XML 仕様の概説

2001 年 9 月

アンテナハウス株式会社

Table of Contents

XML の仕様(1)

「拡張可能なマーク付け言語（XML）1.0」を概説します。以下では、この仕様書のことを「XML 基本仕様書」と言います。

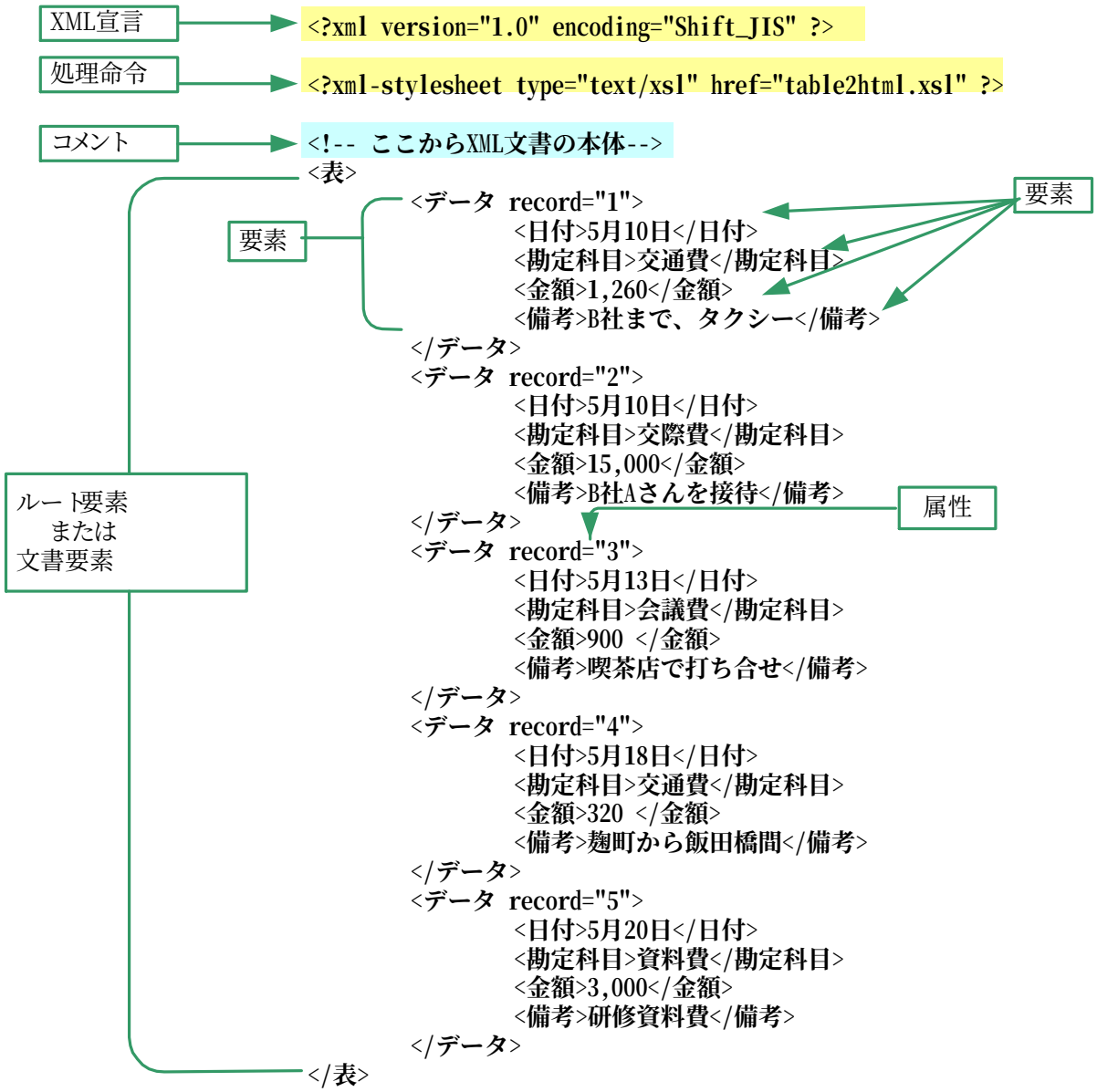
XML 文書の基本と整形式(Wellformed)制約

簡単な XML 文書

次の図は簡単な XML 文書の例です（【XML 文書１）。文書の先頭の<?xml から始まる行は XML 宣言と言います。

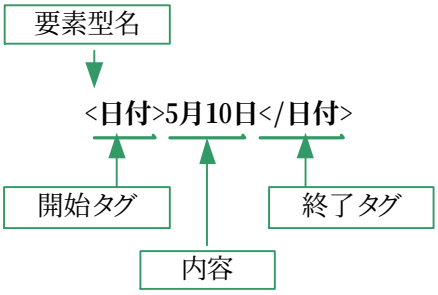
2 行目は処理命令です。処理命令には XML 文書を処理するアプリケーションに対する指示を記述します。

3 行目はコメントです。コメントは文書の文字データの一部ではなく、通常は XML 文書を編集する人間のためのメモ書きです。



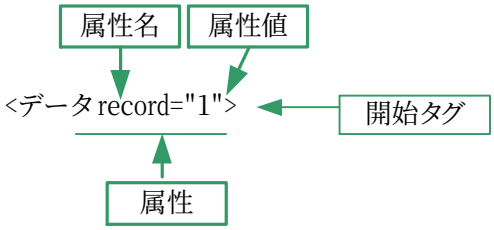
上の XML 文書の例で、XML 宣言は省略可能で、省略時の既定値が決まっています。また、処理命令、コメントは必要に応じてつけるものです。

XML 文書の本体は、図のルート要素（または文書要素と言います）です。そして基本構成単位は要素です。要素は開始タグ、終了タグ、内容からなります。開始タグ、終了タグは要素の始まりと終わりを表すものです。また、タグの中の名前は要素型名と言います。



「XML 基本仕様書」で要素型名に使える文字の種類が規定されています。半角アルファベット、漢字、ひらがな、全角カタカナなどを使うことができます。しかし、全角アルファベット、半角カタカナを要素型名に使うことはできません。

データ要素には属性が設定されています。属性の指定は開始タグまたは空要素タグの中で、属性名と属性値のペアで指定します。



XML 文書はテキスト（文字）です。但し、各種の符号化方式を使用することができますのでメモ帳などのプレーンテキスト・エディタで正しく表示できるとは限りません。

イメージ図形などのバイナリ形式ファイルは、外部実体と言うもので定義し、本文中から実体参照します。但し、外部実体を使わないで単に属性値にファイルの URL を指定してアプリケーションに処理を任せることもできます。

整形式（Wellformed）制約

整形式制約とは、XML 文書として認められるための条件です。大雑把に言えば次の 3 つの条件となります。

1. ルート要素がひとつで、他の要素はルート要素の子孫となる（ルート要素は他の要素に含まれない）。
2. 開始タグと終了タグがセットになっている。HTML のように開始タグや終了タグを省略することはできない。
3. ルート要素に含まれる要素は、互いに親子または兄弟の関係になっている。ひとつの要素の開始タグと終了タグが、他の要素の開始タグや終了タグと入れ違いにならない。

整形式（Wellformed)の他の条件

「XML 基本仕様書」は上の 3 つの条件の他に次のことを XML 文書の条件として要求しています。しかし、次の 2 項を完全に理解するのはかなり大変です。

- 「XML 基本仕様書」で定義するすべての整形式制約に従う。
- 文書内で直接的、間接的に参照される解析対象実体が整形式である。

【XML 文書１】は整形式（Wellformed)制約を満たす（正しい）XML 文書です。

次の例も、正しい (Well formed な）XML 文書です。

```
<root>
  <section>セクション１
    <paragraph>段落１- １</paragraph>
    <paragraph>段落１- ２</paragraph>
  </section>
  <section>セクション２
    <paragraph>段落２- １</paragraph>
    <paragraph>段落２- ２</paragraph>
  </section>
</root>
```

次の例は正しくない（Not wellformed な）XML 文書[1]です。

```
<root>
  <section>セクション１
    <paragraph>段落１- １</paragraph>
    <paragraph>段落１- ２</paragraph>
    </section>
  <section>セクション２
    <paragraph>段落２- １</paragraph>
    <paragraph>段落２- ２</section></paragraph>
</root>
```

この XML 文書が正しくない理由は、<paragraph>の終了タグの前に<section> の終了タグがあるため開始タグと終了タグが入れ違いになっています。

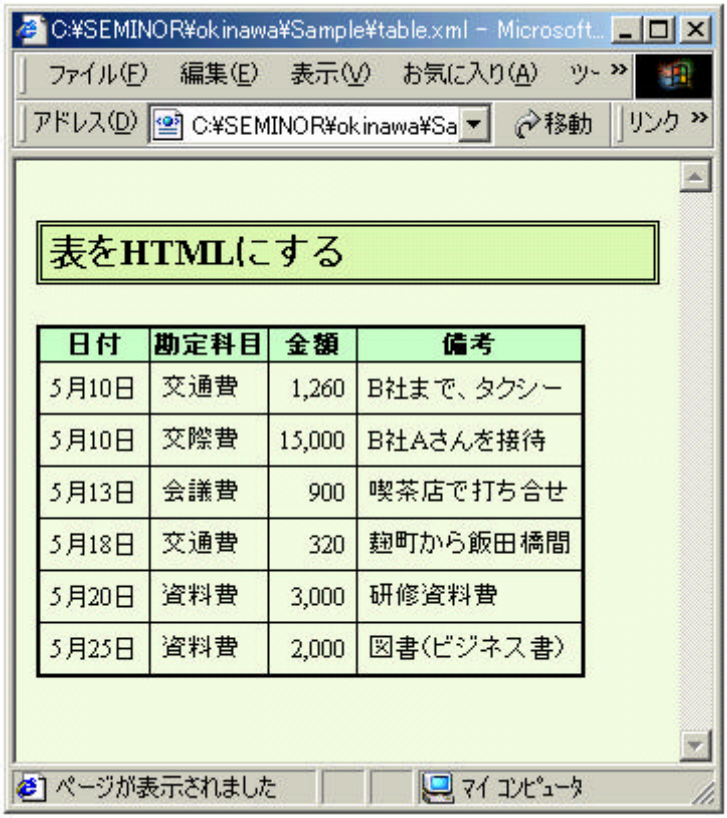
整形式(Wellformed)検証パーサー

XML 文書が整形式(Wellformed)制約を満たしているかどうかを検証する機能提供するのが整形式(Wellformed)検証パーサーです。

[1]厳密には、XML 文書であると認められるには Wellformed でなければならないので、Not wellformed な XML 文書というものは定義上存在しません。XML 基本仕様書では、XML 文書の候補をテキストオブジェクトと表現していますが、この資料では、XML 文書であることを意図したものという意味で、正しくない XML 文書と表現したいと思います。

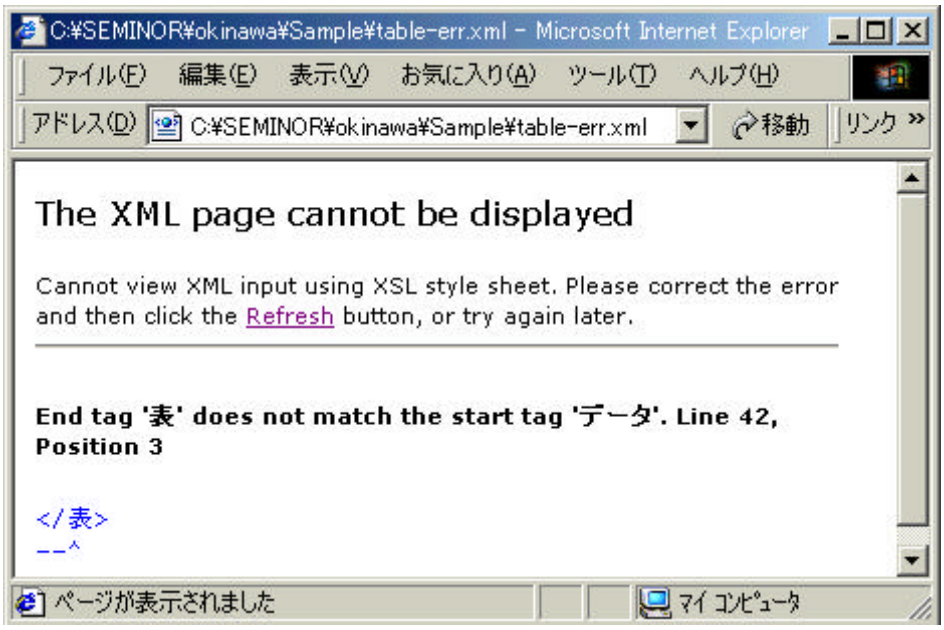
XML 文書の処理

【XML 文書１】は InternetExplorer5 で次のように表示できます。詳しい仕組みは後で説明します。

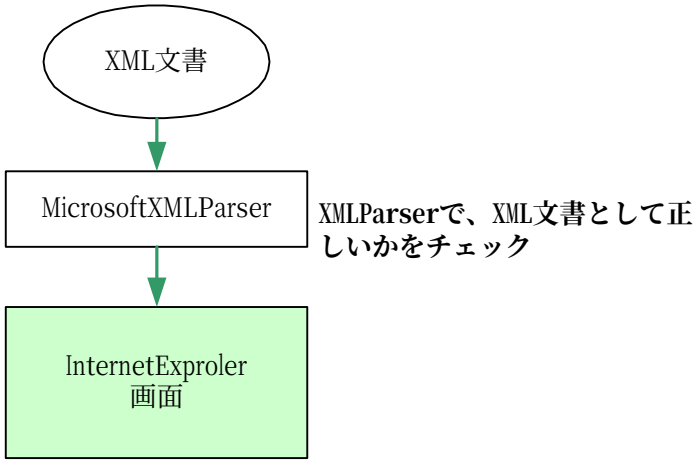


【XML 文書１】の最初の</データ> タグ（データ要素の終了タグ）を削除してみます（【XML 文書２】[Sample\table-err.xml]）。そうしますと開始タグと終了タグの対応がとれなくなりますので Wellformed の条件を満たさなくなります。

【XML 文書２】を、IE5 で読みますと、次のようにエラーになってしまいます。



この理由は、IE5 は、XML 文書を読むとき MicrsoftXMLParser という XML パーサーを使って XML 文書が整形式（Wellformed）かどうかをチェックしながら読んでいるためです。もし、読もうとしているファイルが整形式（Wellformed）でないとエラーになります。上の画面は、整形式でないことを示していますが、このようにエラー・メッセージが出て止まってしまい文書を読み込むことができません。もちろん表示もできません。



HTML との違い

HTML 文書はテキスト・エディタで編集することができます。そして HTML 文書の場合、多少文法的に誤っていても、ブラウザは適切な表示をしてくれます。ところが、XML 文書は少なくとも整形式（Wellformed)制約を満たさないと、IE5 のみでなく、他の XML アプリケーションでも処理ができません。テキスト・エディタで XML を編集できるからと言っても、実際に、テキスト・エディタで XML を作成・編集するには XML の仕様についてある程度詳しい知識をもっていることが必須であるということです。

MicrosoftXMLParser は、Windows の System ホルダに導入され、InternetExplorer が使うのみでなく、他の目的にも使うことができます。たとえば、他のアプリケーションから XML パーサーとして利用することもできます。XML パーサーはこのように XML 文書を処理する際のインフラとして機能するものということができます。後で、MSXML パーサーの他の機能についても説明します。

DTD と妥当性(Valid)制約

XML では文書の型(クラス)のための制約を文書型定義または DTD（Document Type Definition の略）によって定義します。DTD は XML 基本仕様書に規定しているマーク付け宣言を使って記述します。

簡単な DTD の例--表の DTD

簡単な XML 文書の例として紹介した【XML 文書１】の文法は次の DTD です。

DTD

マーク付け宣言

```
<!ELEMENT 表        (データ+) >
<!ELEMENT データ    (日付,勘定科目,金額,備考) >
<!ATTLIST データ    record CDATA #REQUIRED >
<!ELEMENT 日付      (#PCDATA) >
<!ELEMENT 勘定科目  (#PCDATA) >
<!ELEMENT 金額      (#PCDATA) >
<!ELEMENT 備考      (#PCDATA) >
```

マーク付け宣言

DTD の文法を記述するために使います。マーク付け宣言には、この例に示した要素型宣言、属性リスト宣言のほか、実体宣言、記法宣言があります。

<!ELEMENT

要素型宣言と言います。要素型宣言に続いて、要素型名とその要素に許される子供を指定します。

表

要素の名前が「表」であることを指定します。

(データ+)

「表」要素に許される子供は、「データ」要素です。＋（プラス記号）は子供の要素が 1 回以上出現することを示します。

>

要素型宣言の終了区切りです。

データ (日付,勘定科目,金額,備考)

「データ」要素には、「日付」、「勘定科目」、「金額」、「備考」という要素がこの順番で 1 回ずつ出現します。

<!ATTLIST

属性リスト宣言と言います。要素に対して属性と属性値を関連付けるために使用します。

データ record CDATA #REQUIRED

「データ」要素は、「record」という属性を持ちます。「record」属性の型は CDATA（文字列型）であり、#REQUIRED（必須）であることを示します。

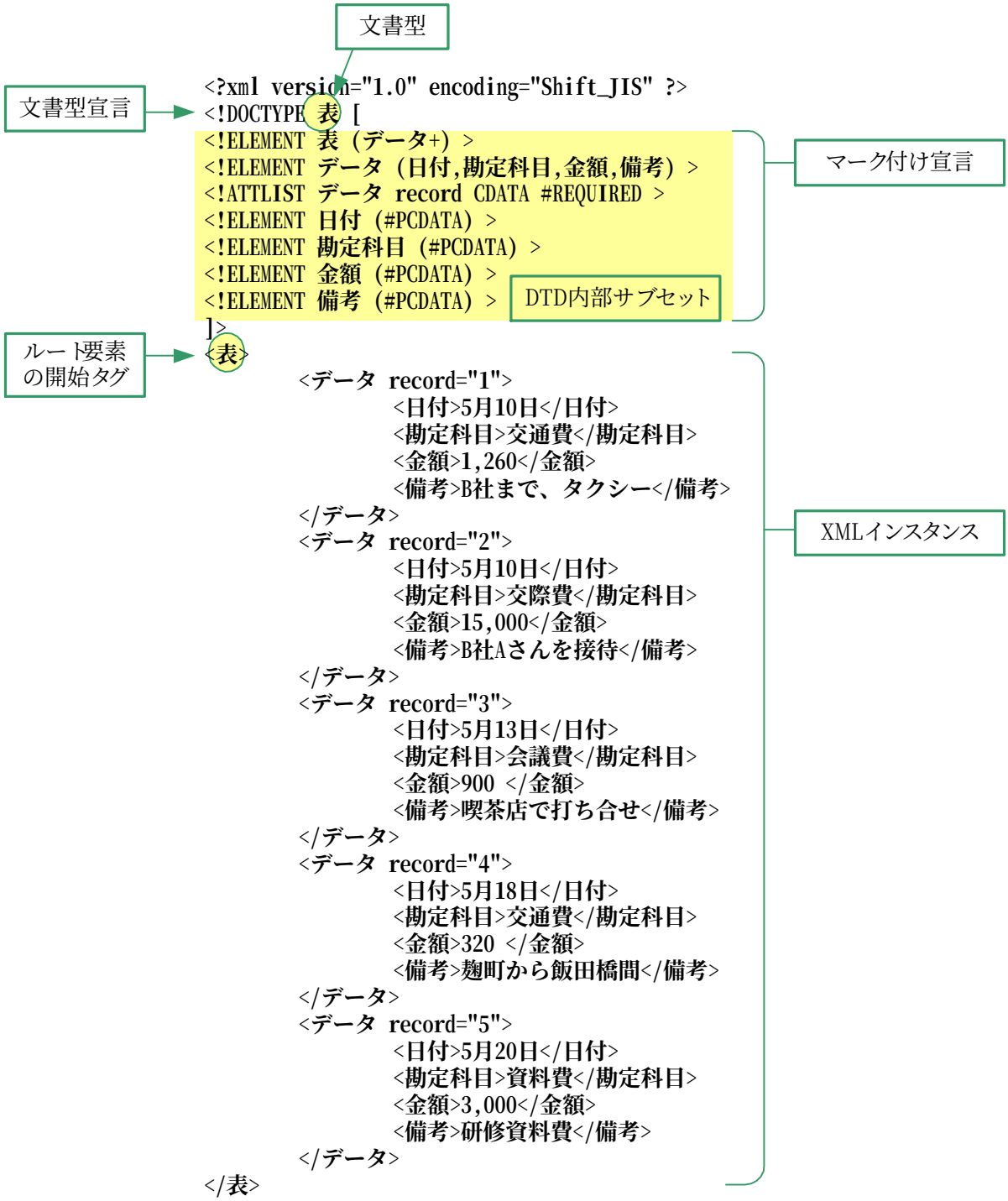
日付 (#PCDATA)

