

目 次

	ページ
序文.....	1
0 適用範囲.....	1
1 導入.....	1
2 位置パス.....	3
2.1 位置ステップ.....	5
2.2 軸.....	6
2.3 ノード試験.....	7
2.4 述部.....	8
2.5 短縮構文.....	8
3 式.....	10
3.1 基本.....	10
3.2 関数呼出し.....	10
3.3 ノード集合型.....	11
3.4 論理型.....	12
3.5 数値型.....	13
3.6 文字列型.....	14
3.7 字句構造.....	15
4 主要関数ライブラリ.....	16
4.1 ノード集合関数.....	16
4.2 文字列型関数.....	17
4.3 論理型関数.....	20
4.4 数値型関数.....	20
5 データモデル.....	21
5.1 ルートノード.....	22
5.2 要素ノード.....	22
5.3 属性ノード.....	23
5.4 名前空間ノード.....	24
5.5 処理命令ノード.....	24
5.6 コメントノード.....	24
5.7 テキストノード.....	24
6 適合性.....	25
附属書 A (規定) 文献.....	26
附属書 B (参考) XML 情報集合との対応付け.....	28

まえがき

この規格は、工業標準化法第 12 条第 1 項の規定に基づき、財団法人日本規格協会(JSA)から、工業標準原案を具して日本工業規格を制定すべきとの申出があり、日本工業標準調査会の審議を経て、経済産業大臣が制定した日本工業規格である。

この規格は、著作権法で保護対象となっている著作物である。

この規格の一部が、特許権、出願公開後の特許出願、実用新案権又は出願公開後の実用新案登録出願に抵触する可能性があることに注意を喚起する。経済産業大臣及び日本工業標準調査会は、このような特許権、出願公開後の特許出願、実用新案権又は出願公開後の実用新案登録出願に係る確認について、責任はもたない。

原勧告の標題及びまえがきの翻訳

XML パス言語 (XPath) 1.0

W3C 勧告 1999 年 11 月 16 日

この版の掲載場所

<http://www.w3.org/TR/1999/REC-xpath-19991116>

(XML 又は HTML で入手可能。)

最新版の掲載場所

<http://www.w3.org/TR/xpath>

以前の版の掲載場所

<http://www.w3.org/TR/1999/PR-xpath-19991008>

<http://www.w3.org/1999/08/WD-xpath-19990813>

<http://www.w3.org/1999/07/WD-xpath-19990709>

<http://www.w3.org/TR/1999/WD-xslt-19990421>

編者

James Clark <jjc@jclark.com>

Steve DeRose (Inso Corp. and Brown University) <Steven_DeRose@Brown.edu>

著作権 © 1999 W3C® (MIT, INRIA, 慶應義塾) が、すべての権利を保有する。免責、商標、文書の使用及びソフトウェアの使用許諾に関する W3C の規則を適用する。

要約

この勧告は、XML 文書の部分を番地付けするための言語である XPath を規定する。XPath は、XSLT 及び XPointer の両者によって用いられるように設計されている。

この文書の状態

この文書は、W3C の勧告である。この勧告は、W3C 会員企業及び関連する団体によって閲読されており、技術統括責任者によって W3C 勧告として承認されている。これは安定した文書であり、参考資料として使用してよく、他の文書から引用規定として引用してもよい。W3C はこの勧告を制定することによって、この規定への注目を喚起し、広い普及を促進するという役割を果たす。この結果、Web の機能及び相互運用性が高まる。

この勧告についての正誤表は、<http://www.w3.org/1999/11/REC-xpath-19991116-errata> から入手できる。

この勧告についてのコメントは、www-xpath-comments@w3.org に報告されたい。コメントの一覧が、入手できる。

この勧告の英語版だけを規定としての版とする。しかし、翻訳については、<http://www.w3.org/Style/XSL/translations.html> を参照されたい。

