl:if>
    <xsl:if test="starts-with(@href,'#')=false">
      <xsl:attribute name="external-destination">
        <xsl:value-of select="@href" />
      </xsl:attribute>
      <fo:inline xsl:use-attribute-sets="a">
        <xsl:variable name="anchor-texts">
          <xsl:apply-templates />
        </xsl:variable>
        <xsl:apply-templates />
        <xsl:if test="@href!=$anchor-texts">
          <fo:inline>
```

```
            <xsl:text>(</xsl:text>
            <xsl:value-of select="@href" />
            <xsl:text>)</xsl:text>
          </fo:inline>
        </xsl:if>
      </fo:inline>
    </xsl:if>
  </fo:basic-link>
</xsl:template>
```

参照先となる name プロパティを持った a 要素は以下の様に id を持つインラインオブジェクトとして処理します。

```
<xsl:template match="a[@name]">
  <fo:inline id="{@name}">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
```

また、目次や索引から文書内の該当する部分にリンクを設定することも可能です。この場合、参照に用いる id は、generate-id( )関数を使って自動的に id を作成します。generate-id( )関数を呼び出すと、XSLT プロセッサがカレントノードに対応する「IDXP83DL」のような適当な文字列を id 用に作成してくれます。

目次の場合、toc.line テンプレートで項目を出力する部分を以下のように記述します。

```
<fo:basic-link internal-destination="{generate-id()}">
  <xsl:value-of select="title" />
</fo:basic-link>
```

参照先となる本文中の title の出力時にも generate-id( )を使って id を設定します。generate-id( )によって生成される id 値は、同じ要素に対しては常に同じであり、異なる要素に対しては必ず異なります。

参考資料の参照

- 巻末の参考資料一覧に資料番号を自動的に振る。
- 本文中の参考資料への参照において、対応する資料番号を自動作成する。

巻末に参考資料一覧を作成し、本文の中に資料番号で巻末の参考資料を参照できるようにします。この資料番号を自動的に生成して資料一覧を改訂しても本文中の参照番号を編集時に更新しなくてもよいようにします。

SimpleDoc.dtd には資料一覧全体を bib 要素で表し、各資料を li 要素の内容とします。li 要素には ID 番号をつけます。参考資料一覧には資料番号を自動的に振ります。このためのスタイルシートは次の通りです。

```
<xsl:template match="bib">
  <fo:list-block>
    <xsl:apply-templates select="li" />
  </fo:list-block>
</xsl:template>

<xsl:template match="bib/li">
  <fo:list-item end-indent="label-end()" id="{@id}">
    <fo:list-item-label>
      <fo:block>
        <xsl:text>[</xsl:text>
        <xsl:value-of select="position()" />
        <xsl:text>]</xsl:text>
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>
        <xsl:value-of select="." />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

本文中の ref 要素の ref-id=″*xxx*″（*xxx* は参照先 li 要素の id 値）で参考資料を参照します。この参照ラベルを生成するスタイルシートは次のようになります。

```
<xsl:template match="ref">
  <xsl:variable name="target">
    <xsl:call-template name="get-position">
      <xsl:with-param name="id-value" select="@ref-id" />
    </xsl:call-template>
  </xsl:variable>
  <fo:basic-link internal-destination="{@ref-id}">
    <fo:inline>
      <xsl:text>（参考資料 [</xsl:text>
      <xsl:value-of select="$target" />
      <xsl:text>]）</xsl:text>
    </fo:inline>
  </fo:basic-link>
</xsl:template>
```

ref の参照先は、bib の中での li の順番を示すので、position( )関数を使い、ref-id の値が一致する li が bib の中で何番目かを求めます。

索引の作成

・巻末に索引を作成する。

索引の作成方法を順を追って説明します。索引用のテンプレートは index.xsl にまとめられています。詳細は、index.xsl の内容も参照してください。

SimpleDoc では、索引に載せる項目のために index 要素が定義されています。

```
<!ELEMENT index (#PCDATA)>
<!ATTLIST index key CDATA #IMPLIED>
```

key プロパティは index 要素をグループ化させるために使用します。日本語では、単語を索引に出力する場合、その読みを使ってグループ化し、あいうえお順に出力するのが一般的でしょう。しかし、<index>単語</index>ではその読みを得ることは不可能です。したがって、<index key="たんご">単語</index>のように key プロパティに読みを指定します。