日付の内容は、文字データであることを示します。

XML 文書を DTD に関連付ける

XML 文書には文書型宣言を含むことができます。文書型宣言によって XML 文書と DTD を関連付けることができます。

【XML 文書１】XML 文書と「表」DTD を関連付けると次のようになります(【XML 文書３】[Sample \table-interDTD.xml])。



<!DOCTYPE

文書型宣言（ドキュメント型宣言）と言います。文書型宣言に続いて、文書型名（DTD の名前）と DTD の ID もしくは DTD 本体を置きます。

表

文書型名が「表」であることを示します。

[

DTD の開始を示します。

]>

DTD の終了を示します。

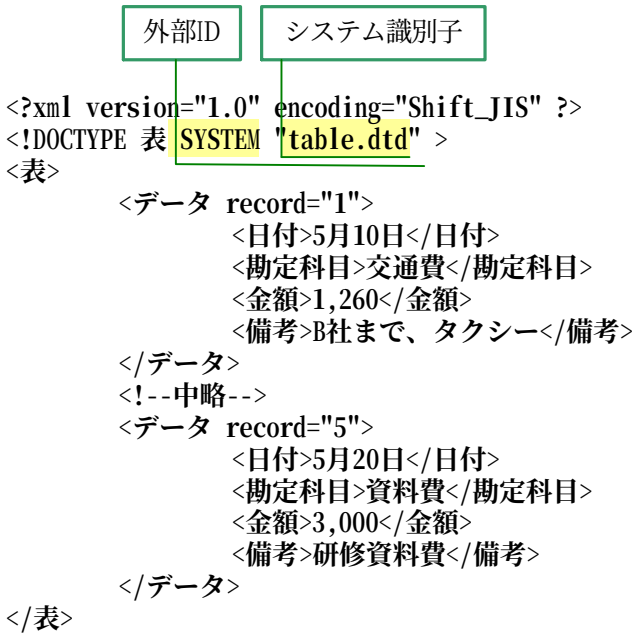
上のように[　]内に置いた DTD を内部 DTD と言います。

外部 DTD

外部 ID を使って XML 文書を外部 DTD ファイルに関連付けることができます。

外部 ID を使って外部 DTD ファイルに関連付けた例

「表」DTD の内容をファイル「table.dtd」に保存したとします。そのとき【XML 文書３】は次のように表すことができます(【XML 文書４】[Sample\table-interDTD.xml])。



SYSTEM "table.dtd"

外部 ID といいます。

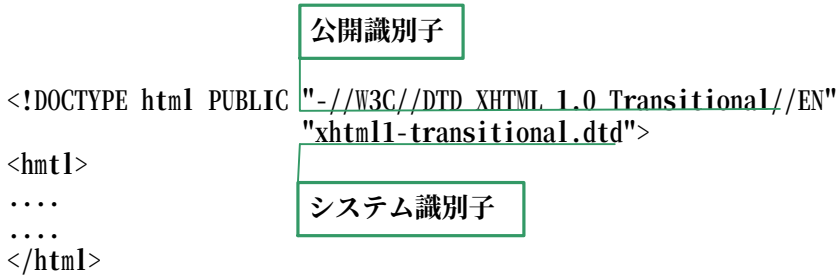
"table.dtd"

システム識別子といい、外部 DTD の URI です。この例では、table.dtd ファイルが XML ファイルと同じホルダにあることを示しています。

公開識別子

外部 ID には、公開識別子を含むことができます。公開識別子は、SGML の時代からよく使われてきたもので、リソースに一定の名前を与えて、その名前を使って参照するものです。但し、残念ながら名前の付け方には標準的基準がありません。このため、公開識別子のみを頼りにリソースを探せないことも起こりえます。そこで XML では、公開識別子の後ろには、必ず、システム識別子を置かなければなりません。

次の例は、XHTML の Transitional-DTD の公開識別子を指定した例です。アプリケーションは DTD を公開識別子で探しますが、見つからないときは、後ろのシステム識別子で探します。



妥当性(Valid)制約

整形式 XML 文書に DTD を対応つけて、ルート要素以下のマークアップが DTD の文法に従っていることを検証したものが、妥当な XML 文書となります。

妥当性(Valid)制約

- 文書型宣言「<!DOCTYPE name ... >」を配置します。name は文書型名です。
- 文書型名は、XML 文書のルート要素の型に一致します。
- XML 文書は DTD の文法に従ってマーク付けされていること。
- その他「XML V1.0 仕様書」で定める妥当性制約を全て満たす(2) ことを確認します。

XML 文書のテキスト

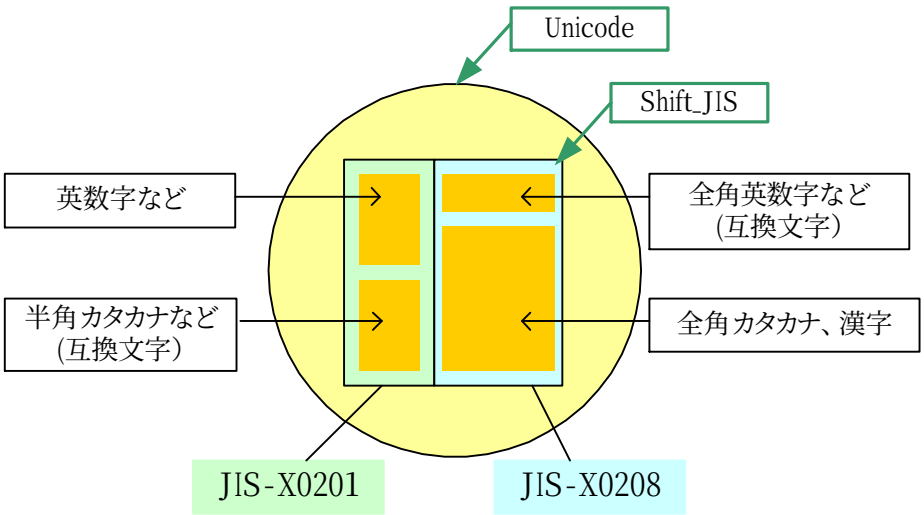
文字

XML 文書で使える文字の種類は Unicode で定義されている文字です。文字データとして使える文字の範囲は次の通りです。

文字の範囲(Unicode のコードポイント)

#x9、#xA、#xD、[#x20-#xD7FF]、[#xE000-#xFFFD]、[#x10000-#x10FFFF] (3)

XML 宣言で文字符号化方式を指定すれば Shift_JIS などの異なる符号化方式を扱うこともできます。Shift_JIS で使える文字の種類は、X0201 と X0208 をあわせたもので、Shift_JIS で使える文字種は全て、Unicode の文字の種類に含まれています。



Shift_JIS の文字種以外で Unicode にある文字については、Unicode のコード・ポイントを文字参照の形で直接指定することで使用できます。XML 文書の中で Shift_JIS で符号化したとしても、文字参照では Unicode のコードポイントで参照しますので注意してください。例えば、次の表は、Unicode のコード表の一部、16 進 00a0 から 16 進 00cf までの部分です。例えば、copyright 記号©のコードポイントの参照は、「©」のように行います。

(2)大きな文書では妥当性制約をすべて満たすかどうかは、人間の眼ではなかなか分かりません。妥当性検証パーサーでチェックします。
(3) Unicode の 6.8 節互換性領域に入る文字[#xF900-#xFFFD] は使用を避けるのが望ましいとされています。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
&#00a		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
&#00b	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
&#00c	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï

名前文字

要素型名、属性名、実体名、文書型名などの名前に使える文字には、内容（#PCDATA)に使える文字の種類よりも強い制約があります。

名前文字

先頭(名前開始文字)として**数字、**"."、"-"、結合文字、エクステンダといわれる文字は使えません。

また、先頭に限らず名前文字には互換性文字（全角英数字、半角カタカナ）は使えません。詳細は、XMLV1.0 仕様書付属書 B.文字クラスを参照のこと。

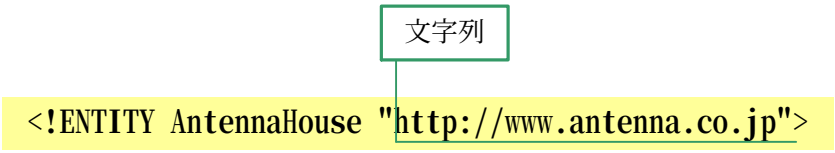
名前文字の大文字と小文字は違うものとして見なされます。SGML では大文字と小文字を区別しませんでした。従って HTML では、タグの文字が大文字でも小文字でも問題ありませんでした。しかし、XML では大文字・小文字を区別しますので注意が必要です。

マーク付けの開始文字

アンド記号(&)、不等号(小なり)(<)の２つは、マーク付けの開始の区切り文字なので、文字データ中ではそのまま使うことはできません。番号による文字参照または、文字列&、<を使用して別扱いしなければなりません。

文字列

属性値、内部実体、外部識別子などの内容の指定には、引用符で囲まれた文字列を使います。文字列には引用符自体は含まず、文字列の中に引用符自体を含めたい時は、外側の引用符と異なる引用符を使うか、それとも、アポストロフィー（一重引用符）は'、二重引用符は"を使います。

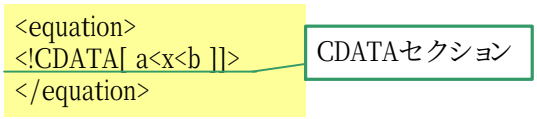


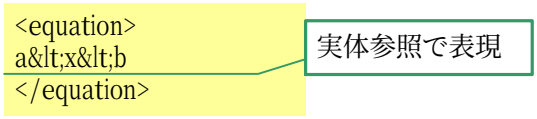
CDATA セクション

CDATA セクションは、<や&をマーク付けの開始区切り文字として認識したくない時に使うためのものです。文字データ中で開始タグや終了タグを、マーク付けとしてではなく、文字として認識させることができます。

- CDATA セクションは、文字データが出現するところであれば、どこにでも置くことができます。
- CDATA セクションには、]]>を除く、任意の文字を入力できます。
- CDATA セクションを入れ子にすることはできません。

次の図の最初の例は、**a<b** という不等式を CDATA セクションに入れた例です。これは 2 番目の例のように CDATA セクションを使う代わりに文字参照を使って表わすこともできます。





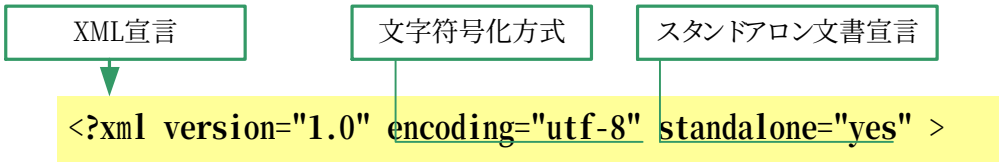
XML 仕様における空白の扱い

XML の仕様では、文字データ中で空白として扱われる文字コードは、英文空白(#x20)、タブコード(#x9)、改行(#xA)、復帰(#xD)の 4 種類です。但し、改行と復帰が連続した場合、または復帰は、XML パーサーが XML 文書を読み込む際に、ひとつの改行に統一します。

XML 文書の文字データ中の空白はアプリケーションにそのまま引き渡されなければならないことになっています。しかし、その後の空白の処理はアプリケーションに任されます。但し、要素に xml:space という属性を加え、その値として preserve を指定することにより、アプリケーションにすべての空白を保存する意図を伝えることができるとされています。xml:space="default"とすることで、アプリケーションのデフォルトの空白処理モードを適用可能となります。この属性は、xml:space 属性の別の指定で上書きされない限り、指定した要素の内容に現れるすべての要素に適用されます。

XML 宣言

XML 文書の先頭には XML 宣言を置くことがあります。XML 宣言では、「XML 基本仕様」のバージョン番号、文字符号化方式、スタンドアロン文書宣言を宣言します。



文字符号化方式（オプション）

XML 文書の文字符号化方式は、既定値が UTF-8 または UTF-16 に決まっていますので、この例のように Shift_JIS を使うときは、XML 宣言で、encoding="Shift_JIS"として符号化方式名を指定しなければなりません。これを省略すると、XML 文書の内容は UTF-8 または UTF-16 で符号化されていると見なされます。従って、XML 宣言の encoding を省略したにも関わらず、文書の内容が Shift_JIS で符号化されているならば、XML 文書を読んだアプリケーションで文字化けとなります。

符号化名称	簡単な説明
UTF-8	Unicode の可変長表現。英数字は US－ASCII と互換
UTF-16	Unicode の 2 バイト固定表現。但し、サロゲートペアを含む。
ISO-10646-UCS-2	Unicode の ISO 標準版(2 バイト固定)
ISO-10646-UCS-4	Unicode の ISO 標準版(4 バイト固定)
ISO-8859-1 から ISO-8859-9	ISO 8859 シリーズ。1 は欧米の文字コードを含む。
ISO-2022-JP	インターネット用日本語符号化（JIS-X0201Roman＋JIS X0208)
Shift_JIS	Windows95/98/Me で使われる日本語符号化方式（JIS-X0201＋JIS X0208)

EUC-JP	Unix でよく使われる日本語符号化方式

XML アプリケーションは、上記以外の文字符号化方式を認識しても良いことになっています。符合化方式の名称は、Internet Assigned Numbers Authority(IANA)に登録した名称を使うのが望ましいとされています。Unicode というような名称は、IANA の登録名ではありませんので使えません。IANA では登録名の大文字・小文字は区別しません。

スタンドアロン文書宣言(オプション)

スタンドアロン文書宣言で standalon="yes"であるとは、その XML 文書実体の外部に XML プロセサからアプリケーションに渡される情報に影響を与えるマーク付け宣言が存在しないことを意味します。これが省略された場合、standalon="no"とされ、文書実体の外部に影響を与えるマーク付け宣言が存在することを意味します。

文書型宣言

文書型宣言は XML 文書の最初の要素の前に置き、次の 2 つの目的に使えます。

1. 文書の論理構造の制約条件を定義する
2. 予め定義された実体を使用するための機構を提供する。

1 番目の文書型宣言で XML 文書の論理構造の制約条件を定義するのは外部 DTD、内部 DTD を参照することで行います。この例は既に説明しました。

次の図は、2 番目の文書型宣言の中で様々な実体を宣言する例です。XML 文書の本文中では、ここで宣言した実体を実体参照によって利用します。

文書型宣言

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sample [
<!ENTITY part1 SYSTEM "chapter1.xml">                    外部解析対象実体の宣言

<!NOTATION emf SYSTEM "image/emf">                       記法宣言
<!ENTITY fig1 SYSTEM "graphics/sample1015.wmf" NDATA emf>
                                                         図形ファイルのパスの宣言
<!ENTITY bull    "&#38;#x2022;">
<!ENTITY rarr    "&#x2192;">                             文字実体の宣言
<!ENTITY uarr    "&#x2191;">

<!ENTITY AntennaHouse "http://www.antenna.co.jp">
]>                                                       内部解析対象実体の宣言
...
```

マーク付け宣言

要素型宣言

要素型宣言では、要素型の名前を決めると同時に、その要素の内容に制約を加えます。一つの要素型を 2 回以上宣言すると妥当性制約違反となります。

要素型宣言の形式

```
<!ELEMENT 要素型名 内容モデル>
```

要素の内容モデルは次の４つのタイプになります。

EMPTY

内容をもたない要素である「空要素」であることを示します。例えば、HTML で、横線を指定する<hr>のようなタグは内容を持ちません。XML では、<hr></hr>と書くか、<hr />と書きます。２つの表記方法は同等です。

ANY

制限なし

要素内容

子供の要素を必ず含み、文字データを含まない。この時、内容モデルで子要素の出現順序や出現回数を規定します。

要素内容モデルの例

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT front (title, date, author+)>
<!ELEMENT body (head, (p | list | note)*, div2*)>
```

上の例で、","は出現順序（この順番）、"|"は選択を示します。?は 0 回または 1 回出現、+は 1 回以上出現、*は 0 回以上出現することを示し、これらの演算子がないときは正確に 1 回だけ出現します。

混在内容モデル

内容として子要素と文字データの両方をもつもの。子要素の型の制約はできますが、子要素の出現順序や出現頻度については制約ができません。

属性リスト宣言

属性は要素の開始タグの中でのみ指定でき、その要素に属性と属性値の対を関連付けます。属性定義の形式は次の図の通りです。

属性リスト宣言の形式

<!ATTLIST 要素型名 属性定義*>

属性定義：:=属性名　属性の型 デフォルト宣言

属性の型

XML での属性の型は、文字列型、トークン化型、列挙型の 3 種類です。

文字列型

CDATA で指定します。属性値に任意の文字列を記述できます。

トークン化型

ID、IDREF、IDREFS、ENTITY、ENTITIES、NMTOKEN、NMTOKENS の 7 種類があります。

ID

妥当な XML 文書では、ID の値は名前文字であり、XML 文書内で ID の名前は複数回現れてはなりません。要素を一意に特定できます。

IDREF、IDREFS

妥当な XML 文書では、名前文字であり、XML 文書内に存在する ID 属性の値とマッチしなければなりません。

ENTYTY、ENTITIES

妥当な XML 文書では、名前文字であり、DTD で宣言する解析対象外実体とマッチしなければならない。

NMTOKEN、NMTOKENS

妥当な XML 文書では、それぞれの値が「XML 基本仕様書」で定める生成規則にマッチしなければならない。

列挙型

属性リスト宣言で幾つかの値を宣言し、その中の一つを取ることができます。NOTATION 型と列挙型があります。NOTATION 型は DTD の記法宣言で宣言され、NOTATION 型の属性が付与された要素の解釈を特定します。列挙型は属性値を列挙します。

デフォルト宣言

属性宣言の中で、属性の指定が必須かどうかを規定します。

#REQUIRED

この属性は必須属性であることを意味します。

#IMPLIED

デフォルト値がないことを意味します。

(#FIXED)?デフォルト値

ここで指定する属性値の値が、デフォルト値です。#FIXED キーワードはその属性の値がデフォルト値と一致する必要があることを示しています。

属性の設定例

属性リスト宣言の例

```
<!ATTLIST termdef
         id   ID    #REQUIRED
         name CDATA #IMPLIED >
<!ATTLIST list
         type (bullets|ordered|glossay) "ordered" >
<!ATTLIST form
         method CDATA #FIXED "POST" >
```

上の例では、termdef 要素には ID 属性と nama 属性を設定しています。 また、list 要素には type 属性を設定しています。type 属性は、列挙型で bullets、ordered、glossary の 3 つの値のどれかを取ります。デフォルトは ordered です。 また、form 要素には method 属性が設定されていますが、この属性の値は POST に固定です。

また、属性の順序は特に意味がありません。また、何を属性にし、何を要素にするかは XML の応用仕様を設計する人の自由です。次の例の(a)、(b)の２つの表現方法は XML の基本仕様ではどちらも使えます。

例(a) paragraph 要素に属性を設定

```
<paragraph font-name="MS 明朝" font-size="12point">
段落の文字のフォントは明朝で、サイズは 12 ポイントです。
</paragraph>
```

例(b) paragraph 要素の子要素として文字スタイル要素を設定

```
<paragraph>
    <font-name>MS 明朝</font-name>
    <font-size>12point</font-size>
```

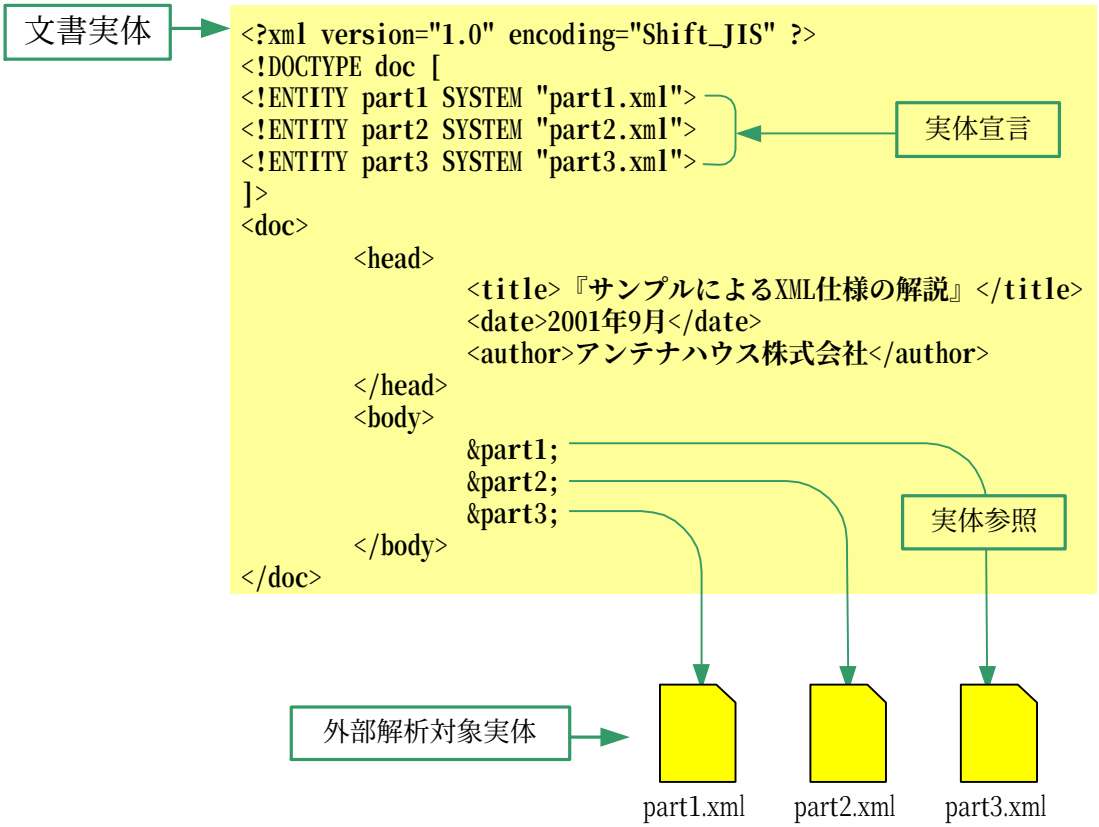
```
    段落の文字のフォントは明朝で、サイズは 12 ポイントです。
</paragraph>
```

実体宣言

XML では、文書の記憶単位を実体と言います。XML 文書は『文書実体』が必ずひとつあります。XML 文書全体が文書実体に含まれることもあります。文書実体以外の実体は、実体宣言で宣言して、実体参照で使います。次の図は、実体宣言とその使用例です。