現在の W3C 勧告及び他の技術文書の一覧は、<http://www.w3.org/TR/>で見ることができる。

この勧告は、XSL 作業グループ及び XML リンク付け作業グループの共同作業であって、W3C Style 及び W3C XML の活動の一部である。

白 紙

XML パス言語 (XPath) 1.0

XML Path Language (XPath) Version 1.0

序文

この規格は、1999 年 11 月に World Wide Web Consortium (W3C) から公表された XML Path Language (XPath) Version 1.0 勧告を翻訳し、その後に発行された正誤表を取り込んで、技術的内容を変更することなく作成した日本工業規格である。

なお、この規格で点線の下線を施してある箇所は、原勧告にはない事項である。

0 適用範囲

この規格は、XML 文書の部分を番地付けするための言語である XPath を規定する。XPath は、XSLT 及び XPointer の両者によって用いられる。

1 導入

XPath は、XSL 変換 (JIS X 4169 で定義する。以下 XSLT ともいう。) 及び XPointer[XPointer] で共有する機能に関し、共通の構文及び意味を提供する活動によって開発した。XPath の主要な目的は、XML (JIS X 4159) 文書の部分を番地付けすることにある。この主要な目的を支援するために、XPath は、文字列型、数値型及び論理値型を扱うための基本的な機能をも提供する。XPath は、URI の中及び XML の属性値の中で XPath を利用しやすくするために、簡潔であって XML ではない構文を用いる。XPath は、XML 文書の見た目の構文ではなく、XML 文書の抽象的で論理的な構造を操作する。XPath は、XML 文書の階層構造をたどるために、URL と同じように、パス (経路) 記法を利用することから、その名が付いている。

注記 原勧告では、外部文書への参照は、[XPointer] などといった記述から参照箇所にリンクされている。ただし、この規格では参照箇所へのリンクは記載しない。代わりに、この記法によって、附属書 A の対応する項目を参照する。

番地付けのための XPath の使用に加えて、XPath は、一致をとる (すなわち、ノードがパターンに一致するかどうかを試験する。) ために使用できる自然なサブセットをもつことも目指して設計されている。XPath のこの使用法は、XSLT (の規定) において示される。

XPath は、XML 文書をノード木としてモデル化する。要素ノード、属性ノード及びテキストノードを含む様々なノードの型がある。XPath は、ノードの各型に関して文字列値を計算する方法を定義する。名前をもつノードの型もある。XPath は、XML 名前空間 (JIS X 4158) を完全に提供する。したがって、ノードの名前は、局所部分と、ヌルになることもある名前空間 URI とから成る対としてモデル化される。これは展開名と呼ばれる。データモデルは、箇条 5 において詳細に示す。

XPath における主要な構文構成子は、式とする。式は、生成規則 Expr に一致する。式は、評価されてオブジェクトを生じる。このオブジェクトは、次の四つの基本的な型の一つをもつ。

- a) ノード集合型（ノードの重複のない順序なし集合）
- b) 論理型（真又は偽）
- c) 数値型（浮動小数点数）
- d) 文字列型（UCS 文字の列。[ISO/IEC 10646]参照。）

式の評価は、文脈に関して行われる。XSLT 及び XPointer の規定は、それぞれ XSLT 及び XPointer で用いる XPath 式について、文脈をどのように決定するかを示す。文脈は、次のもので構成する。

- a) ノード（文脈ノード）
- b) 0 でない正の整数の対（文脈位置及び文脈サイズ）
- c) 変数束縛の集合
- d) 関数ライブラリ
- e) その式の有効範囲内にある名前空間宣言の集合

文脈位置は、常に文脈サイズ以下とする。

変数束縛は、変数名から変数値への対応付けから成る。変数の値は、オブジェクトであって、式の値として可能な任意の型をもつことができ、この規格に規定していない追加の型であってもよい。