Key の作成

索引の項目となる index 要素を、先頭の文字を使ってグループ化させるために、xsl:key を使って名前付きキーを宣言します。

```
<xsl:key name="index-key" match="index" use="substring(@key,1,1)" />
  <xsl:key name="index-value" match="index" use="." />
```

xsl:key はスタイルシート内でトップレベル要素でなければならないので、SD2FO-DOC.xsl スタイルシートの先頭に記述されています。一番目の xsl:key は index-key という名前のキーに、key プロパティの先頭の一文字を値として宣言しています。英語のドキュメントのように、もし index 要素の内容からグループ化可能であれば、use="substring(.,1,1)"として、key プロパティを使わない方法で可能です。二番目の xsl:key では、index-value という名前のキーに index 要素の内容を値として宣言しています。これは、先頭の文字でグループ化したノードセットの中でソートして出力するために使います。

索引ページの作成

index.xsl スタイルシートの、index.create テンプレートから索引の作成が開始されます。最初に、他のページと同様に、fo:page-sequence や fo:flow を生成して、次に索引ページの先頭にタイトルを出力します。

次に、数字およびアルファベットの index 要素を処理し、その次に日本語の index 要素を処理します。

```
<xsl:template name="index.create">
  <fo:page-sequence master-reference="PageMaster-index">
  <!-- ヘッダ領域に文書名を設定する -->
    <fo:static-content flow-name="xsl-region-before">
      <fo:block font-size="7pt" text-align="center"
        border-after-width="thin" border-after-style="solid">
        <xsl:value-of select="/doc/head/title" />
      </fo:block>
    </fo:static-content>
    <fo:flow flow-name="xsl-region-body">
      <fo:block>
      <!-- タイトルは全段抜きとする -->
        <fo:block span="all" line-height="15mm" background-color="#9bd49d"
          text-align="center" font-size="20pt" space-after="10mm"
```

```
          axf:outline-level="1" id="index-page">
          INDEX
        </fo:block>
        <fo:block span="all">
           
        </fo:block>
        <fo:block xsl:use-attribute-sets="index">
          <!-- 数字とアルファベット -->
          <xsl:call-template name="index.create.mainALPHA" />
          <xsl:if test="(/doc/@lang = 'ja')
            or (/doc/@lang = '') or not(/doc/@lang)">
            <!-- かな -->
            <xsl:call-template name="index.create.mainKANA" />
          </xsl:if>
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

index 要素をグループ化して取り出す

″index.create.mainALPHA″ と″index.create.mainKANA″がアルファベットと日本語（かな）の index を生成するテンプレートです。処理内容はどちらもほぼ同じです。″index.create.mainALPHA″ を例に説明します。

索引の作成で難しいと思われるのは、対象となるノードをグループ化して取り出す処理です。このために、前述した xsl:key ともうひとつ index.xsl の先頭にあるデータを使用します。

```
<!DOCTYPE DOCUMENT [
<!ENTITY ALPHA "'@1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ'">
<!ENTITY KANA "'あいうえおかきくけこさしすせそたちつてとなにぬねのはひふへほまみむめもやゆよらりるれろわをん'">
]>
```

アルファベット用、かな文字用のそれぞれの文字をエンティティとして定義し、グループ化のために使います。このエンティティから一文字ずつ取り出し、それをキーとして持っているノード集合を取り出します。

```
<xsl:template name="index.create.mainALPHA">
  <xsl:param name="lettersALPHA" select="&ALPHA;" />

  <!-- (1) 0 ～ 9, A ～ Z のうちのどれか１つを処理する -->
  <xsl:call-template name="index.create.section">
    <xsl:with-param name="letter" select="substring($lettersALPHA,1,1)" />
  </xsl:call-template>

  <!-- (2) 0 ～ 9, A ～ Z のうちの次の１つを処理するための準備 -->
  <xsl:variable name="remainderALPHA" select="substring($lettersALPHA,2)" />

  <!-- (3) 0 ～ 9, A ～ Z のうちの次の１つを処理する -->
  <xsl:if test="$remainderALPHA">
    <xsl:call-template name="index.create.mainALPHA">
      <xsl:with-param name="lettersALPHA" select="$remainderALPHA" />
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