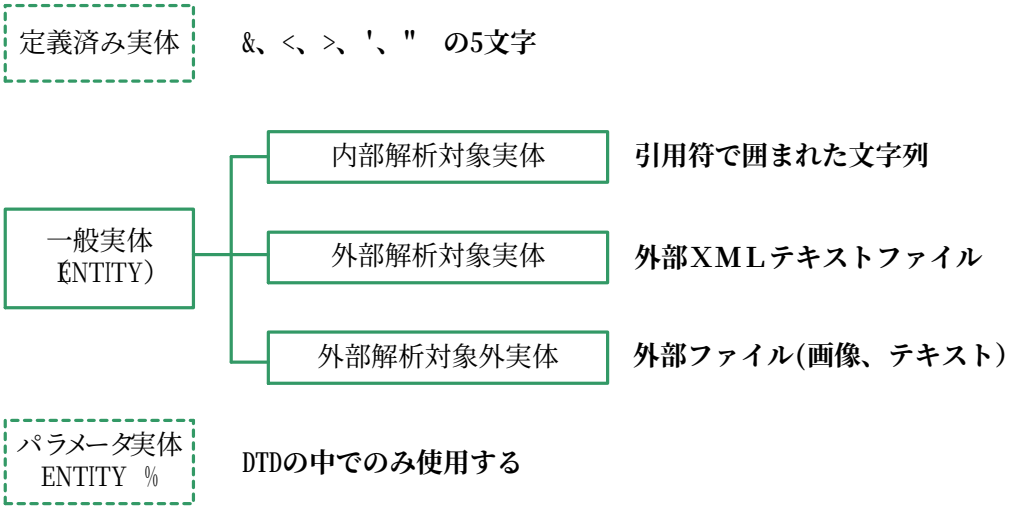
1. 文書の本文を 3 つの別ファイルに保存します。
2. 実体宣言(<!ENTITY)で、実体名(part1 など)とファイルのパス(part1.xml など)を対応付けます。
3. 実体参照(&part1 など)によって文書実体から参照します。

このように実体を使うことで XML 文書をモジュール化することができます。



実体の種類

実体には次図の種類があります。実体を大きく分けると一般実体とパラメータ実体に分かれます。パラメータ実体は DTD の中でのみ使用するもので、単に「実体」というと一般実体を指します。



定義済み実体

XML 文書中で実体を使うには DTD の実体宣言で定義しなければなりませんが、定義済み実体のみは DTD で定義することなく使うことができます。定義済み実体とは「amp、lt、gt、apos、quot」の 5 文字です。

内部実体と外部実体

実体宣言(<!ENTITY >の内部で与えられるものを内部実体と言います。内部実体は必ず解析対象実体でなければなりません。内部実体でないものは外部実体です。

外部解析対象実体と解析対象外実体

解析対象実体のうち、外部実体を宣言する場合は、システム識別子として URI を使います。さらに、記法宣言があるものが一般外部解析対象外実体です。通常、解析対象外実体は、画像ファイルなどテキスト以外の内容に使います。

内部実体

実体宣言<!ENTITY で実体の名前に続いて実体の値があるものが内部実体です。内部実体の値は置換テキストとして使われます。

内部実体宣言例

```
<!ENTITY Pub-Status "This is a pre-release of the specification.">
```

外部実体

実体宣言<!ENTITY で実体の名前に続いて、外部 ID（SYSTEM または PUBLIC URI）があるものが外部実体です。次は外部実体の宣言例です。

外部実体宣言例

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

NDATA のキーワードがあるものは解析対象外実体、それ以外は外部解析対象実体です。解析対象外実体は記法宣言で宣言されていないと妥当性検証でエラーになります。

画像ファイル

画像ファイルなどを解析対象外実体として定義して、XML 文書中で実体として参照するのは、基本的な方法です。しかし、HTML のように、要素の src 属性の値にファイルのパスを指定しておき、それをアプリケーションに渡してアプリケーションに処理を任せることもできます。

外部解析対象実体の文字符合化方式は、本体 XML 文書と異なっていても構いません。なお、妥当性検証パーサーは、外部解析対象実体を必ず取り込んで検証します。しかし、整形式検証パーサーでは、必ずしも読むことを義務付けられていません。但し、読まないことをアプリケーションに通知しなければなりません。

パラメータ実体

パラメータ実体は DTD の中だけで使うことができます。繰り返し現れる要素グループ、属性グループ、文字列などを置換するために使うものです。

パラメータ実体の例

```
<!ENTITY % block "p | figure | ul | ol | dl | table | program | pre | div | hidden
">
```

記法宣言（NOTATION）

記法とは解析対象外実体の形式、記法属性を持つ要素の形式、または処理命令の対象とするアプリケーションを特定する名前のことです。

DTD の中の記法宣言で記法の名前及び外部識別子を提供します。この名前は、外部実体宣言、属性リスト宣言、及び属性指定に用います。外部識別子は，与えられた記法のデータを処理できるソフトウェア（ヘルパアプリケーションなど）を探すために利用します。

記法宣言の例

```
<!NOTATION emf SYSTEM "image/emf">
```

実用的 XML

DTD は必要か？

XML は SGML のサブセットと良く言われます。しかし、SGML になかった整形式（Wellformed)という概念が新しく導入されたことで、非常に幅広く使えるようになりました。InternetExplorer を初めとして XML を処理するアプリケーションも整形式の XML 文書を処理できるようになっています。従って、昔のように DTD が必須という状況ではなくなってきました。

DTD の実例について

- SimpleDoc
- JepaX
- XHTML
- NewsML
- DocBook

XML の仕様(2)

名前空間(ネームスペース)

XML の基本仕様を拡張する補助的仕様としての名前空間について説明します。

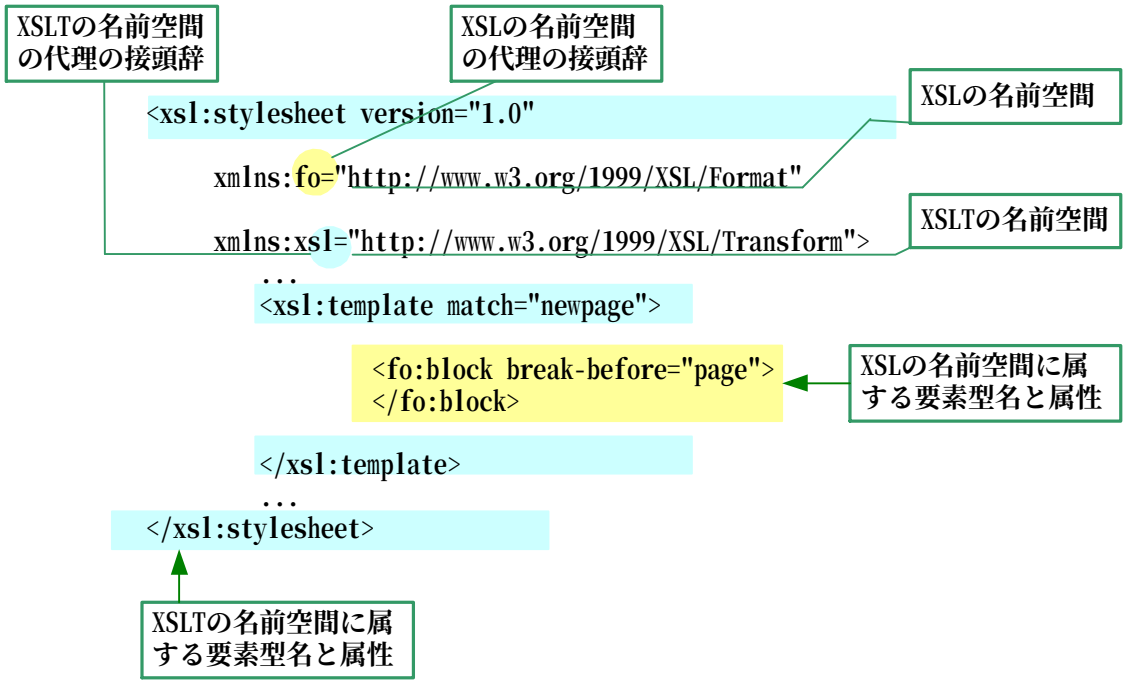
XML のマークアップ語彙(タグセット)は、色々なアプリケーションで汎用に使うのが望ましいのですが、ひとつの XML 文書で複数のタグセットを使おうとするとき、タグ名の競合という問題が起きます。この問題を解決するために策定されたものが「名前空間」仕様です。

XML 名前空間の定義

XML 名前空間とは、URI 参照によって識別される要素型名と属性名の集合です。URI 参照は全文字が一致するときに同一と見なします。そして、XML 名前空間の名前はひとつのコロンを含む修飾名として表現することもできます。コロンの前を名前空間接頭辞、コロンの後ろをローカルパートと言います。URI 参照は、XML で名前として使うことのできない文字を含む可能性がありますので、名前空間接頭辞を URI 参照の代理として使うわけです。

名前空間は、XML 文書中でその名前空間接頭辞の出現する前に宣言しなければなりません。宣言は、URI 参照と名前空間接頭辞を関係つけるもので、XML の属性の文法を使って宣言します。

次の例は、ひとつの XML 文書(スタイルシート）の中で２つの名前空間に属する要素型名を使用する例です。



名前空間の宣言

名前空間は、xmlns という属性または xmlns:という接頭辞をもつ属性を使って宣言します。

- 名前空間宣言用の属性名は、xmlns:NCName(XML の名前文字から:を除く文字列）か、もしくは、xmlns のいずれかとなります。
- NCName は名前空間接頭辞となります。
- 名前空間宣言の属性名が xmlns の時、この名前空間は宣言がなされている要素の中でのデフォルト名前空間となります(デフォルト名前空間については後述)。

- 名前空間宣言は XML 文書中で直接宣言するか DTD の中でデフォルトとして与えられます。

XML 文書で名前空間を宣言する例

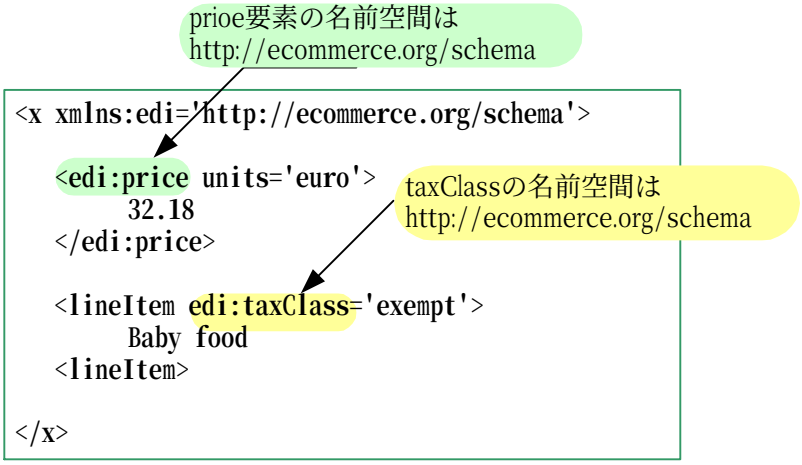
```
<x xmlns:edi="http://ecommerce.org/schema">
<!-- x 要素及びその内容に対して、edi 接頭辞は名前空間 http://ecommerce.org/schema に関係つけられます-->
</x>
```

属性値のデフォルト

DTD の属性リスト宣言で属性値のデフォルトを指定できます。この機能をつかって DTD でデフォルト属性として名前空間を宣言することもできます。しかし、DTD の内部サブセットなら良いですが、外部 DTD サブセットで与えられるとアプリケーションによってはそれを取得することができません。

修飾された名前

名前空間の仕様に準拠する XML 文書では、要素型名、属性名は次のように修飾された名前として与えられます。修飾された名前のうち、コロン（:）の前の部分を名前空間接頭辞といいますが、接頭辞は、それが使われる要素の開始タグか、あるいはその祖先の要素の中で宣言されている必要があります。



名前空間のスコープとデフォルト名前空間

名前空間宣言は、それが指定された要素とその要素の内部にある全ての要素に適用されます。次の例では、すべての要素が、HTML 名前空間に属することを明示的に示しています。

名前空間宣言は子供の要素にも適用される

```
<?xml version="1.0"?>
<html:html xmlns:html='http://www.w3.org//1999/xhtml'>
  <html:head>
    <html:title>名前空間宣言
    </html:title>
  </html:head>
  <html:body>
    <html:p>Moved to
      <html:a href='http://frob.com'>here.
      </html:a>
    </html:p>
  </html:body>
</html:html>
```

デフォルト名前空間は、それが宣言された要素（その要素が名前空間接頭辞を持たない場合）、及びその内部にあるすべての接頭辞をもたない要素に適用されます。デフォルト名前空間は、属性には適用されません。次の例では、ルート要素（文書要素）にデフォルト名前空間を宣言していますので、すべての要素が名前空間 HTML に存在します。

デフォルト名前空間宣言

```
<?xml version="1.0"?>
<html xmlns='http://www.w3.org//1999/xhtml'>
  <head>
    <title>デフォルト名前空間
    </title>
  </head>
  <body>
    <p>Moved to
      <a href='http://frob.com'>here.
      </a>
    </p>
  </body>
</html>
```

デフォルト名前空間が空の場合、デフォルト名前空間が存在しないことを意味します。次の例は、表のセルの中ではデフォルト名前空間が存在しないことを示しています。

デフォルト名前空間宣言を上書きする

```
<?xml version="1.0"?>
<html xmlns='http://www.w3.org//1999/xhtml'>
  <head>
    <title>デフォルト名前空間の上書き
    </title>
  </head>
  <body>
    <table>
    <tr>
    <td>Name</td><td>Origin</td><td>Description</td>
    </tr>
    <tr>
       <td><brandName xmlns="">Huntsman</brandName></td>
       <td><origin xmlns="">Bath, UK</origin></td>
       <td><details xmlns=""><class>Bitter</class><hop>Fuggles</hop>
           <pro>Wonderful hop, light alcohol, good summer beer</pro>
           <con>Fragile; excessive variance pub to pub</con>
           </details>
       </td>
    </tr>
    </table>
  </body>
</html>
```

コロン(:)の使用の注意

名前空間に対応する XML 文書では":"の使用が次のように制限されます。

- 要素型名、属性名にはコロンは０回または１回しか出現してはならない。
- 実体名、処理命令のターゲット、記法名にはコロンを含んではならない。
- ID、IDREF(S)、ENTITY(IES)、NOTATION 型であると宣言された属性値もコロンなしでなければならない（妥当性検証パーサーでしか検証できません）。

例えば、次のような例は、名前空間に対応しない場合は正しいですが、名前空間対応の XML 文書としては正しくありません。

名前空間に正しく対応しない処理命令

```
<?xml:stylesheet type="text/xsl" href="denpyo.xsl"?>
```

次のように修正する必要があります。

名前空間に正しく対応する処理命令

```
<?xml-stylesheet type="text/xsl" href="denpyo.xsl"?>
```

XML パス言語(XPath)

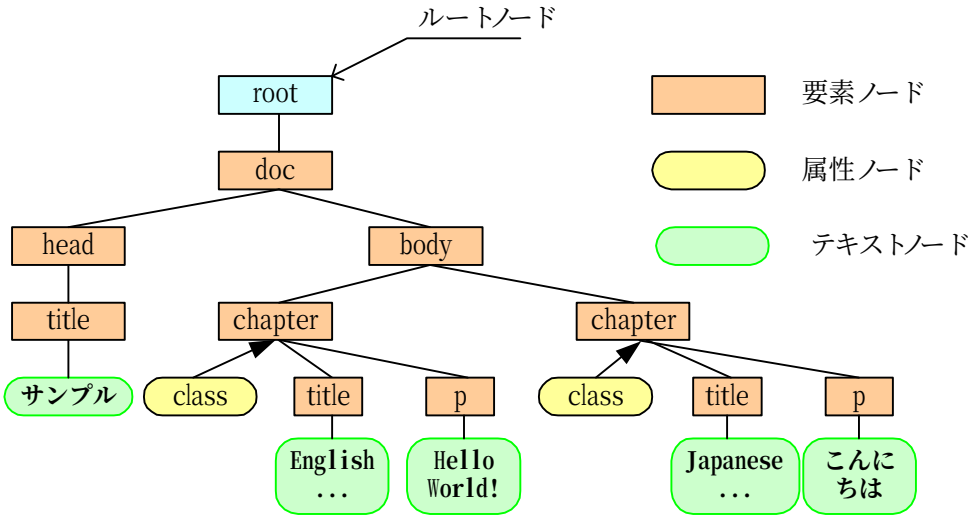
XMLPath 言語(XPath)Version1 について説明します。XPath は、XML 文書の一部分を指定するためのものです。XPath はあとで説明する XSLT、あるいは XML スキーマでも使用します。また、他にも、XPointer、XForms などの仕様で共通に利用することを目的に設計されています。

XPath のデータモデル

XPath では XML 文書をノードの樹（ツリー）でモデル化して表現します。次の例は、簡単な XML 文書を XPath のデータモデルによるノード・ツリーで表したものです。

サンプル XML 文書

```
<doc>
    <head>
        <title>サンプル</title>
    </head>
    <body>
        <chapter class="english">
            <title>English Greeting</title>
            <p>Hello World!</p>
        </chapter>
        <chapter class="Japanese">
            <title>Japanese Greeting</title>
            <p>こんにちは。</p>
        </chapter>
    </body>
</doc>
```



ノードの種類

XML 文書の中の要素、属性、テキスト、処理命令、コメントなどの意味のある項目を指し示す言葉としてノードという用語を使います。XML 文書はノードを別のノードに付け加えていくことによってノードのツリーで表現できます。XPath では次の 7 種類のノードが定義されています。

ルート・ノード

XML 文書をテキスト、マーク付けに分ける処理を行い、ツリー表現にした時の開始地点のこと。すべての XML 文書のツリーにはルート・ノードがあります。ルート・ノードは、文書のルート要素(文書要素）ではありません。ルート・ノードの周囲には、コメントや処理命令のノードが存在する可能性もあります。ルート要素のノードはルート・ノードの子孫となります。

要素ノード

XML 文書の要素毎にひとつの要素ノードがあります。要素ノードの子は、要素ノード、注釈ノード、処理命令ノード、テキストノードです。属性ノードは要素ノードの子にはなりません。

テキストノード

要素の中の文字データは、CDATA の内容もまとめてテキストノードとなります。

属性ノード

要素ノードに結び付けられている属性のセット。要素ノードは、要素の開始タグの中で明示的に指定されたもの、または、DTD においてデフォルト値付きで明示的に宣言された属性についてのみ属性ノードをもちます。なお、要素ノードは属性ノードの親となります。

名前空間ノード

要素に結び付けられる名前空間のセットを示すノード。

処理命令ノード

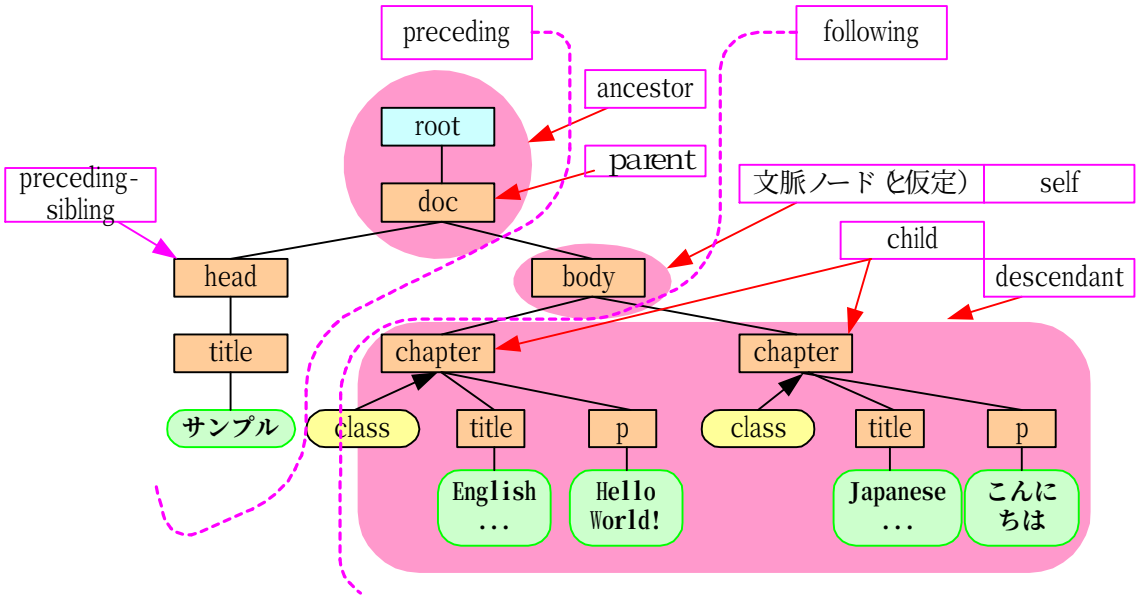
DTD の中の処理命令を除き、処理命令毎に一つの処理命令ノードがあります。

注釈(コメント)ノード

DTD の中の注釈を除き、注釈毎に一つの注釈ノードがあります。

ノードの親子、兄弟

XML 文書の中のノードにはすべて順序があります。この順序は、一般実体を展開した後の XML 文書の中でそのノードの最初の文字が出現する順序に一致します。ルート・ノードが最初です。要素についてはその開始タグの出現順になり、要素の属性ノードや名前空間ノードは、要素の子ノードよりも前に出現します。これを文書順と言います。文書順を逆にしたものが逆文書順です。



ノードの順序に関連する主な用語を次に挙げます。

文脈ノードまたはカレント・ノード（context node, current node）

着目しているノード

兄弟(sibling)

あるノードの中に幾つかのノードがある場合の、それらのノードの一つ。兄弟は出現順に順序があります。

親(parent)

現行のノードを含むノードです。ルートノード以外のノードには必ず一つだけ親があります。親は要素ノードかルートノードのどちらかです。

子(child)

あるノードの子は要素ノード、テキストノード、処理命令ノード、注釈ノードのどれかです。子ノードの子供を子孫と言います。

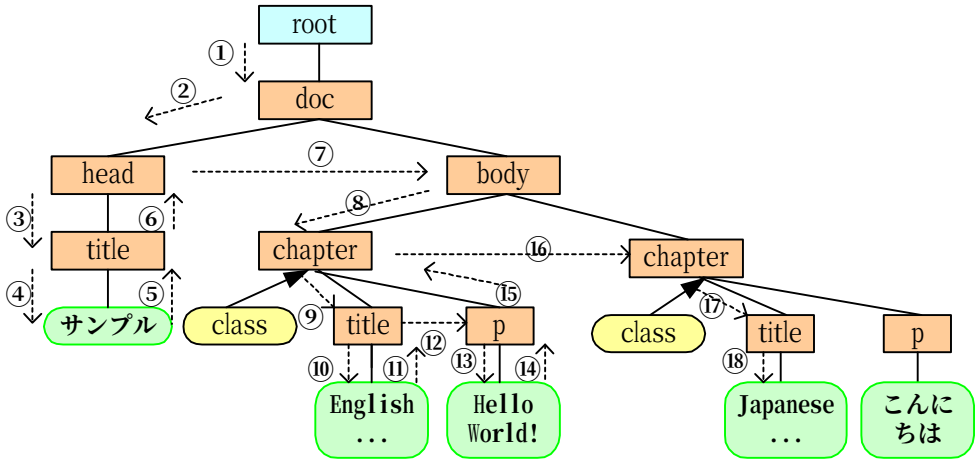
ツリー・ウオーク

ツリーの一番左からノードを枝の末端までたどり、ついで新しい枝に移動して末端までたどり、さらに新しい枝へとすべてのノードを移動していくことをツリー・ウオークといいます。

ツリー・ウオークの規則は次の通りです。

1. カレント・ノードが子ノードを持っていれば最初の子を選択します。
2. そうでない場合、カレント・ノードが後続の兄弟をもっていれば、次の兄弟を選択します。
3. そうでもない場合、最も近い祖先の次の後続の兄弟を選びます。

次の図のように、ツリー・ウオークは XML 文書を上から読んで行くときに開始タグ、終了タグの出現する順序と同じになります。



XPath の基本

XPath の中核は XML 文書ツリーの中でノードの集合などを指定するための式(expression)の文法を規定し、その意味や評価方法を決めた仕様です。

式は文脈(context)との関係で評価されます。文脈として与えられる情報は、context node、context position、context size、変数束縛の集合、関数ライブラリー、その式に対して有効な名前空間の集合などです。

評価結果は次のいづれかになります。

node-set

ノードの集合です。

boolean

ブール値。true、false のいづれかの値を取り得る真偽値です。

number

倍精度(64 ビット)浮動少数

string

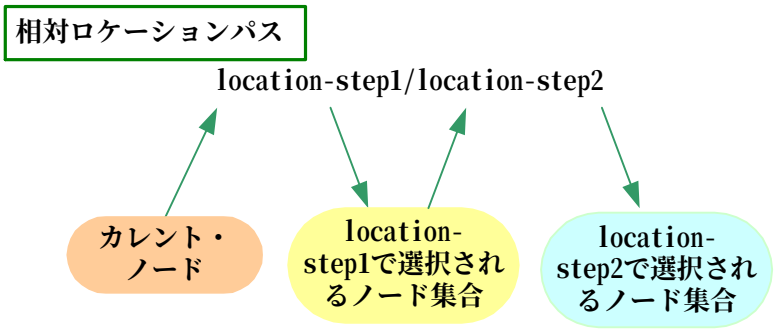
文字列

ロケーションパスとロケーションステップ

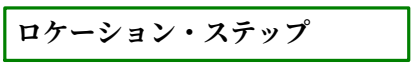
式の中の特別なものがロケーションパスです。ロケーションパスを評価した結果は、そのロケーションパスによって選択されるノードの集合となります。

ロケーションパスはロケーションステップを"/"で結合したものです。ロケーションステップは、文脈ノードからみて相対的に、指定された条件を満たすノード集合を返すものです。ロケーションステップを結合した場合、１つめのロケーションステップから返されるノード集合のそれぞれが２つめのロケーションステップの文脈ノードになります。従って、２つのロケーションステップを結合したものは、文脈ノードから２つのステップを経て辿ることのできるノードの集合となります。

ロケーションパスの先頭を"/"にすると絶対ロケーションパスとなり、これは最初の文脈ノードをルートノードにした場合に相当します。



ロケーションステップは、次の形をしています。



【形式】

軸の名前::　ノードテスト　述部(0個以上)

軸

そのロケーションステップによって選択されるノードの候補群と文脈ノードとの間の関連性を指定します。axis 名として指定できるのは、ancestor、ancestor-or-self、attribute、child、decendant、decentant-or-self、following、following-sibling、namespace、parent、preceding、preceding-sibling、self の 13 種類です。

ノードテスト

ノードテストにはノードの名前を指定します。もし、名前を指定しないで全ての要素ノードを指定したいときは、ワイルドカード"*"を使います。名前空間を限定して何でも良いことを示すには、接頭辞:*とします。ノードテストでは、node()、text()、comment()、processing-instruction()を使うこともできます。それぞれ、すべての型の全ノード、テキストノード、注釈ノード、処理命令ノードを指定します。

0 個以上の述部（複数指定可）

述部の形式は[]の中に式を記述します。述部の役割は式を使ってロケーションステップによって選択されたノードの集合を更に絞り込むことです。なお、式に付いては後述します。述部が複数指定されている場合は、順番にフィルターを掛けて絞り込んでいきます。

次の図にロケーションステップの指定例を示します。

【例】 述部なし　　【意味】

child::para	文脈ノードの子para要素を選択
child::*	文脈ノードの子要素すべてを選択
child::text()	文脈ノードの子テキスト要素を選択
child::node()	文脈ノードの子すべてを選択
attribute::href	文脈ノードのhref属性を選択
descendant::para	文脈ノードの子孫para要素を選択
ancestor::div	文脈ノードの祖先div要素を選択

述部あり

child::para[position()=1]	文脈ノードの最初の子para要素を選択
child::para[position()=last()]	文脈ノードの最後の子para要素を選択
child::para[position()>1]	文脈ノードの最初の子para要素以外の文脈ノードの子para要素を全て選択
child::para[attribute::type="warning"]	文脈ノードの子孫paraで、type属性にwarningという値をとるものすべてを選択する
child::chapter[child::title]	文脈ノードの子chapterで、1つ以上の子titleをもつものを選択
child::*[self::chapter or self::appendix]	文脈ノードのchapterおよび子appendixを選択

省略形

ロケーションステップは次の省略形を使えます。

省略形	完全形	説明
.	self::node()	文脈ノードを選択する
..	parent::node()	文脈ノードの親を選択する
name	child::name	child::はデフォルト軸。文脈ノードの子 name を選択する。
@name	attribute::name	文脈ノードの属性 name を選択する。
//	/descendant-or-self::node()/	

次は略形をもちいたロケーションパスの例です。

- para は、文脈ノードの子 para 要素を選択します。

- *は、文脈ノードの子である要素ノードすべてを選択します。
- text()は、文脈ノードの子であるテキストノードすべてを選択します。
- @name は、文脈ノードの name 属性を選択します。
- @*は、文脈ノードの属性すべてを選択します。
- para[1]は、文脈ノードの子 para 要素の最初のものを選択します。
- para[last()]は、文脈ノードの子 para 要素の最後のものを選択します。
- */para は、文脈ノードの孫である para 要素すべてを選択します。
- /doc/chapter[5]/section[2]は、doc の 5 番目の chapter 要素の 2 番目の section を選択します。
- chapter//para は、文脈ノードの子 chapter 要素の子孫 para 要素を選択します。
- //para は、ルートノードの子孫 para すべてを選択します。従って文脈ノードと同じ文書にある para 要素すべてを選択します。
- //olist/item は、文脈ノードと同じ文書にある item 要素で、親 olist をもつものすべてを選択します。
- .//para は、文脈ノードの子孫 para 要素を選択します。
- ../@lang は、文脈ノードの親の lang 属性を選択します。
- para[@type="warning"]は、文脈ノードの子 para のうち type 属性に warning という値をとるものすべてを選択します。
- para[@type="warning"][5]は、文脈ノードの子 para で、type 属性に warning という値をとるもののうち 5 番目のものを選択します。
- para[5][@type="warning"]は、文脈ノードの 5 番目の子 para が type 属性に warning という値をとるならば、それを選択します。
- chapter[title="Introduction"]は、文脈ノードの子 chapter で、文字列値が Introduction に等しい子 title を 1 個以上もつものを選択します。
- chapter[title]は、文脈ノードの子 chapter で、子 title を 1 個以上もつものを選択します。
- employee[@secretary and @assistant]は、文脈ノードの子 employee で、secretary 属性と assistant 属性との両方をもつものすべてを選択します。

XSLT での制約

ロケーションパスの中で、XSLT のテンプレートルールにおいて、適用先ノードを識別するのに使える軸は、省略形か、child または attribute のみです。descendant-or-self 基準点を使うことはできませんが、//を使うことはできます。

コア関数ライブラリー

XPath を実装するとき、式を評価するのに使う関数として必ず用意しなければならない関数が定義されています。次の表には、ノードセットを操作する関数だけを挙げます。これ以外に、文字列関数、ブール値関数、数値関数が定義されています。

種類	関数のプロトタイプ	説明
ノードセット関数	number last()	文脈サイズに等しい数値を返す。
	number position()	文脈ポジションに等しい数値を返す。
	number count(node-set)	引数 node-set のノード数を返す。
	node-set id(object)	引数のリストの中のどれかに等しい一意的 ID をもつものを包含したノードセットを返す。
	string local-name(node-set?)	引数のノードセットの中で最初のノードのローカル名を返す。
	string namespace-uri(node-set?)	引数のノードセットの中で最初のノードの名前空間 URI を返す。
	string name(node-set?)	引数のノードセットの中で最初のノードの接頭辞付き名前を返す。

XSLT の追加関数

XSLT 仕様では、XPath で定義するコア関数の他にも独自関数が追加されています。

DOM

DOM とはなにか

文書オブジェクトモデル（DOM)は、はいままで説明してきました XML の仕様とは役割が少し異なります。いままで説明してきました XML 仕様はいづれも XML という情報表現の新しい枠組みを定めるものです。しかし、DOM は XML 文書オブジェクトを処理するプログラムのための標準的なインターフェイスを定めるものです。

例えば、XML パーサーで XML 文書を読み、それを、エディタで編集するアプリケーションを想定します。XML パーサーが作成した文書オブジェクトをエディタが操作するという役割分担が考えられますが、この文書オブジェクトを DOM に準拠して作成することによって、DOM オブジェクトを作成できる XML パーサーをエディタとは独立に作成し、あるいは他のベンダの XML パーサーを使うこともようになります。

XML 文書を読み、処理するためのシステム・インテグレーションにあたって、XML 文書を DOM に対応する XML パーサーを使って読めば、XML 文書の読み込み時の符合化方式の対応、あるいは正しい XML 文書（Wellformed)かどうかのチェックなど、めんどうな処理に気を使わなくてもよくなります。

DOM の特徴

DOM には次のような特徴があります。

- DOM は、XML 文書のツリーをメモリ上に構築します。従って、ノードの出現順序の制約がなく、DOM ツリーを移動することで複雑な XML 文書のアクセスも簡単にできます。
- 一方、大きな XML 文書の場合は大量のメモリを使うこと、あるいは、文書の読み込みが終わってツリーの構築が完了しないと次の処理に入れないという欠点もあります。
- DOM では、文書オブジェクトの内容の取り出しだけでなく内容の更新、ノードの追加、削除などの処理もできますので、XML 文書のダイナミックな処理が可能です。

DOM の仕様の種類

W3C が定めている DOM の仕様は、DOM の仕様書として単独で標準化を図っているものと、MathML、SVG など XML の応用仕様の一部として標準化されているものがあります。現在、次の構成となっています。

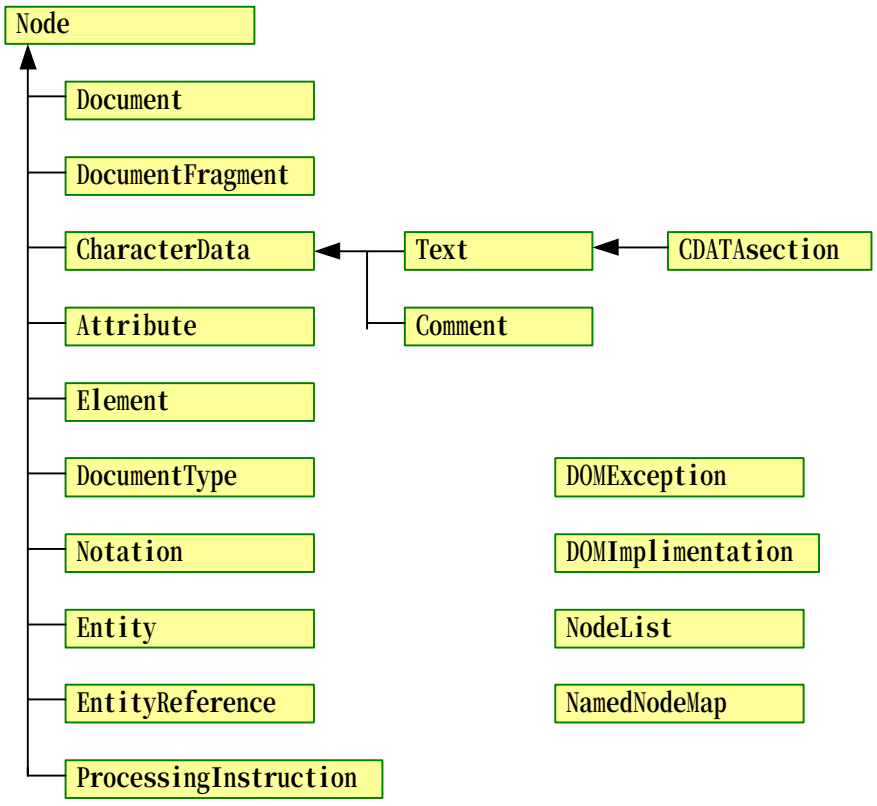
レベル	内容	ステータス
DOM レベル 1 コア	文書オブジェクトにアクセスしたり操作するための最小限のオブジェクトやインターフェイスを定義する。	W3C 勧告
DOM レベル 1HTML	HTML 文書に特有のオブジェクト、メソッドを記述するために第 1 水準コアを拡張したもの。	W3C 勧告
DOM レベル 2 コア	DOM レベル 1 コアに新しいインターフェイスと例外が追加された。	W3C 勧告
DOM レベル 2 ビュー	ドキュメントのビューへの参照	W3C 勧告
DOM レベル 2 イベント	一般的なイベントシステムを設計し、標準的なイベント・モジュールを提供する	W3C 勧告
DOM レベル 2 スタイル	スタイルシート（LINK、STYLE、スタイルシート処理命令）、CSS のための基本的インターフェイス	W3C 勧告
DOM レベル 2 トラバーサルとレンジ	ツリーウオーク、ノードの繰り返し、ノードフィルター、範囲指定と処理のインターフェイス	W3C 勧告
DOM レベル 3 コア	レベル 2 コアの改訂版	W3C ワーキング・ドラフト
DOM レベル 3 抽象的スキーマと Load,Save 仕様	DTD、XMLSchema の取得と変更のインターフェイス、Load と Save インターフェイス	W3C ワーキング・ドラフト
DOM レベル 3 イベント	レベル 2 イベントの改訂版	W3C ワーキング・ドラフト
DOM レベル 3XPath	XPath の式を DOM で使えるようにするための仕様。	W3C ワーキング・ドラフト(2001/8/30)
MathML V2 のための DOM	MathML 文書にアクセスするための DOM 拡張	W3C 勧告
SVG V1 のための DOM	SVG 文書にアクセスするための DOM 拡張	W3C 勧告(2001/9/4)
SMIL アニメーションのための DOM	SMIL アニメーション文書にアクセスするための DOM	W3C 勧告(2001/9/4)
SYMM のための DOM	SMIL 言語と SMIL モジュールの次期バージョンのための DOM	W3C ワーキング・ドラフト

DOM のインターフェイス

W3C が定める DOM インターフェイスは、CORBA の IDL(インターフェイス定義言語）で記述された抽象的なものです。W3C の仕様書では、付録として JAVAScript、ECMAScript の 2 種類の言語バインディングを規定しています。しかし、C、C++などの言語対応については W3C 仕様書では規定しておらず、ベンダ依存となっています。例えば、Microsoft の XML パーサーの DOM は、COM の仕様に基づくインターフェイスとなっていますので、MSXML パーサーの DOM インターフェイスを使って作ったプログラムをそのまま他のベンダの DOM インターフェイスをもつ XML パーサーに適用することはできません。しかし、まったく XML 文書オブジェクトへのアクセス方法が標準化されていない場合と比較すれば、例えば、XML パーサーを別のベンダに変更したとして、アプリケーションの XML 文書アクセスの変更に要する工数は遥かに少なくなると言えるでしょう。

分類	インターフェイス	説明
DOM レベル１コア基本インターフェイス	Exception DOMException	エラーコード
	interface DOMImplimentation	機能のサポート有無を返す
	interface DocumentFragment : Node	XML 文書ツリーの一部
	interface Document : Node	XML,HTML 文書全体
	interface Node	XML 文書ツリーの単一のノード
	interface NodeList	ノードの順序付き集合
	interface NamedNodeMap	名前でアクセスできるノードの集合
	interface CharacterData : Node	DOM の中の文字データにアクセスするための属性、メソッドの集合
	interface Attr : Node	要素オブジェクトの属性
	interface Element : Node	要素オブジェクト
	interface Text : CharacterData	要素、または属性の中のテキストを表わす
	interface Comment : CharacterData	注釈の内容を表わす
DOM レベル１コア拡張インターフェイス	interface CDATAsection : Text	CDATA セクション
	interface DocumentType : Node	文書型、実体、ノーテーション
	interface Notation : Node	ノーテーション
	interface Entity : Node	解析対象、非解析対象実体を表現する
	interface EntityReference : Node	実体参照
	interface ProcessionInstruction : Node	処理命令

DOM インターフェイスの継承関係は次の図のとおりで Node オブジェクトが最も中核になっています。



SAX

SAX（Simple API for XML)は、XML パーサーのインターフェイスとしては DOM とならんで普及しているものですが、DOM とは違ってユーザ・グループにおいて提唱されたものです。オブジェクトの出現に従ってイベントを発生させて、それをアプリケーションが取り出すという簡単なインターフェイスです。DOM がメモリ上にツリーを構築するのと比べてメモリの使用量が少ないと言うメリットがあります。しかし、ドキュメントを更新するインターフェイスはありません。

SAX は 2000 年 5 月に SAX 2.0 として公開されています。しかし、SAX2 に関するドキュメントは存在しません。次のリストは SAX2 の実装リストの一部です。

SAX2/Java

SAX2/Java Parsers and Drivers

- Apache XML Project's Xerces Java パーサー。Xerces はネイティブの SAX2 driver を含んでいます。
- David Brownell の SAX2 XML Utilities。このパッケージは、SAX2 パーサーを幾つか含んでいます。SAX2 DOM パーサー、および AElfred の改訂版。
- Michael Kay の SAXON。SAXON は、AElfred の修正版で SAX2 準拠のものを含んでいます。

SAX2/C++ and SAX2/COM

C++と COM の世界では、SAX インターフェイスはひとつに整理されていません。つまり、すべての実装で共有できる#include が存在しないということです。結果として、以下のソフトウエアで使われている SAX 的なインターフェイスは単にプロトタイプとして扱うべきであるということです。

SAX2/C++ プロトタイプ実装

- Microsoft の XML パーサーは、SAX をベースとする COM インターフェイスを実装しています。
- Apache XML プロジェクトの Xerces C++ パーサーである Xerces/C++は、SAX ベースの C++インターフェイスを実装しています。

XML 言語記述の新しい方法

DTD の問題点

XML の基本仕様で定義されている DTD のベースは SGML の DTD[4]です。SGML はもともとが文書を扱うための仕様として設計されたことも理由と考えられますが、DTD では機能的に弱い部分が多く、次のような様々な問題点が指摘されています。

- DTD のマーク付け宣言は XML の文書インスタンスとは異なる文法で記述するもので、DTD ファイル単独では XML 文書ではありません。
- DTD では要素のデータ型は#PCDATA しかありません。また、属性にはいくつかのデータ型が定義されていますが、しかし、例えば自然数、日付などの要素型は組み込みで定義されていません。このように伝票などのデータ志向の XML 文書を扱うにはデータ型の種類が余りにも少ないという問題があります。
- 要素の出現回数の制限機能が弱い。DTD では要素の出現回数を 0 回または 1 回、0 回以上無制限、1 回以上無制限という制限しかできず、X 回以上 Y 回以下というような制限ができません。
- 名前空間を扱えない。XML の名前空間に対応する文書については DTD ベースの妥当性検証を行うことができません。

そこで、DTD に変わる新しい XML 文書の構造を記述する方法がいろいろ提案されています。その中で、最も有力なものが、次に説明する W3C の「XML スキーマ」仕様です。