(1)の部分で、キーとなる一文字をパラメータとして″index.create.section″テンプレートに渡し、″index.create.section″テンプレートの中でノード集合が処理されます。(2)では次の一文字を取り出し、(3)で″$remainderALPHA″を呼び出す（再帰させる）ことにより、全ての文字を処理します。

ノード集合の出力

″index.create.section″テンプレートでは、キーとなる文字を index-key として持つノード集合を出力します。

```
<xsl:template name="index.create.section">
  <!-- 処理するキャラクタを受け取る -->
  <xsl:param name="letter" />

  <!-- キーの値が letter のノードセットを変数 terms にセットしてソートする-->
  <xsl:variable name="terms" select="key('index-key',$letter)" />
          <!-- 受け取ったパラメータと先頭 1 文字が同じ index を取得 -->

  <xsl:if test="$terms">
    <!-- キー文字の出力 -->

    <fo:block font-weight="bold" text-align="center" space-before="1em">
    <!-- 受け取った 1 文字をそのまま出力 -->
      <xsl:value-of select="$letter" />
    </fo:block>

    <!-- 同一テキストを重複処理しないようにする -->
    <xsl:for-each select="$terms[not(.=preceding::index)]">
      <!-- key（読み）順にソート -->
      <xsl:sort select="@key" />
      <fo:block text-align-last="justify"
        axf:suppress-duplicate-page-number="true">
        <!-- テキストをそのまま出力 -->
        <xsl:value-of select="." />

        <fo:leader leader-pattern="dots" leader-length.optimum="3em" />
        <!-- 同一テキストを取得 -->
        <xsl:variable name="terms2" select="key('index-value',.)" />

        <xsl:if test="$terms2">
          <fo:inline>

            <!-- 同一テキストが複数箇所にあったときそれらのページ番号をすべて出力する -->
            <xsl:apply-templates select="$terms2" mode="line" />

          </fo:inline>
        </xsl:if>
      </fo:block>

    </xsl:for-each>
  </xsl:if>
</xsl:template>
```

axf:suppress-duplicate-page-number プロパティ（アンテナハウス拡張仕様 （参考資料［6］））を使うと、同一ページ番号が連続した場合に削除することが可能です。

A

Antenna House 1,1,2,3,3,5,7,7,8

X

XSL-FO 2,10,11,11,13,13

axf:suppress-duplicate-page-number を使わなかった場合

axf:suppress-duplicate-page-number を使った場合

その他

mode を使用する

SD2FO-DOC.xsl スタイルシートには、本書で説明した他に article.xsl というスタイルシートが含まれています。これは、XML 文書の doc 要素に class="article"と指定した場合に、表紙/目次/索引無しの組版をするようになっています。全てのテンプレートには、mode="article"のように mode が指定され、SD2FO-DOC.xsl スタイルシートのテンプレートとは別の処理を行うようになっています。このようにテンプレートに mode プロパティを使うことによって、任意の要素の処理を分けることができます。

付録

参考資料

[1] Extensible Stylesheet Language (XSL) Version 1.0,
W3C Recommendation 15 October 2001,
http://www.w3.org/TR/2001/REC-xsl-20011015/
Extensible Stylesheet Language (XSL) Version 1.1,
W3C Recommendation 05 December 2006,
http://www.w3.org/TR/2006/REC-xsl11-20061205/

[2] （日本語訳） TR X 0088:2003 拡張可能なスタイルシート言語（XSL）1.0,
発行年月：平成 15 年 9 月 1 日,
発行者：日本工業標準調査会 標準部会,
発行所：日本規格協会

[3] XSL Transformations (XSLT) Version 1.0
W3C Recommendation 16 November 1999,
http://www.w3.org/TR/xslt

[4] （日本語訳） XSL Transformations （XSLT）バージョン 1.0
W3C 勧告 1999 年 11 月 16 日,
http://www.infoteria.com/jp/contents/xml-data/REC-xslt-19991116-jpn.htm

[5] SmartDoc, http://www.asahi-net.or.jp/~dp8t-asm/java/tools/SmartDoc/index_ja.html

[6] アンテナハウス拡張仕様 http://www.antenna.co.jp/XSL-FO/data/ext4.htm

索引