[4]但し、SGML の DTD と XML の DTD はかなり仕様が違います。

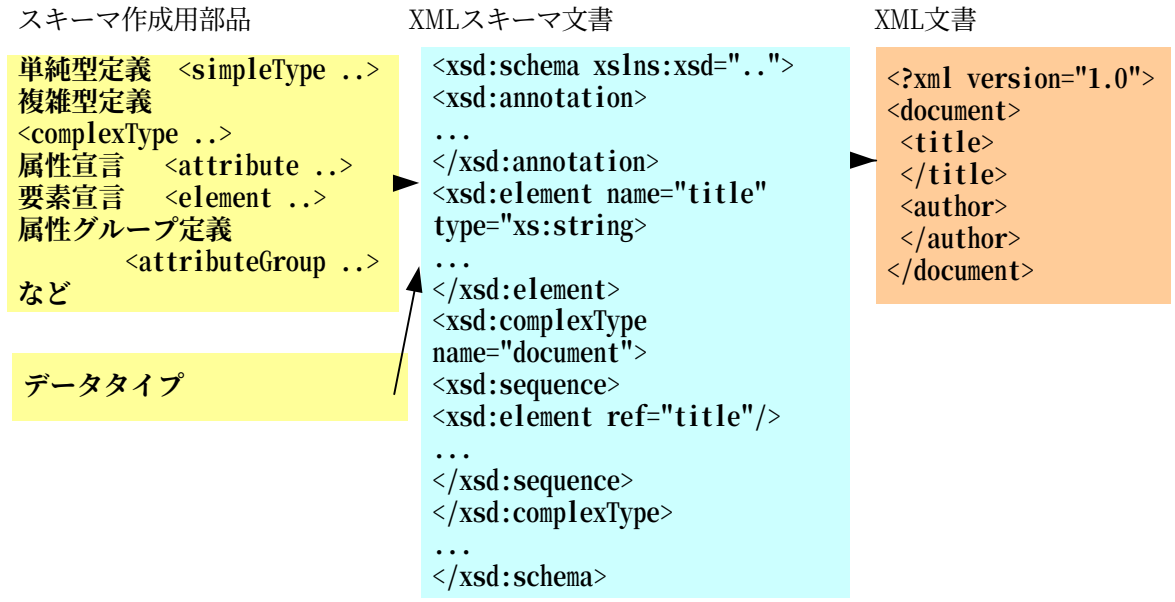
他に、日本から提唱されている「Relax」あるいは、JamesClark の提唱する「TREX」などがあります。

XML スキーマ（XML Schema）

XML スキーマは、1999 年の初頭に要求仕様が公開され、その後、約 2 年にわたる開発期間を経て、2001 年 5 月に W3C の勧告となりました。

XML スキーマ仕様では、基本的なスキーマ作成用部品とそのデータモデルを決めています。このスキーマ作成用部品を使うことにより、XML 文書の構文設計者は XML スキーマ文書を記述することができます。XML スキーマ文書は、DTD に相当するものです。

アプリケーションまたはユーザが作成する XML のデータまたは文書は、インスタンスという言葉で指定されることが多いですが、この XML スキーマ文書で定めた構文に従って作成されることになります。スキーマ作成用部品と XML スキーマ文書と XML 文書(インスタンス)の関係は次の図のようになります。



スキーマ作成用部品

スキーマ作成用部品とは XML スキーマの抽象的な枠組みを構築するためのブロックです。次の部品があります。

ブロック	使用例
スキーマ自身	**<xs:schema** xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.com/example"> . . . </xs:schema>
単純型定義(Complex Type Definition)	**<xs:simpleType** name="farenheitWaterTemp"> <xs:restriction base="xs:number"> <xs:fractionDigits value="2"/> <xs:minExclusive value="0.00"/> <xs:maxExclusive value="100.00"/> </xs:restriction> </xs:simpleType>

ブロック	使用例
複雑型定義(Symple Type Definition)	**<xs:complexType** name="PurchaseOrderType"> <xs:sequence> <xs:element name="shipTo" type="USAddress"/> <xs:element name="billTo" type="USAddress"/> <xs:element ref="comment" minOccurs="0"/> <xs:element name="items" type="Items"/> </xs:sequence> <xs:attribute name="orderDate" type="xs:date"/> </xs:complexType>
要素宣言(Element Declaration)	**<xs:element** name="PurchaseOrder" type="PurchaseOrderType"/> <xs:element name="gift"> <xs:complexType> <xs:sequence> <xs:element name="birthday" type="xs:date"/> <xs:element ref="PurchaseOrder"/> </xs:sequence> </xs:complexType> </xs:element>
属性宣言(Attribute Declaration)	**<xs:attribute** name="age" type="xs:positiveInteger" use="required"/>
属性グループ定義(Attribute Group Definition)	**<xs:attributeGroup** name="myAttrGroup"> <xs:attribute . . ./> . . . </xs:attributeGroup> <xs:complexType name="myelement"> . . . <xs:attributeGroup ref="myAttrGroup"/> </xs:complexType>
識別制約定義(Identity-constraint Definition)	**<xs:key** name="fullName"> <xs:selector xpath=".//person"/> <xs:field xpath="forename"/> <xs:field xpath="surname"/> </xs:key> **<xs:keyref** name="personRef" refer="fullName"> <xs:selector xpath=".//personPointer"/> <xs:field xpath="@first"/> <xs:field xpath="@last"/> </xs:keyref> **<xs:unique** name="nearlyID"> <xs:selector xpath=".//*"/> <xs:field xpath="@id"/> </xs:unique>

ブロック	使用例
モデルグループ定義(Model Group Definition)	**<xs:group** name="myModelGroup"> <xs:sequence> <xs:element ref="someThing"/> . . . </xs:sequence> </xs:group> <xs:complexType name="trivial"> <xs:group ref="myModelGroup"/> <xs:attribute .../> </xs:complexType> <xs:complexType name="moreSo"> <xs:choice> <xs:element ref="anotherThing"/> <xs:group ref="myModelGroup"/> </xs:choice> <xs:attribute .../> </xs:complexType>
記法宣言(Notation Declaration)	**<xs:notation** name="jpeg" public="image/jpeg" system="viewer.exe">
注釈（Annotation)	**<xs:annotation>** <xs:documentation>A type for experts only</xs:documentation> <xs:appinfo> <fn:specialHandling>checkForPrimes</fn:specialHandling> </xs:appinfo> </xs:annotation>
モデルグループ(Model Group)	**<xs:all>** <xs:element ref="cats"/> <xs:element ref="dogs"/> </xs:all> **<xs:sequence>** **<xs:choice>** <xs:element ref="left"/> <xs:element ref="right"/> </xs:choice> <xs:element ref="landmark"/> </xs:sequence>
分子(Particle)	<xs:element ref="egg" minOccurs="12" maxOccurs="12"/> <xs:group ref="omelette" minOccurs="0"/> <xs:any maxOccurs="unbounded"/>
ワイルドカード(Wild Card)	**<xs:any** processContents="skip"/> <xs:any namespace="##other" processContents="lax"/> <xs:any namespace="http://www.w3.org/1999/XSL/Transform"/> <xs:any namespace="##targetNamespace"/> <xs:anyAttribute namespace="http://www.w3.org/XML/1998/ namespace"/>
属性使用(Attribute Use)	<xs:complexType> . . . <xs:attribute ref="xml:lang" use="required"/> <xs:attribute ref="xml:space" default="preserve"/> <xs:attribute name="version" type="xs:number" fixed="1.0"/> </xs:complexType>

XML スキーマの基本用語

複合型（complexType 要素）

複合型は、XML スキーマ中で complexType 要素を使って定義します。定義には内容として要素宣言や、要素の参照、属性宣言をもつことができます。

単純型（SimpleType 要素）

数字、文字列のみを含むが子要素はもたない要素及びすべての属性。単純型は、XML スキーマ中で定義するか、または、XML スキーマ定義言語に組み込みの単純型として定義されています。

組み込み済み単純型

string,normalizedString,token,byte,unsignedByte,base64Binary,hexBinary,integer,positiveInteger, negativeInteger,nonPositiveInteger,int,unsigneInt,long,unsigedLong,short,unsignedShort,decimal,float, double,boolean,time,dateTime,duration,date,gMonth,gYear,gYearMonth,gDay,gMonthDay,Name, QName,NCName,anyURI,language,ID,IDREF,IDREFS,ENTITY,ENTITIES,NONATION, NMTOKEN,NMTOKENS が定義されています。

新しい単純型（SimpleType 要素）

組み込みの単純型から制限という手法で派生型として定義できます。さらに、ファセット (pattern、enumeration など）と正規表現を使ってより複雑・巧妙な派生型の定義もできます。

派生型

既存の型(基底型)から新しい型を作り出します。派生の方法には制限、拡張があります。例えば、string(文字列)をもつ基底型を制限して、リスト型にするようなことができます。

定義

XML スキーマの中で complexType、SimpleType 要素を使って新しい型をつくります。

混在内容を持つ要素の定義

子要素と文字データが並んで出現する要素は混在内容をもつといいます。混在内容をもつ要素は、copmplexType 要素に mixed='true'属性を指定することで定義できます。XML1.0 の混在モデル（DTD)では、文書中に出現する子要素と順番と数を制限することができないのに対して、XML スキーマでは子要素の順番や数を厳密に指定できます。

宣言

特定の名前と型をもった要素や属性が XML 文書中に出現できるようにします。要素は element 要素で宣言し、属性は atribute 要素によって宣言します。

参照

XML スキーマの中では、他で宣言されている既存の要素を参照できます。

グローバルな要素と属性

XML スキーマのルート要素は schema 要素です。schema 要素の子供として宣言された要素と属性をグローバルな要素、属性と言います。グローバル要素と属性は、他の(グローバルでない）要素、属性から何回でも参照できます。しかし、グローバル宣言から他の要素・属性を参照することはできません。またグローバル宣言では、type 属性を使って他の型を指定する必要があります。グローバル宣言では出現回数を制限する属性をもたせることはできません。

要素及び属性の出現回数

要素宣言の中で要素の出現回数を制限できます。XML の仕様では、属性はもともと 0 回または 1 回出現できるものですが、XML スキーマの属性宣言においてその出現回数を指定できます。

ファセット

length,minLength,maxLength,pattern,enumerarion,whiteSpace,maxInclusive,maxExclusive, minInclusive,minExclusive,totalDigital,fractionDigital が定義されています。XML スキーマ文書中で組み紺単純型とファセットを使って新しい型を派生させることができます。

XML スキーマ文書の例

次の XML 文書は、注文書を処理するための XML スキーマ文書の例です。

注文書スキーマ[5]

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

 <xsd:element name="comment" type="xsd:string"/>

 <xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
   <xsd:element name="shipTo" type="USAddress"/>
   <xsd:element name="billTo" type="USAddress"/>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
     fixed="US"/>
 </xsd:complexType>

 <xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity">
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice"  type="xsd:decimal"/>
      <xsd:element ref="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
```

[5]W3C「XML Schema Part0:Premier」より

```
  </xsd:sequence>
 </xsd:complexType>

 <!-- Stock Keeping Unit, a code for identifying products -->
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

XML 文書の例

次は、上述の XML スキーマ文書に基づいて、XML で表わした注文書の例（XML 文書インスタンス)です。

注文書 XML 文書[6]

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

XML スキーマ・パーサー

XML 文書が XML スキーマ文書に適合しているかどうかを検証するには、スキーマ検証パーサーが必要になります。既に、欧米では W3C の勧告版 XML スキーマをサポートする XML スキーマ・パーサーが出始めています。スキーマ検証パーサーは、まだ普及しているとは言いがたいですが、Microsoft が MSXML4 において XML スキーマに対応作業を行っていますので、完成が待たれます。現在は、テクニカル・プレビュー版が公開されています。

[6]W3C「XML Schema Part0:Premier」より

- XSV, an Open Source XML Schema Validator, web-form アクセス機能付き
  University of Edinburgh/W3C (beta) (update Jul 06 2001(http://lists.w3.org/Archives/Public/xmlschema-dev/2001Jul/0067.html))
- Microsoft XML Parser (MSXML) 4.0 B2 supports XML Schema validation and type discovery with both DOM and SAX. It is available free from the MSDN XML site(http://msdn.microsoft.com/xml)
  詳細な場所： (http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/MSDN-FILES/027/001/677/msdncompositedoc.xml).
- xerces-j 1.4.1 released(http://lists.w3.org/Archives/Public/xmlschema-dev/2001Jun/0171.html)

  Xerces-J is a validating XML parser which has support for the XML Schema Recommendation.
- Xerces-C 1.5.0 is now available(http://lists.w3.org/Archives/Public/xmlschema-dev/2001Jun/0099.html)

  This is to announce the much awaited release Xerces C(http://xml.apache.org/xerces-c/index.html) 1.5.0 which brings Schema Subset Support to Xerces-C!
- Sun's XML Datatypes Library(http://lists.w3.org/Archives/Public/xmlschema-dev/2001May/0008.html)

  XML Datatypes Library(http://www.sun.com/xml) is a Java implementation of W3C XML Schema Part 2. It can be used from any Java code to
  - validate strings with datatypes
  - convert strings into Java objects
- Commercial XML Schema (and others) aware streaming validator(http://www.extensibility.com/solutions/xml_validate/index.htm) from TIBCO Extensibility;

Relax(Regular Language Desription for XML)

Relax は、日本の INSTAC XML SUBWG が制定した仕様です。XML Schema は DTD が扱う範囲を大きく超えた仕様なのに対して、Relax は DTD 相当の機能と XML Schema のデータ型を XML 構文で表現していくつかの拡張を行ったものです。XML Schema と比べて、実装が簡単なのが特徴とされています。

XML の仕様(3)

スタイルシートとは

スタイルシートはなぜ必要？

XML 文書は、主に論理的構造に沿って作成します。文書のレイアウト指定は XML 文書のツリーからは分離されています。このことによって、文書の内容をいろいろな目的に使うことが簡単にできる、すなわち「**ワンソース・マルチユース**」に最適という XML 文書の利点が生れます。しかし、このようなタグと本文が複雑に交錯した XML 文書は人間には理解しにくいです。XML 文書を表示・印刷するためのプログラムを使って人間が理解しやすいように整形する必要があります。

HTML の場合は、H1、H2、P、などの使用可能なタグ・セットと各タグの意味合いが決まっています。例えば H1 タグは一番大きな見出しを意味します。H1 タグの内容をブラウザが表示上する際も、大見出しの内容ですので一番大きな文字として表示することになります。ブラウザは、HTML 専用の表示・組版ソフトということもできるわけです。

一方、XML は要素名をタグセットの設計者が自由に決定することができます。要素型名の種類(タグの名前)は無限となります。要素型名を見てもその要素の内容を大きな文字で表示するべきか、あるいはどこで改行を入れるべきかなど、画面に表示したり、紙に印刷する際のレイアウト指定の情報を得ることができません。このため XML の表示・印刷プログラムは、XML 文書が与えられても、それだけでは、要素の内容を適切に整形して表示・印刷することができません。

このような事情から、XML を表示したり印刷したりする技術は簡単ではありません。いままでは、XML を表示・印刷するアプリケーションは高価であったり、使い難いという問題がありました。しかし、XML をスタイルシートを使って表示・印刷する技術が、ここ１年位の間に実用化の段階に入ってきました。いまや、誰でも簡単に XML 文書を作成し、表示・印刷することが可能になったといっても過言ではありません。

XML を様々な形式で表示・印刷するためのキーワードが「スタイルシート」です。

スタイルシートという言葉は XML 以前から、ワープロや DTP ソフトでは使われてきました。特に Microsoft Word がスタイルシートという言葉を、段落などの表示書式を設定するために使っていますので、スタイルシートという言葉を知っている人は多いと思います。しかし、XML の世界でのスタイルシートという言葉の意味するところは、ワープロや DTP とは少し異なります。

また、XML のスタイルシートという言葉についても、1998 年から現在までの間に、だんだん仕様の確立が進んでいますが、仕様のドラフトが提案されてから勧告に至るまでに、紆余曲折があり、その影響を受けて、スタイルシートという言葉は非常に混乱しています。詳細は、付録をご参照ください。

次に、現在の XML のスタイルシート関連の仕様について、実例を交えながら最新状況を説明してみたいと思います。

スタイルシート仕様の種類

XML に関係する仕様で、スタイルシートと呼ばれる仕様、またはスタイルシートに関係する仕様には次の３つがあります。いずれも、World Wide Web Consortium（W3C）が制定しているものです。

CSS 仕様

「Cascading Style Sheets」の略。1998 年 5 月にレベル 2 の仕様が勧告になっています。

XSLT 仕様

「XSL Transformations」の略。1999 年 11 月にバージョン 1.0 が勧告になっています。同時に勧告になりました「XML Path Language」とセットで使われます。

XSL 仕様

「Extensible Stylesheet Language」(以下、混乱を避けるため、XSL-FO 仕様と略記します。）の略。2000 年 11 月に勧告候補となりました。現在、勧告候補の段階です。

XML を Web で表示する方法(1)--CSS

CSS とは

CSS は、HTML や XML などの Web ドキュメントに簡単に表示書式を付加するための仕様です。文書の内容とその表示のためのスタイルを分離して、Web の作成とメンテナンスを簡略化することがその狙いとされています。

CSS の仕様は、レベル 1 が 1996 年 12 月に W3C の勧告になりました。1996 年と言いますと、まだ XML 仕様も決まっていないころですので、HTML を表示することのみを想定しています。その後、「Cascading Style Sheets, level2 CSS2 Specification」(CSS2 仕様)が 1998 年 5 月に W3C 勧告となっています。現在は、レベル 3 仕様のドラフトが発表されています。

CSS では、セレクタによって要素または要素のパターンを指定します。表示書式は {} で囲まれた宣言ブロックの内部にプロパティとその値という形式で指定します。セレクタと宣言ブロックは半角空白で区切ります。

CSS スタイルシートの例

```
/* for top document title */
h1.doctitle {font-size: 20pt;
             font-weight: 500;
             background-color: rgb(51,153,153);
             color:rgb(255,255,255);
             text-align:center;
             margin-left:20%;
             margin-right:20%;}

/* any documents title */
h1.subtitle {font-size: 20pt;
             font-weight: 900;
             color: #000000;
             text-decoration: underline; }

h2 {font-size: 14pt;
    color: #000000;
    font-weight: bold;
    line-height: 15pt;
    border-style:double;
    border-width: 3px;
    padding:3px;
    background-color:rgb(222,252,182);
    margin:30px 0px 20px 0px;}

h3 {font-size: 12pt;
    font-weight: 600;
    color: #000000;
    line-height: 11pt;
    margin:20px 0px 15px 0px;}

body {background-color:rgb(249, 254,226);
      font-size: 10pt; }

strong {color: #a8fbff;
        font-size: 8pt; }
```

上の例では、h1.doctitle、h1.subtitle、h2、h3、body、strong がセレクタにあたります。{} で囲った宣言ブロックの内部が表示書式です。

セレクタは XML の文書ツリーのどの要素に、表示書式を適用するかを決定するものです。上の例のような簡単な要素名単独指定から、複雑な文脈指定まで可能です。

CSS2 の仕様書には、表示・印刷媒体としてブラウザのみではなく、紙媒体、および聴覚媒体も想定していると書かれています。しかし、実際のプロパティははほとんどがブラウザを想定しているようです。

HTML+CSS とブラウザ

CSS は、HTML とのセットで、また表示・印刷用プログラムとしてはブラウザを前提として説明されることが多いようです。CSS を解説する書籍はほとんどが HTML をどうやってブラウザで綺麗に表示するかという観点から CSS を解説しています。

Internet Explorer や Netscape Navigator は HTML+CSS をサポートしています。しかし、ブラウザのバージョンによってサポートレベルが異なっており、互換性に問題があるようです。

W3C の CSS のホームページには、CSS をサポートするブラウザのリストがあります。Netscape の Netscape6 が、最も CSS を良くサポートするブラウザの一つとしてあげられています。

ブラウザで HTML+CSS を使用する場合、CSS ファイルを外部ファイルとしてリンクする方法と、HTML ファイルの中に CSS を記述する方法があります。

外部 CSS ファイルをリンクする方法を使えば、ソース HTML ファイルを変更することなくスタイルシートの内容を変更して見かけを変えることができます。また、複数の HTML ファイルで同一のスタイルシートを共有できますので便利です[7]。

HTML4.0/XHTML から外部 CSS をリンクする例

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="ja"
      lang="ja">
       <head>
              <title>
              Antenna House XSL Formatter オンラインヘルプ
              </title>
              <meta http-equiv="Content-Type"
                    content="text/html; charset=Shift_JIS" />
              <link rel="stylesheet"
                    href="axf.css"
                    type="text/css"/>
       </head>
       <body >
       ...略...
       </body>
</html>
```

link 要素には次のことを指定します。

- リンクの型："stylesheet"へのリンク

[7] HTML のバージョンとスタイルシートについて

- HTML3.2 では STYLE 要素に type 属性は使えません。type 属性は、HTML4.0/XHTML から定義されました。
- HTML3.2 では LINK 要素に type 属性は使えません。type 属性は、HTML4.0/XHTML から定義されました。
- HTML4.0 と XHTML には、スタイルシートで指定するような表示に関連する要素を除外した HTML4.0 Strict.dtd/XHTML Strict.dtd があります。これらの DTD は、CSS で表示に関する部分を指定し、HTML タグで論理構造を指定するように機能分類したものです。HTML+CSS をセットで使うときには Strict.dtd で規定する要素のみを HTML ファイル中で使うのが望ましいとされています。

- "ref"属性：スタイルシート・ファイルの位置。上の例では、axf.css ファイルが、CSS スタイルシート・ファイルで、HTML ファイルと同じホルダに置くことになります。
- リンクされるスタイルシートの型："text/css"(HTML4.0 と XHTML のみ)

外部 CSS ファイルを使わないで、次の例のように、HTML ファイルの STYLE 要素の中に CSS を記述することもできます。

HTML4.0/XHTML ファイルの中で CSS 記述

```
<html>
       <head>
              <style type="text/css">
              body {color: red}
              h1 {color: blue}
              </style>
       </head>
       ...略...
</html>
```

CSS による XML 表示

CSS は HTML のみではなく、XML を表示・印刷する方法として使うこともできます。セレクタとして、HTML のツリーではなく XML ツリーのパターンを指定すれば良いのです。

XML 文書を表示するためのスタイルシートの例を示します。

XML を表示するスタイルシートの例

```
author {display: block;
        font-size: 18pt}


title {font-size: 24pt;
       background-color: yellow;
       display: block;}

date {display: block}

section title {display: block;
            font-size: 16pt;
            color: blue}

p {display: block; font-size: 11pt}
```

CSS を使って XML を IE5 で表示する機能を紹介した例としては、次の書籍があります。「ステップバイステップで学ぶ XML 実践講座」（マイクロソフト・プレス、日経 BP 社）。CSS の構文は直感的で簡単ですので、XML をブラウザで表示する簡単な方法として活用が期待できます。

CSS の限界

CSS のセレクタでは、ツリーに対してパターンがマッチするかどうかを指定できます。しかし、XML のツリー構造を変換することができません。

XML 文書は、要素をツリー化して文書の論理構造を表現していますので、ツリーの枝をコピーしたり、削除したり、うまく変換して新しく構造の異なるツリーを簡単に作り出すことができるという利点があります。このあたりが、バイナリ形式のワープロ・ドキュメントでは、簡単にできない、XML 独自のメリットなのです。

例えば、表示・印刷する際に、次のようなことが簡単にできれば便利です。

- XML の中の要素の順番を並び替える
- 二次元の配列になっていないものから二次元配列の表を自動的に生成する
- title 要素を選択してコピーして目次を自動的に作り出す
- index 要素をコピーして索引を作り出す
- note 要素を取り出して、ツリーに新しい枝（注の一覧など）を作る。ツリーの作り方次第で、注の置き場所を簡単に移動できます。

しかし、CSS ではこれを行うことができません。これを行うことができるのが、次に説明する XSLT です。

XSLT 仕様の概要

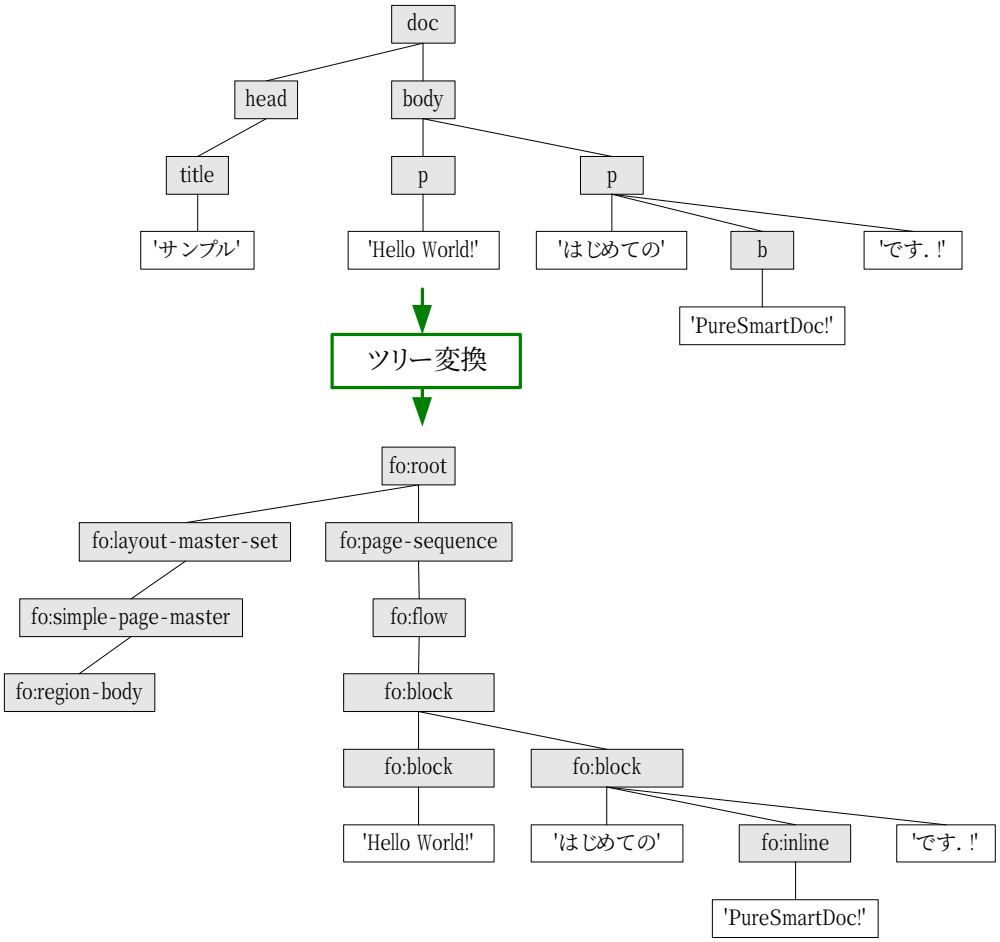
XSLT 仕様は XML 文書のツリー構造を変換するための仕様です。具体的な利用例は後述しますが、1999 年 11 月に W3C の勧告として標準化されて以来、もっとも多くの書籍が書かれ、実際に使われている仕様と言えます。

XSLT の仕様は XSLT 言語の文法と意味を定義しています。

XSLT の仕様を実現するのが XSLT プロセサです。XSLT プロセッサは XML のソース文書を結果 XML 文書[8]に変換するものです。

XSLT 言語による変換規則を記述した文書は XSLT スタイルシートと呼びますが、XML 名前空間に適合する整形式（Wellformed）XML 文書です。変換規則には、XSLT 言語で定義されている要素（XSLT 名前空間の要素）と定義されていない要素を含みます。XSLT 名前空間に属する要素は、XSLT プロセサに対する命令となります。処理結果は定義されていない要素とともに結果ツリーに出力されます。

[8]結果文書は、XML のみではなく、単なるテキストから RTF のような書式付きテキストまで広範囲の出力もできます。



XSLT の命令セット

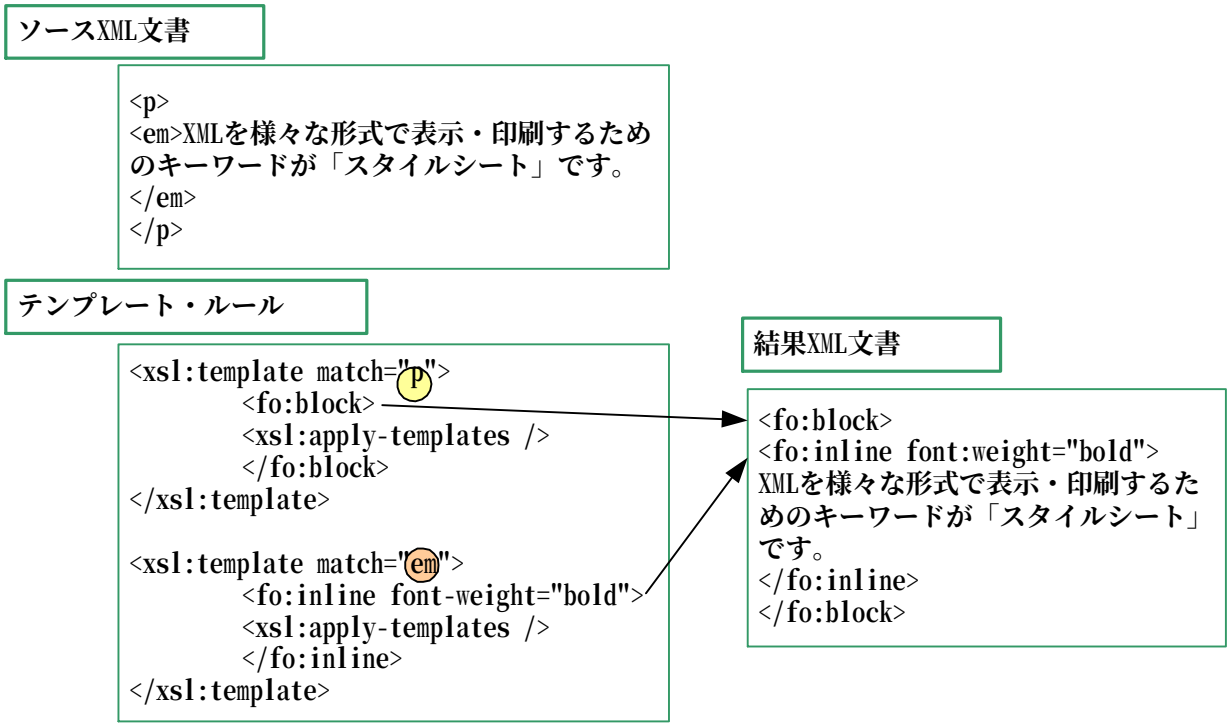
XSLT 言語で定義されている要素の一覧は次の通りです。

分類	命令	説明
スタイルシート	xsl:stylesheet	スタイルシート
	xsl:transform	xsl:stylesheet の同義語
	xsl:include	別の XSLT スタイルシートを取り込む
	xsl:import	別の XSLT スタイルシートをインポートする
	xsl:strip-space	空白を保存しない要素
	xsl:preserve-space	空白を保存するべき要素
テンプレート	xsl:template	テンプレート・ルールを定義する
	xsl:apply-templates	テンプレート・ルールを適用する。カレントノードのすべての子を再起的に処理します。select 属性を使用すると式で選択したノードのみを処理します。
	xsl:apply-imports	テンプレート・ルールを上書きする
	xsl:call-template	名前付きテンプレートを呼び出す

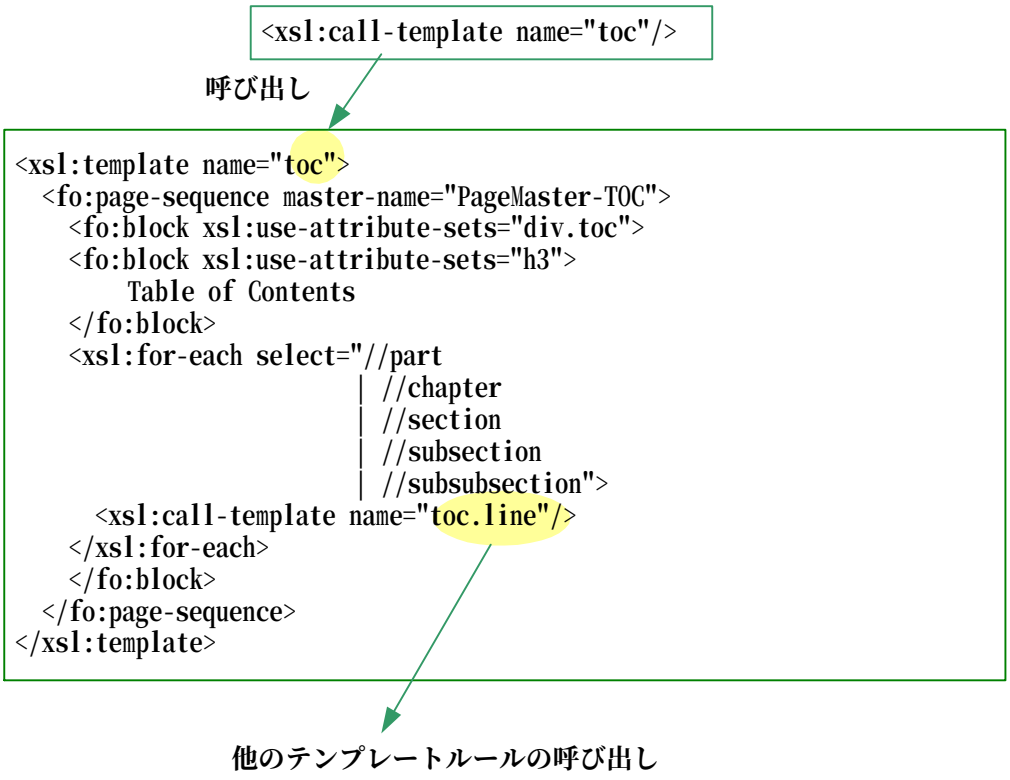
分類	命令	説明
出力	xsl:namespace-alias	１つの名前空間を他の名前空間のエイリアスであると宣言する
	xsl:element	計算処理した名前で要素を生成する
	xsl:attribute	結果要素に属性を追加する
	xsl:attribute-set	名前の付いた属性の集合を定義する
	xsl:text	テキストを生成する
	xsl:processing-instruction	処理命令を生成する
	xsl:comment	注釈を生成する
	xsl:copy	カレントノードのコピーを生成する
	xsl:value-of	select 属性の式を評価した結果を文字列として生成する
	xsl:number	整形した番号数値を結果ツリーに挿入する
フローコントロール	xsl:for-each	select 属性で選択した全ノードをカレントノードとして繰り返す
	xsl:if	test 属性の式を評価した結果が真なら内容のテンプレートを処理する
	xsl:choose	いくつかの選択肢から一つを選択する
	xsl:when	xsl:choose に続いて記述され、test 属性の式を評価した結果が真なら内容を処理する
	xsl:otherwise	テスト結果が真の xsl:when が無いとき処理する
ソート	xsl:sort	ソートする
変数、パラメータ	xsl:variable	変数の名前を指定する
	xsl:param	変数の名前を指定する
	xsl:copy-of	ノード集合を文字列に変換することなく結果ツリーにコピーする
	xsl:with-param	テンプレートへのパラメータの引渡し
その他	xsl:key	キーの宣言
	xsl:decimal-format	10 進フォーマットを宣言する
	xsl:message	メッセージを送信する
	xsl:fallback	フォールバック
	xsl:output	結果ツリーの出力方法を指定する

XSLT スタイルシートの中核は、テンプレート・ルールと呼ばれる xsl:template 要素です。テンプレート・ルールは、xsl:template 要素の match 属性の値であるパターンを使用して適用されるノードを識別します。xsl:template の内容は、テンプレート・ルールがそのノードに適用されたときに処理されるテンプレートです。

次の例では、ソース XML 文書の p 要素に最初のテンプレート・ルールがマッチし、ついで em 要素に、2 番目のテンプレート・ルールがマッチします。出力される結果 XML 文書は、２つのテンプレート・ルールを再帰的に適用した結果となります。



テンプレート・ルールは、また、name 属性で名前をつけておき、他から呼ぶこともできます。



パターンと式

テンプレート・ルールの match 属性値ではノードを指定するためにパターンや式を使用します。パターンや式は XPath で定義する式言語を使用します。式によって、処理対象ノードを選択したり、ノードを異なる方法で処理するための条件の指定、結果ツリー内に挿入するテキストの挿入を行います。

XSLT プロセサ

ソース文書を読みこみ、テンプレート・ルールに従って処理を行い、結果 XML ツリーを出力するのが **XSLT プロセサ**です。

XSLT が W3C の勧告になってから、勧告版の XSLT 仕様を実装する XSLT プロセサが沢山出てきました。その多くは無償で配布されています。現時点で、一番、機能的に強力な XSLT プロセサは、Microsoft が配布している MSXML3.DLL でしょう。これは、正式名称は「MSXML パーサー　バージョン 3.0」と言い、2000 年 11 月 1 日から、Microsoft の Web サイトからダウンロードによる配布が始まっています。まだ、Internet Explorer5.5 には添付されていませんが、MS の開発者がメーリング・リストで発言しているところによりますと、Internet Explorer6 には添付されるようです。

MSXML3.DLL

Microsoft が配布している XML パーサーで、XSLT プロセッサ機能も備えています。

MSXML3.DLL は、プログラムの部品(ライブラリー)です。Internet Explorer から使うか、または、DLL を動かすプログラムを作成しないと、単独の XSLT プロセサとしては使えません。そこで、コマンドラインから使用するためのユーティリティも、別途、Microsoft のサイトから入手することができます。

MSXSL.EXE

MSXML3.DLL は、DLL ですのでエンドユーザが直接使うことができません。MSXML3.DLL をインストールした PC に MSXSL.EXE をインストールしますと、DOS 窓からコマンドラインで XSLT 変換ができるようになります。

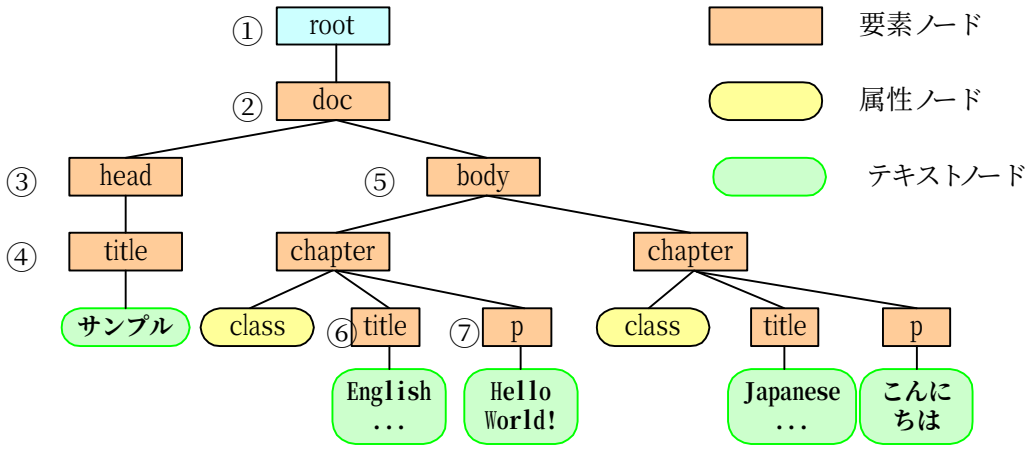
XSLT スタイルシートとツリー変換の仕組み

次に XSLT スタイルシートと XSLT プロセサを使って XML 文書を変換するプロセスを簡単な例で説明します。ソース XML 文書は、XMLPath 言語の項で示した XML 文書です。次の図は、ソース XML 文書のツリーとそれを HTML に変換する最も初歩的な XSLT スタイルシートを示したものです。

XSLT スタイルシートのルート要素は、<xsl:stylesheet>要素です。

ルート要素には２つの名前空間が定義されています。

- xmlns:xsl="http://www.w3.org/1999/XSL/Transform"は、名前空間接頭辞 xsl が、W3C 勧告の XSLT 仕様を指すことを示します。XSLT スタイルシートの中で xsl という接頭辞のついた要素が XSLT で定義する命令セットです。
- xmlns="http://www.w3.org/1999/xhtml"は、デフォルトの名前空間が W3C 勧告の xhtml であることを示します。XSLT スタイルシートの中で名前空間接頭辞のつかない要素は XHTML に属します。




```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml1">
<xsl:template match="/">
①       <xsl:apply-templates />
</xsl:template>

<xsl:template match="doc">
        <html>
②               <xsl:apply-templates />
        </html>
</xsl:template>

<xsl:template match="head">
        <head>
③               <xsl:apply-templates />
        </head>
</xsl:template>

<xsl:template match="head/title">
        <title>
④               <xsl:apply-templates />
        </title>
</xsl:template>

<xsl:template match="body">
        <body>
⑤               <xsl:apply-templates />
        </body>
</xsl:template>

<xsl:template match="title">
        <h1>
⑥               <xsl:apply-templates />
        </h1>
</xsl:template>

<xsl:template match="p">
        <p>
⑦               <xsl:apply-templates />
        </p>
</xsl:template>

</xsl:stylesheet>
```

次にソースの XML 文書の各ノードに対して、それを処理するテンプレート・ルールを作ります。<xsl:template match="">の match 属性の値はそのテンプレートを適用するノードを規定します。XSLT の処理では、ルート・ノードから始めて各ノードに適用されるテンプレート・ルールがあるかどうかを検査します。テンプレート・ルールがあればそのルールを呼び出します。

最初のテンプレート・ルール①はルート・ノードに適用するものです。このルールでは、ルート・ノードについてはなにもしないで xsl:apply-templates の呼び出しを行います。xsl:apply-templates はルート・ノードの子ノードについてテンプレート・ルールを探す処理を行います。テンプレート・ルール①の処理はまだ終了していないことに注意してください。

ルート・ノードの子ノードはルート要素 doc に対応するノードで、適用されるテンプレート・ルールは②です。このテンプレート・ルールは、まず結果ツリーに html の開始タグを出力します。次に xsl:apply-templates の呼び出しを行います。xsl:apply-templates の呼び出しで、doc ノードの子ノードの処理に進みます。

doc ノードの最初の子ノードは head ノードで、適合するテンプレート･ルールは③です。そこで、head の開始タグを出力します。次に xsl:apply-templates の呼び出しを行います。head ノードの子ノードは title ノードですが、title ノードに適合するテンプレート・ルールは 2 つあります。④と⑥です。このように適合するテンプレート・ルースが複数ある場合は、テンプレート・ルールの競合が起きますが、XSLT 仕様で優先順位が決まっています。テンプレート・ルールに優先順位プロパティをつける事もできますが、優先順位が指定されていない場合、パスが詳しく指定されている方が優先します。そこで、テンプレート・ルール④が適用され title の開始タグが出力されます。

ソース XML 文書ツリーの title ノードの子はテキスト・ノード「サンプル」です。ところがこのテキスト・ノードに適用するテンプレート・ルールはありません。XSLT では、この場合デフォルト・テンプレートが適用されます。デフォルト・テンプレートは結果ツリーに全ての子ノードのテキストを出力するものです。そこで、結果ツリーの titl 開始タグの内容にテキスト「サンプル」が出力されます。もう子要素はありませんので、テンプレート・ルール④の apply-templates の処理はここで終わりです。そこで、title タグの終了タグを出力し、テンプレート・ルール④の処理が終わります。

テンプレート・ルール③の途中に戻りますが、ソース XML 文書ツリーの head ノードの子ノードはもうありませんので、結果ツリーに head 終了タグを出力してテンプレート・ルール②に戻ります。

テンプレート・ルール②では、doc ノードの子ノード head の処理が終わりましたので、その次の兄弟ノード body の処理に進みます。body ノードに適合するテンプレート・ルールは⑤です。⑤では結果ツリーに body 開始タグを出力します。次に apply-templates がありますので、body の子ノードの処理に進みます。

最初の子ノードは chapter です。しかし、chapter ノードにあてはまるテンプレート・ルールは見つかりません。title ノードまで進むと、テンプレート・ルール⑥が見つかります。そこで、h1 の開始タグを出力します。

title ノードの子はテキスト・ノード「English・・・」です。ところがこのテキスト・ノードに適用するテンプレート・ルールはありません。前に述べたように、デフォルト・テンプレートが適用され、子ノードのテキストを出力されます。結果ツリーの h1 開始タグの内容にテキスト「English・・・」が出力されます。もう子要素はありませんので、テンプレート・ルール⑥の apply-templates の処理はここで終わりです。h1 タグの終了タグを出力し、テンプレート・ルール⑥の処理が終わります。

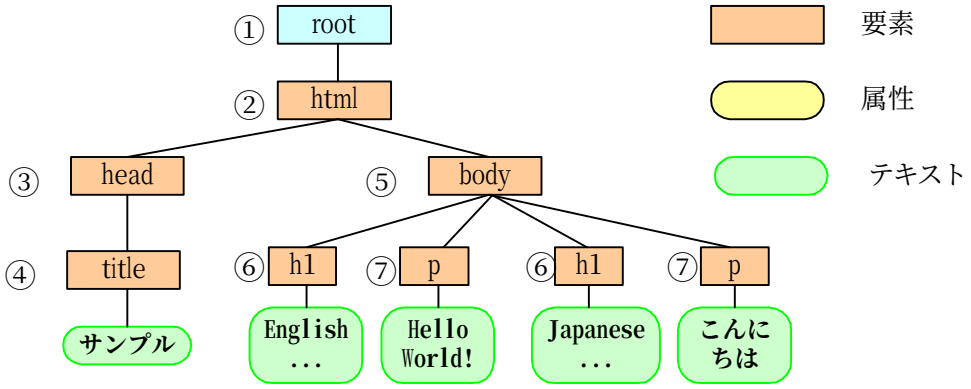
現在、テンプレート・ルール⑤の apply-templates の処理中ですので、次のノード p に進みます。ノード p にはテンプレート・ツール⑦が適合します。テンプレート・ルール⑦により、結果ツリーに p の開始タグ、文字列「Hello World」、p の終了タグの順で出力されます。

また、テンプレート・ルール⑤の apply-templates により、2 つ目の chapter の子ノードである title ノードの処理が行われ、結果ツリーに、h1 の開始タグ、テキスト「Japanese・・・」、h1 の終了タグが出力されます。

また、テンプレート・ルール⑤の apply-templates により、2 つ目の chapter の子ノードである p ノードの処理が行われ、結果ツリーに、p の開始タグ、テキスト「こんにちは」、p の終了タグが出力されます。

これで全てのノードの処理が終わりました。いま処理中のテンプレート・ルールは⑤ですので、結果ツリーに body の終了タグを出力して、テンプレート・ルール②に戻ります。そこで html の終了タグを出力してテンプレート・ルール①に戻り、終了します。

以上の処理により、できあがる結果ツリーは次のようになります。このツリーをシリアライズしたものは XHTML ファイルとなります。

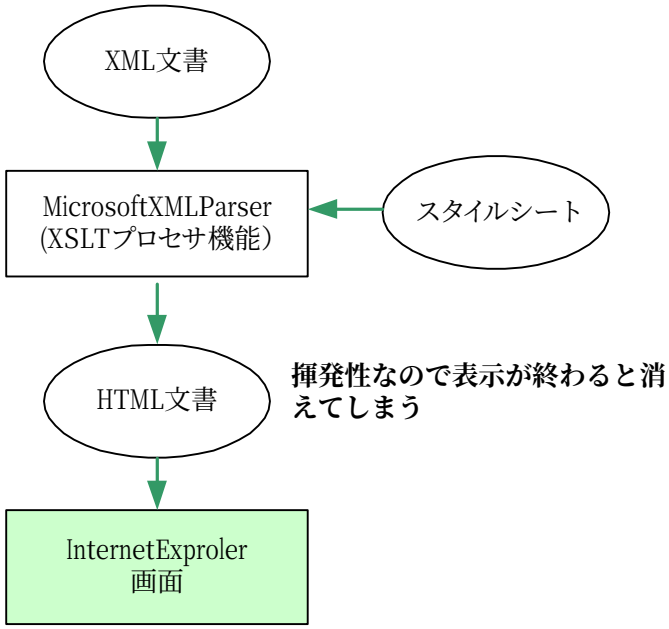


XSLT で XML を HTML に変換する

現在、XSLT の応用として一番の用途は XML 文書を HTML に変換することです。この変換は、サーバ側で行う場合とクライアント側で行う場合の２つのシステムが考えられます。

クライアント PC の IE5 で XML を表示する

一番最初に表の XML ファイルを IE で表示する例を示しました。IE5 で XML 文書をブラウザ画面上で表示することができる仕組をより詳しく示したものが次の図です。



IE5 は、ソース XML 文書を MSXMLL パーサーを使って読み込みます。ソース XML 文書には XSLT スタイルシートがリンクされています。そこで、IE5 は、MSXML パーサーの機能の一部である XSLT プロセサ機能を使って、ソース XML 文書を HTML 文書に変換します。IE5 のブラウザ機能は変換後の HTML 文書を画面に表示します。

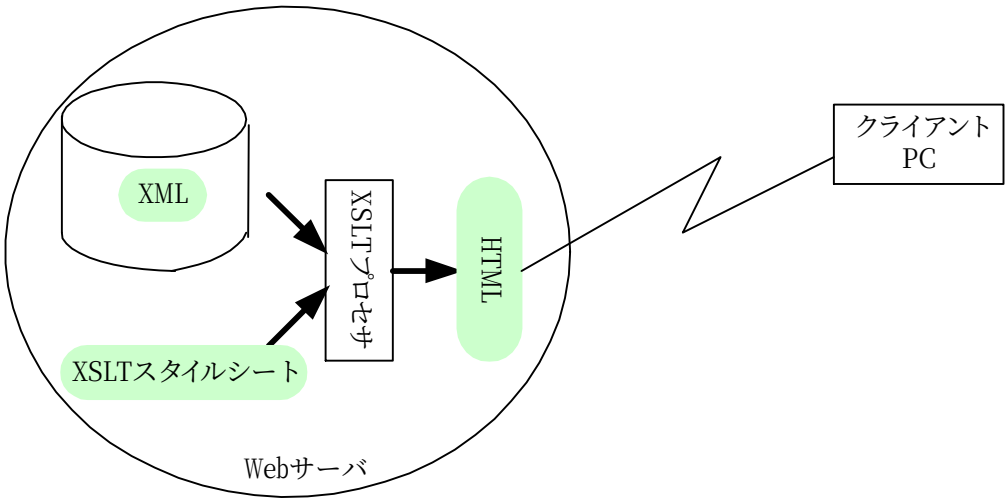
XSLT スタイルシートは、XML 文書とは別のファイルになります。XML 文書から、XSLT スタイルシートを指定するには、次のように XML 文書の先頭に処理命令を置きます。

XML を表示するためにスタイルシートを指定する処理命令

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<!--  XML 文書の先頭に次のような処理命令を置き、スタイルシート
      のファイル名を指定します。-->
<?xml-stylesheet type="text/xsl" href="table2html.xsl" ?>
<!-- XML 文書の本体-->
...XML 文書本文(ルート要素以下）...
```

サーバ側で XML から HTML に変換する

クライアント PC 上のブラウザで、XML を HTML に変換できるかどうかはブラウザに依存します。現在は、まだ、幅広いブラウザで XSLT 機能を使える状態ではありません。従って、企業内のシステムでブラウザを特定できるならば別ですが、もし、不特定多数のクライアントを対象とするならば、サーバ側で XML を HTML に変換するべきです。サーバ側で変換するならば、次のようなシステム構成となります。

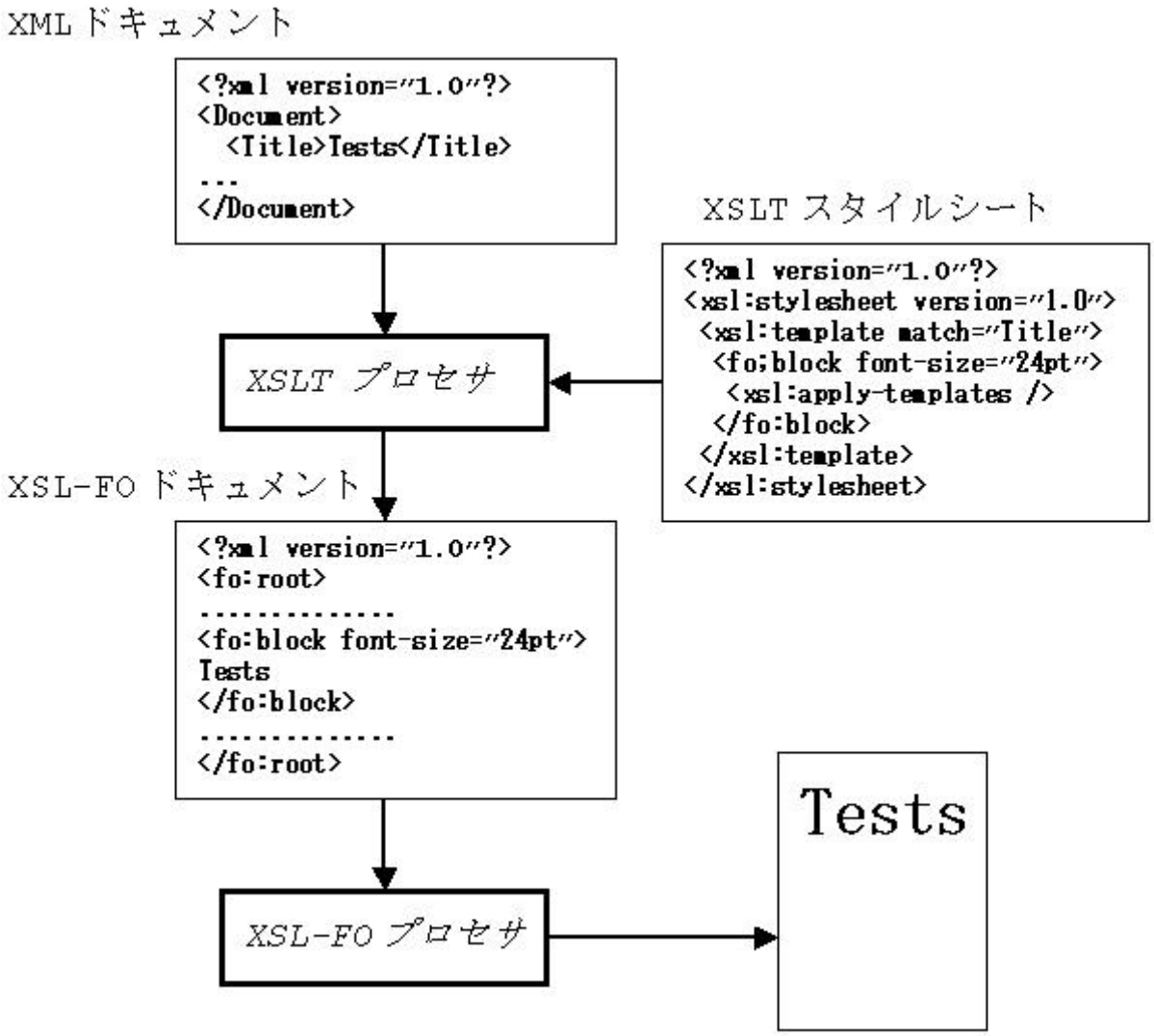


XML 文書を印刷するための XSL 仕様

ドキュメントを XML 化する目的は、Web ページを作成したり、電子的に配布するデータを作り出す、電子的なマニュアルを作成するなど、いろいろあります。どのような意図で XML 文書を作成するにしても奇麗にレイアウトして「紙」に印刷することを欠かせません。しかし、XML 文書を奇麗に組版して印刷するのは技術的に難しい課題です。「Extensible Stylesheet Language(XSL)」仕様は、World Wide Web Consortium (W3C)で、「XML を奇麗にレイアウトして組版するための仕様」として策定が進んでいるもので、XSL ドキュメント印刷のための仕様です。次に、XSL の仕様の概要、その仕様を実装した XSL-FO プロセサ、及びスタイルシートについて解説します。

XSL による組版プロセス概観

XSL による組版プロセスは、次の図のようになります。XML 文書をいきなり組版するのではなく、二段階で行います。



【図】XSL による組版プロセス

第一段階は XSLT プロセサを使って、XML 文書を XSL-FO ドキュメントに変換します。このための変換ルールを記述したものが XSLT スタイルシートです。XSLT は、現在、XML 文書を HTML ドキュメントに変換するために使われることが多いですが、もともとソース XML 文書を XSL-FO ドキュメントに変換するために設計されたものです。

XSL-FO ドキュメントは、コンテンツにレイアウトを指定した状態のもので、ページにレイアウトされた状態ではありません。第二段階で、XSL-FO ドキュメントを組版処理します。XSL-FO ドキュメントの組版処理を担当するのが XSL-FO プロセサです。XSL-FO プロセサが組版結果を印刷したり、PDF に出力したり、画面に表示します。世界で公開されている主な XSL-FO プロセサ(または、組版エンジンとも言います)には次のようなものがあります。

- Antenna House XSL FormatterV1.1：アンテナハウスの XSL-FO 組版エンジンとユーザ・インターフェイス(GUI)。組版エンジンは、日本語も英語も関係なく使うことができます。GUI は日本語版(V1.1J)と英語版(V1.1E)があります。
- FOP：Web サーバの開発で有名な Apache が取り組んでいる JAVA 版 XSL-FO 組版エンジン。XSL-FO ドキュメントから PDF を出力します。サーバ上で XML を PDF に変換することを主な用途としているようです。
- XEP：米国の RenderX というベンチャー企業の JAVA 版 XSL-FO 組版エンジン。XSL-FO ドキュメントから PDF を出力します。PDF 以外に Postscript の出力も可能です。

Antenna House XSL Formatter は、組版エンジンが Windows の ActiveX コントロールであるということから、画面上で対話式に組版結果を確認することができます。XSLT スタイルシートを変更して、結果を直ちに確認できますので、セミナーなどの教育用にも向いています。既に欧米では、XSL Formatter を使って XSL-FO に関する専門のセミナーを開催する業者も現れているなど、人気を得ています。

現状

XSL 仕様は、2000 年 11 月 21 日 Candidate Recommendation[9] となり、W3C は、2001 年春から、XSL-FO プロセサを実装している企業、団体、個人の協力によって仕様と実装の評価を行いました。このため W3C が米国の NIST(National Institute of Standards and Technology)に依頼して作成したテストデータや協力企業・団体の提出したテストデータをまとめたテストスイートが作成されています。各 XSL-FO プロセサの組版結果についても PDF ファイルとして収集されました。日本から試験フェーズに参加しているのはアンテナハウスのみです。7 月に試験のフェーズは終了しました。

XSL 仕様書はエンドユーザ向けではなく開発者向けに書かれています。内容は、最初にページ組版プロセスの概念を説明し、次に XSL-FO プロセサが準拠するレイアウト・モデルを規定しています。さらに Formatting Objects（以下、FO と言います）を規定します。XSL 仕様 Version 1 ではまだ高度なページレイアウトは困難ですが、テクニカル・マニュアル、操作説明書、報告書などのレベルであれば充分実用に耐えると考えています。

フォーマッティング・オブジェクト(FO)とプロパティ

FO とは、段落、リスト、表などの組版の対象になるもののことです。FO には、ページマスターの記述方法やマージンや本文領域の設定方法等ページネーション用のもの、段落やヘッドラインなどブロック・レベルのもの、行中に配置されるインライン・レベルのものなど 56 種類があります。

ページの大きさやエリアの広さ、文字のフォント名やフォントのサイズなどは FO のプロパティで表します。プロパティは 246 種類あり、中には CSS2 の仕様からコピーされたものが多くあります。FO のレイアウト仕様は CSS2 を強く意識して設計されています。

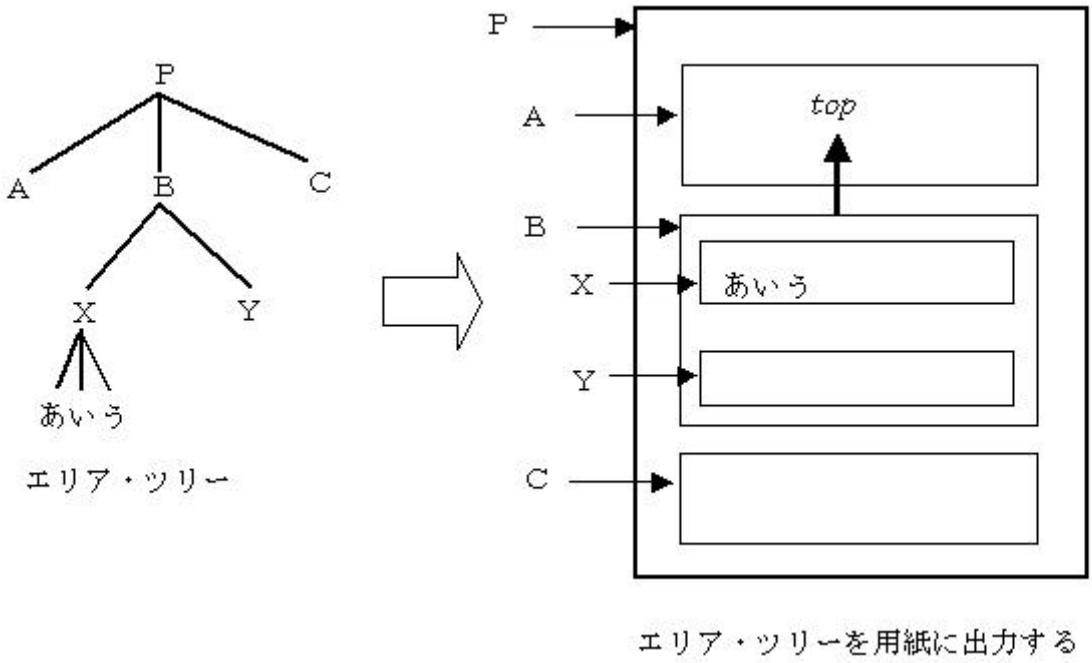
エリア・モデル

XSL 仕様の組版モデルはエリア・モデルといいます。エリア・モデルは CSS2 のボックス・モデルにほぼ対応しています。CSS2 のボックスは XSL 仕様のエリアに相当し、CSS2 の仕様で現れる要素は XSL 仕様では FO に相当します。XSL-FO プロセサは、FO からエリアを生成することで、XSL-FO ドキュメント・ツリーをエリア・ツリーに変換します。XSL-FO ドキュメントの FO は、出力媒体の領域と対応づけられている訳ではなく、エリアが出力媒体上の四角い領域に対応付けられます。エリア・ツリーはドキュメントの中の文字、図形やイメージを配置するための位置情報、間隔空けの情報やその他組版のために必要な情報を含む順序づけられたツリーです。

エリアはブロック・エリアとインライン・エリア(行内エリア)の２種類に分類できます。ブロック・エリアのひとつにライン・エリア(行エリア)がありますが、ライン・エリアはその子供がすべてインライン・エリアになります。また、インライン・エリアのひとつにグリフ・エリアがありますが、グリフ・エリアは子供をもたず、内容文字を表示するグリフを含むという末端のエリアになります。

エリア・ツリーと出力媒体の上の領域との対応関係は次の図のようになります。「Ｐ」がページ、「Ａ、Ｂ、Ｃ」はブロック・エリア、「ＸＹ」がライン・エリア、「あ、い、う」がインライン・エリアに相当します。

[9]XSL 仕様は、2001 年 8 月 28 日に Proposed Recommendation となりました但し、以下の説明は、CandidateRecommendation に基づいています。



【図】エリア・ツリーと出力媒体上の領域

リファレンス・オリエンテーション

ページ全体の絶対方向は、用紙の上が top、下が bottom、左が left、右が right となります。ページ内のエリアの種類の中で、参照エリアについては、その参照エリアを設定する FO の reference-orientation プロパティを設定することで絶対方向を変更することができます。reference-orientation の値は初期値は 0 です。例えば 90 を指定しますと、指定したエリアの参照エリアの座標系はその親の参照エリアの方向に対して反時計廻りに 90 度回転します。これは、用紙の回転あるいは部分縦書などの方向回転用に使います。

ライティング・モード

エリアの中に、子供のブロックを配置していく方向が行の進行方向です。また、インライン・エリアで子供のエリアを配置していく方向は、テキスト文字の進行方向に相当します。この２つの組み合わせをライティング・モードと言います。ライティング・モードの初期値は行の進行方向は上から下、文字の進行方向は左から右です。参照エリアを設定できる FO の writing-mode プロパティを設定することで、ライティング・モードを変更することができます。プロパティの値は次のものがあります。

1. lr-tb(初期値)：インラインのエリア、テキストの文字は左から右、行とブロックは上から下に進む。
2. rl-tb：インラインのエリア、テキストの文字は右から左に進み、行とブロックは上から下に進む。アラビア語、ヘブライ語で使われます。
3. tb-rl：インラインのエリア、テキストの文字は上から下に進み、行とブロックは右から左に進む。日本語、中国語の縦書きで使われます。

Keep と Break

ページ参照エリア(ページ・マージンの内側)、カラム・エリア(段組の段領域)、ライン・エリアの 3 種類のエリアをコンテキスト・エリアと言い、コンテキスト・エリア内で Keep と Break 条件を設定できます。

Break は改ページを行うものです。break-before、break-after の２つの条件があります。それぞれ FO の break-before、break-after プロパティに、page、even-page、odd-page、column、auto を指定することで設定されます。例えば FO の break-before を page に設定すれば、その FO から生成される最初のエリアの前で改ページが行われます。同様に、FO の break-after を page に設定すれば、その FO の生成するエリアの後で改ページが行われます。

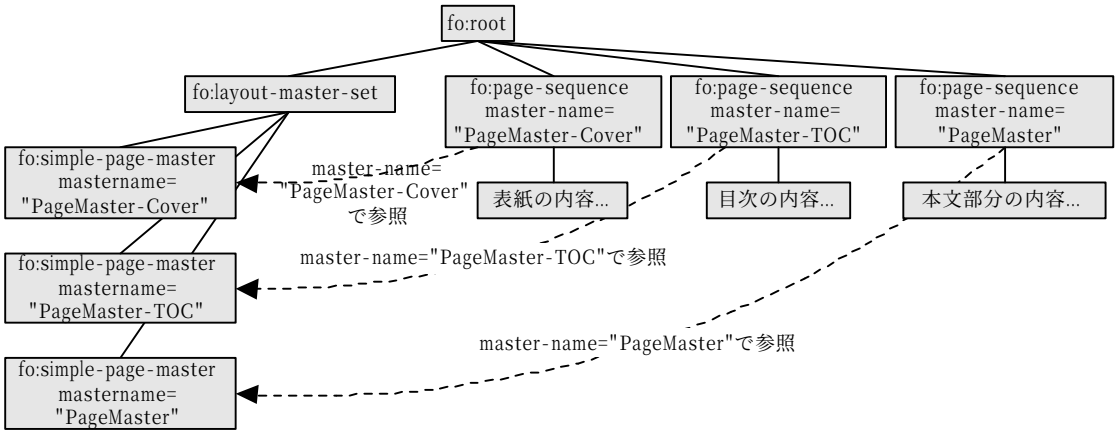
Keep は見出しと本文を同一ページ、同一カラムにするなどのために使うものです。Keep には keep-with-previous、keep-with-next、keep-together の３つの条件があります。それぞれ、FO の keep-with-previous、keep-with-next、keep-together の within-page、within-column、within-line コンポーネントに強度を示す値を設定するか always とします。数字が大きいほど強く always は最強の条件となります。auto にすれば Keep 条件はなにも設定されません。

ページネーションの基本

XSL-FO ドキュメントのツリーのルート・ノードは、fo:root ですが、その子供として 1 つの fo:layout-master-set、fo:declarations(オプション)、一つ又はそれ以上の fo:page-sequence が来ます。

fo:layout-master-set は内容を割り付けるレイアウト用紙とその並び、すなわちレイアウトマスターを必要数だけ規定します。fo:declarations はカラー・プロフィールなどのリソースを記述します。

fo:page-sequance(ページシーケンス)には、ページに流し込む内容を子供として配置します。XSL-FO プロセサは、fo:page-sequence の内容を、指定したレイアウトマスターで規定されるレイアウト用紙に流し込んで行ってページを生成します。レイアウトマスターの指定は、fo:page-sequence の属性 master-name の値に、レイアウトマスターの名前を設定することで指定します。



【図】ページネーション関連の FO ツリー構造

レイアウトマスターセット

レイアウトマスターには fo:simple-page-master と fo:page-sequence-master の 2 種類あります。

fo:simple-page-master(ページマスター)は、1 ページのサイズ、本文領域、ヘッダ、フッタ、サイドバー(左右）などのページ領域を規定します。一つのドキュメントでいろいろなページ書式を使う場合は、fo:simple-page-master を必要数定義し、master-name プロパティにユニークな名前を付けて区別します。

fo:page-sequence-master はページマスターの並び順を規定します。fo:page-sequence-master の子供には、ひとつのページマスターからなる fo:single-page-master-reference とページマスターの繰り返しを規定する fo:repeatable-page-master-reference、 fo:repeatable-page-master-alternatives があります。 fo:repeatable-page-master-reference は、ひとつのページマスターの繰り返しを規定します。fo:repeatable-page-master-alternatives はページマスターの組み合せの繰り返しを規定するもので、偶数ページと奇数ページでページマスターを切り替えたり、最初のページ、最後のページ、ブランク・ページでページマスターを切り替えるなどを実現できます。

ページマスターとページの領域

fo:simple-page-master にはその子要素として、fo:region-body、fo-region-before、fo:region-after、fo-region-start、fo-region-end の５つの領域を指定します。通常の横書モードでは、fo:region-before がヘッダ、fo:region-after がフッタ、fo:region-start が左サイドバー、fo:region-end が右サイドバーに相当します。ページのサイズは page-width と page-height で指定します。ページには上下、左右に margin-top、margin-bottom、margin-left、margin-right が取られます。

ページシーケンス

fo:page-sequence はフローと呼ばれ、レイアウトマスターの５つの領域に配置するコンテンツを指定します。フローには静的なもの fo:static-content と fo:flow があります。fo:static-content は、通常はヘッダやフッタの領域に配置され、すべてのページに繰り返されます。fo:flow は、通常は本文(fo:region-body)に配置されるテキストを内容として持ちます。

XSL-FO プロセサは fo:page-sequence を処理する際、fo:page-sequence の master-name プロパティで指定しているページマスターまたは fo:page-sequence-master を参照してページを作り出します。ひとつひとつのフローには、flow-name プロパティで名前を設定します。ページマスターの中の領域とフローの対応関係を flow-map によって決定します。

ブロックレベル要素、リスト、表

ブロックとブロックコンテナ

XSL-FO ドキュメントでは、本文テキストには見出し要素、段落要素などの区別がなく、テキストはすべて fo:block の内容となります。フォントの種類や文字の大きさなどは fo:block のプロパティとして指定します。

fo:block-container は、横書文書の中に部分縦書を挿入するなど、本文と異なるライティング・モードのエリアを組み込むのに一般的に使います。

リスト

箇条書は fo:list-block によって作成します。各箇条は、fo:list-item になり、list-item の中に fo-list-item-label(リストのマークを表示する部分)と list-item-body(リストのコンテンツを表示する部分)を配置します。リストのラベル項目を明示的に指定する必要があります。したがって箇条書きであれば箇条書き項目の先頭の記号を、番号付きリストであれば xsl:number で明示的に番号を振らなければなりません。また、list-item-label と list-item-body の開始位置を明示的に指定しなければなりません。

表

表は fo:table-and-caption を使って、表本体と表のキャプションを一緒に組版できます。表本体は、fo:table の子になります。ひとつの表は、ヘッダとフッタとボディから構成します。ヘッダとフッタはオプションですが、表がページの途中で次ページに跨った、ヘッダとフッタを繰り返すかをプロパティで指定できます。fo:table の子供要素の fo:table-column はオプションですが、同じカラムとスパンを有するセルの特性を指定するとき使います。表のカラム幅は、fo:table-column の column-width プロパティで指定します。表のセルには、セル単位で修飾ができます。さらに、枠線や間隔については、上下左右の 4 辺に対して個別に設定することができます。セル単位でバックグラウンド・カラーを設定したり、セルの枠線とテキスト文字列との間の間隔(padding)を設定できます。

その他の F0

インラインの F0 による文字修飾

イタリック、ボールドなどの文字修飾、あるいは、上付き、下付きなどは、fo:inline で指定します。fo:inline の font-size プロパティに文字サイズを指定することで文字の大きさを変更することができます。fo:inline-container を使えばルビの表現もできます。また、画像もインライン要素である fo:externat-graphic を使って行中に埋め込むことができます。

脚注

脚注は脚注を付けたい文字の後ろに fo:footnote を設定します。fo:footnote-body に脚注の本文を設定します。

索引や目次の作成方法

索引や目次の項目に fo:page-number-citation を使ってページ番号を引用したり、リーダをつけることができます。ページ番号を入れたい箇所に fo:page-number-citation を置き、そのプロパティ ref-id に参照先項目の ID 番号を付けます。また、本文の中の該当項目には、fo:inline のプロパティとして ID 番号を付けます。リーダは fo:leader によって設定します。leader と fo:page-number-citation が配置される fo:block に text-align-lastjustify を設定すればページ番号を行末揃えし、間をリーダで埋めることができます。

XSL-FO 用 XSLT スタイルシート作成

XSL-FO ドキュメントを作成する方法として、標準的な方法は既に述べた XSLT スタイルシートを使う方法です。XML 文書を XSL-FO ドキュメントに変換するための XSLT スタイルシートを作成するには、XSLT のルールの書き方と、XSL 仕様に関する知識が必要となります。既に説明しましたように XSL-FO ドキュメントではページレイアウト、ブロック間の間隔なども細かく指定しますので、XML を XSL-FO に変換する XSLT スタイルシートは、XML を HTML に変換する XSLT スタイルシートよりも複雑です。

XSL-FO への XSLT スタイルシートの作成のステップを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。しかし要素や属性の種類・内容、出現順序などの、通常 DTD に記述される情報が XSLT 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-FO 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～本文のレイアウト設定、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-FO 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 XSLT 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良いでしょう。
XSLT 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 XSLT 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。XSLT 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もあります。XSLT そのものは日本でも書籍が出始めて来ていますので、参考にすると良いでしょう。

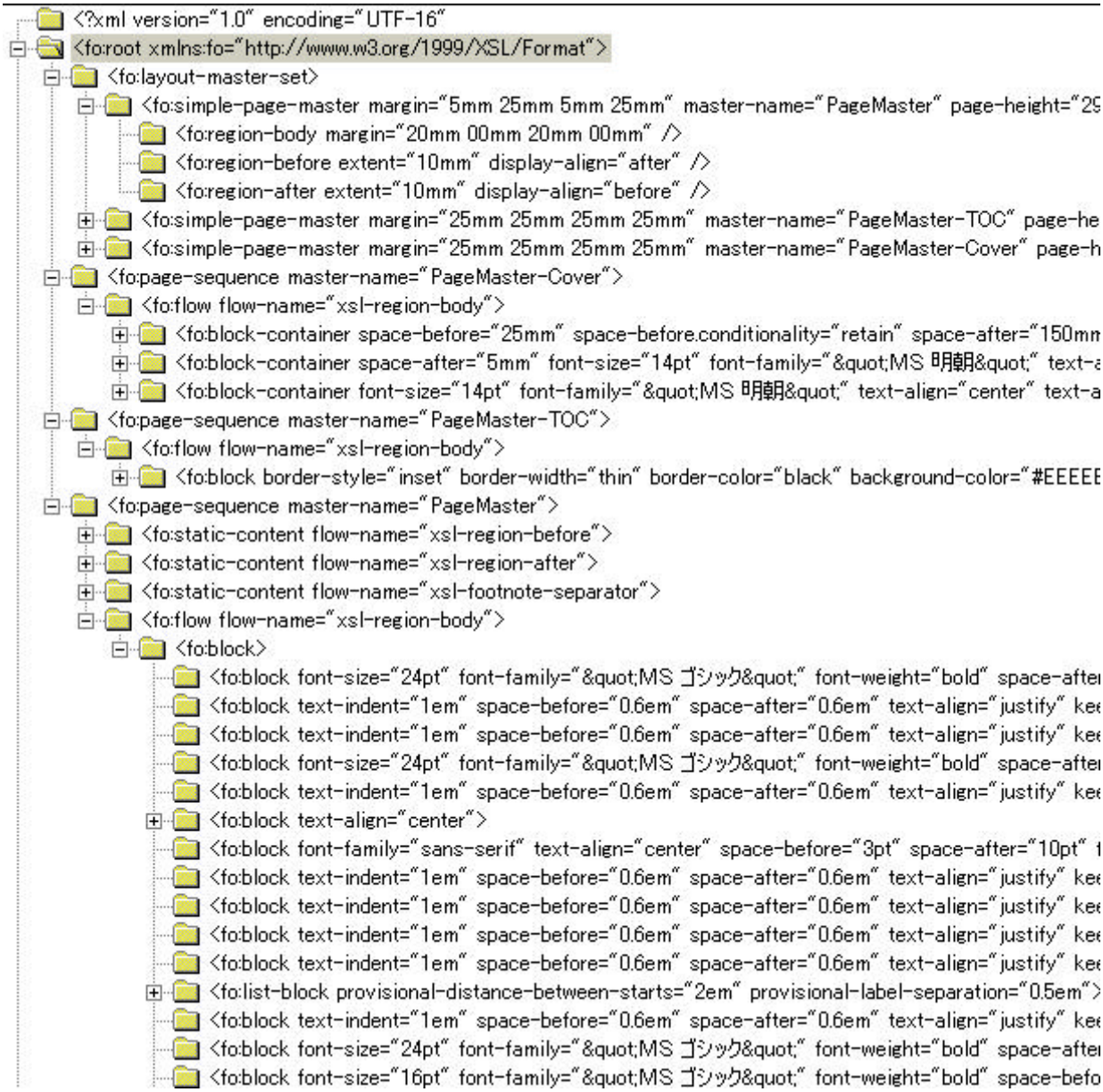
XML 文書の例

XML 文書の構造が図のようになっていたとします。ルート要素は doc で、doc の子供に head と body 要素があります。head 要素の内容に表紙、body 要素の内容に本文を配置しています。

【図】XML 文書のツリー構造

XSL-FO ファイルの例

これを XSL-FO に変換する XSLT スタイルシートを説明する前に、順序は逆ですが、変換された XSL-FO ドキュメントを説明します。次の図は XML 文書を XSL-FO に変換したものです。fo:root がルート要素、その子供には、まず fo:layout-master-set 要素があります。その子供に本文のページマスター(PageMaster)、表紙のページマスター（PageMaster-Cover)、目次のページマスター（PageMaster-TOC）があります。fo:root の次の子供として、fo:page-sequence が３つあります。一つは、表紙の内容（<fo:page-sequence master-name="PageMaster-Cover">)、次は目次の内容、（<fo:page-sequence master-name="PageMaster-TOC">)、最後に本文の内容（<fo:page-sequence master-name="PageMaster">）があります。



【図】XML 文書のツリー構造

XSLT スタイルシートの全体構造

XML 文書を XSL-FO に変換する XSLT スタイルシートの全体構造は次のようになります。

- XSLT スタイルシートでは、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(これらは fo:page-sequence)の順で FO ツリーを生成します。
- 表紙と目次は、入力の XML データの順に沿った XSLT 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ります。
- これらの制御を、ルート要素の doc を処理するテンプレートで行います。

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>
      <!-- ページレイアウトの設定(fo:simple-page-master).  -->
```

```
    </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" />
  </fo:root>
</xsl:template>
```

さらに表紙を作るテンプレート、目次を作るテンプレート、本文を処理するテンプレートを含んでいます。

参考資料

仕様書

XML 基本仕様書

「Extensible Markup Language」(原文）W3C(http://www.w3.org/TR/1998/REC-xml-19980210)
「拡張可能マークアップ言語」（日本語訳）どらねこ本舗 (http://www.doraneko.org/xml/xml10/xml10.html）
「拡張可能なマーク付け言語（XML）1.0」（日本語訳）(http://www.fxis.co.jp/DMS/sgml/xml/rec-xml.html)

名前空間の仕様書

「Namespaces in XML」(原文）W3C(http://www.w3.org/TR/1999/REC-xml-names-19990114)
「XML ネームスペース」(日本語訳) どらねこ本舗（http://www.doraneko.org/xml/namespace/namespace.html）

XML パス言語(XPath 仕様)

「XML Path Language(XPath) Version 1.0」（原文）W3C (http://www.w3.org/TR/1999/REC-xpath-19991116)
「XML パス言語（XPath)Version 1.0」(日本語訳) どらねこ本舗（http://www.doraneko.org/xml/xpath/REC991116.html）

DOM 仕様書

「Document Object Model (DOM) Level 1 Specification」W3C (http://www.w3.org/TR/1998/REC-DOM-Level1-19981001)
「文書オブジェクトモデル(DOM)第 1 水準 仕様書 Version 1.0」(日本語訳) どらねこ本舗 (http://www.doraneko.org/misc/dom1/cover.html)

XML Schema 仕様書

「XML Schema Part0:Premier」W3C 勧告（http://www.w3.org/TR/xmlschema-0/）
「XML Schema Part1:Structure」W3C 勧告（http://www.w3.org/TR/xmlschema-1/）
「XML Schema Part2:Datatypes」W3C 勧告（http://www.w3.org/TR/xmlschema-2/）
「XML Schema Part0:Premier」（日本語訳）W3C working Draft 7, April 2000 の翻訳 日本アイ・ビー・エム(http://www.tr.ibm.com/projects/xml/xmlschema-oj.html)*翻訳版は古いドラフトで、仕様が変更になっている部分的があるので注意。

Relax 仕様書

Relax 公式サイト　http://www.xml.gr.jp/relax
「XML 正規言語記述 Relax コア」JIS TR X0029:2000 (http://www.y-adagio.com/public/standards/tr_relax_c/toc.htm)
「XML 正規言語記述 Relax 名前空間」JIS TR X0044:2001 (http://www.y-adagio.com/public/standards/tr_relax_ns/toc.htm)

XSLT 仕様書

「XSL Transformations(XSLT) Version 1.0」原文）W3C (http://www.w3.org/TR/1999/REC-xslt-19991116)
「XSL Transfortmations バージョン 1.0」(日本語訳）インフォテリア（http://www.infoteria.com/jp/contents/xml-data/REC-xslt-19991116-jpn.htm）

XSL 仕様書

「Extensible Stylesheet Language」W3C Candidate Recommendation (http://www.w3.org/TR/2000/CR-xsl-20001121/)
Proposed recommendation (http://www.w3.org/TR/2001/PR-xsl-20010828/)

関連文書

DOM プログラミングについて

「DOM プログラミング虎の巻」浅海 智晴 http://www.asahi-net.or.jp/~dp8t-asm/java/articles/XMLJava2/article.html

MSXML パーサーについて

「置換モードによる MSXML3.DLL インストール」（日本語）。MSXML3.DLL と旧バージョンとの関係について説明した資料。

http://www.microsoft.com/japan/developer/workshop/xml/general/replacemode.asp

SAX について

SAX は XML-dev のメーリングリストで作成されました。まとまった情報は、次のホームページで入手できます。草の根的に作られたため詳しいドキュメントがないようです。SAX1 のドキュメントは下記にあります。また、Apache の Xerces のドキュメントなども参考になります。
http://www.megginson.com/SAX/

書籍

「改訂版 XML 完全解説(上)」株式会社日本ユニテック著 技術評論社発行

XML の基礎からの理論的解説があります。XML 全体の知識を得るには適切な書籍です。

「ステップバイステップで学ぶ XML 実践講座」（日本語訳）ミッチェル・J・ヤング著日経 BP ソフトプレス発行

初歩から具体的な例を用いて説明しています。やや、Microsoft 独自機能の説明に偏っているのが難点です。

「XML 技術リファレンス W3C 正式仕様の完全解説」（日本語）ボブ・ドゥシャーム著株式会社ピアソン・エデュケーション発行

XML Version1 仕様を仕様書に沿って解説した書籍です。

「XML バイブル」（日本語）エリオット・ラスティ・ハロルド著　日経 BP 出版発行

XML に関する技術を網羅した概説書。XSL 変換（XSLT）、XSL-FO に関する新しい解説があります。

プログラム、ツール

XSLT Samples Viewer（英語）

XSLT の要素と関数、及び XPath 関数のサンプル・スタイルシートを提供する、ビジュアルでインタラクティブな学習ツール。

http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/MSDN-FILES/027/001/677/msdncompositedoc.xml (http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/MSDN-FILES/027/001/677/msdncompositedoc.xml)

MSXML Parser 3.0 Service Pack 1 Release

Microsoft の MSXML3.DLL(無償)。

http://msdn.microsoft.com/downloads/default.asp?URL=/code/sample.asp?url=/msdn-files/027/001/591/msdncompositedoc.xml (http://msdn.microsoft.com/downloads/default.asp?URL=/code/sample.asp?url=/msdn-files/027/001/591/msdncompositedoc.xml)

Instant Saxon

Michael H. Kay 氏によるコマンドラインの XSLT プロセサです。

http://www.users.iclway.co.uk/mhkay/saxon/saxon6.2/instant.html (http://www.users.iclway.co.uk/mhkay/saxon/saxon6.2/instant.html)

Antenna House XSL Formatter

XSL-FO 組版エンジン(XSL-FO プロセッサ)。下記アドレスより 30 日間評価版を入手していただけます。

http://www.antenna.co.jp/XML/axf11/axf11top.htm

Apache XML Project

Web server で有名なアパッチには XML 関連のインフラに相当するツール類の開発プロジェクトがあります。オープンソースで、商用レベルの標準に基づく XML ツールを揃えようとしています。現在、次のプロジェクトが進んでいます。

URL:http://xml.apache.org/

ツール名	内容
Xerces	XML パーサー。Java, C++ (with Perl and COM bindings)
Xalan	XSLT プロセッサ。Java and C++
Cocoon	XML-based web publishing, in Java
FOP	XSL FO プロセッサ, in Java
Xang	Rapid development of dynamic server pages, in JavaScript
SOAP	Simple Object Access Protocol
Batik	A Java based toolkit for Scalable Vector Graphics (SVG)
Crimson	A Java XML parser derived from the Sun Project X Parser.

XSL 仕様小史

XML のスタイルシートに関して、なぜ、現在のような混乱が生じているかを理解するには、XSL 仕様の歴史を知る必要があります。スタイルシート（XSL）[10]は、1997 年 8 月に、Microsoft、Inso、ArborText から W3C に提案されました。

初期の「Extensible Stylesheet Language」ドラフト仕様では、XSLT 仕様と XSL-FO の仕様が、ひとつの仕様の前段と後段を構成していました。初期のドラフト仕様では、XSLT は、XSL-FO を作成するための変換言語として設計されたのです。

ところが、1999 年 4 月に「Extensible Stylesheet Language」仕様は、「XSL Transformations」（XSLT)、「Extensible Stylesheet Language」（XSL-FO）の 2 つの仕様に分離しました。

さらに、XSLT 仕様は、同年 7 月に XSLT 仕様と「XML Path Language」(XPath 仕様）に分離しました。

そして、XSLT、XPath 仕様の方が一足先に、1999 年 11 月に W3C 勧告になりました。

当初から Microsoft は、XSLT/XPath 仕様に積極的に取り組んでいます[11]。こうしたことから、今では、XSLT 仕様は、XSL-FO の生成に限らず、汎用の XML 変換言語の仕様として幅広く使われるようになりはじめています。

[10]提案時の仕様案である「A Proposal for XSL」は、現在とはまったく違うものです。
[11]しかし、Microsoft は XSL-FO の方には熱心でないようです。XSL-FO 仕様に取り組んでいるのは、むしろ ArborText の方と思われます。