関数ライブラリは、関数名から関数への対応付けから成る。各関数は、0 個以上の引数を取り、単一の結果を返す。この規格は、すべての XPath 実装が提供しなければならない主要関数ライブラリ（箇条 4 参照）を定義する。主要関数ライブラリの関数について、引数及び結果は、四つの基本的な型のいずれかをもち、XSLT 及び XPointer のどちらも、追加の関数を定義することによって、XPath を拡張する。これらの関数の幾つかは、四つの基本的な型を操作し、他の関数は XSLT 及び XPointer が定義する追加のデータ型を操作する。

名前空間宣言は、接頭辞から名前空間 URI への対応から成る。

部分式を評価するのに使う変数束縛、関数ライブラリ及び名前空間宣言は、部分式を含む式を評価するのに使うものと常に同じとする。部分式を評価するのに使う文脈ノード、文脈位置及び文脈サイズは、部分式を含む式を評価するのに使うものとは異なることがある。幾つかの種類の式は、文脈ノードを変更する。述部だけが、文脈位置及び文脈サイズを変更する（2.4 参照）。ある種類の式の評価が記述される場合、部分式の評価で文脈ノード、文脈位置及び文脈サイズが変わるとき、それは常に明示的に示される。文脈ノード、文脈位置及び文脈サイズについて何も示していない場合、その種類の式の部分式の評価では、これらは変更されない。

XPath 式は、XML の属性の中に現われることが多い。この規格で規定する文法は、XML 1.0 の正規化の後の属性値に適用する。したがって、例えば、文法が文字 “<” を用いるとき、これは XML ソースの中で “<” として現れてはならず、例えば < としてそれを入れることによって、XML 1.0 の規則に従って引用されなければならない。式の中では、リテラル列は一重引用符又は二重引用符によって区切られるが、それらは、XML の属性を区切るのにも使われる。式の中の引用符が、属性値を終端していると XML プロセッサによって解釈されることを避けるために、引用符は、文字参照（" 又は '）として入れることができる。これに代えて、XML の属性が二重引用符で区切られているときには、式は一重引用符を使うことができ、その逆もできる。

重要な種類の式の一つに、位置パスがある。位置パスは、文脈ノードに対して相対的に、ノードの集合を選ぶ。位置パスになっている式を評価した結果は、その位置パスによって選ばれるノードを含むノード集合とする。位置パスは、ノードの集合をフィルタリングするのに使われる式を再帰的に含むことができる。位置パスは、生成規則 LocationPath に一致する。

以下の文法において、非終端記号の QName 及び NCName は、JIS X 4158 で定義し、S は、JIS X 4159 で定義される。文法は、JIS X 4159 と同じ EBNF 記法を用いる。ただし、文法の記号は、常に先頭に大文字をもつことを除く。

式は、まず解析対象の文字列をトークンに分割し、それから結果のトークン列を解析することによって、構文解析される。トークンの間には空白を自由に使うことができる。トークン化処理は、3.7 に示す。

なお、便宜上、JIS X 4159 及び JIS X 4158、すなわち、XML 1.0 及び XML Names 1.0 への参照を常に使用する。しかし、実装は、JIS X 4159 及び JIS X 4158 の字句規定、又は[XML 1.1]及び[XML Names 1.1]の字句規定の採用を選択してよい。したがって、同様に、URI への参照が常に使用されるが、IRI を提供してもよい。XML 1.0 及び XML 1.1 の定義が、全く同じ場合もある。

2 位置パス

位置パスは、XPath 言語の中で最も一般的な文法的構成子ではない（LocationPath は、Expr の特別な場合である。）が、最も重要な構成子なので、最初にそれを示す。

位置パスはどれも、直接的ではあるが少しばかり面倒な構文を使って表現できる。共通の場合を簡潔に表現できる多くの構文的短縮形もある。箇条 2 は、非短縮構文を使って、位置パスの意味を説明する。その後で、短縮構文を、それがどのように非短縮構文に展開されるかを示すことによって規定する(2.5 参照)。

次に、非短縮構文を使った位置パスの例を示す。

例

- a) child::para は、文脈ノードの子要素 para を選択する。
- b) child::* は、文脈ノードの子要素のすべてを選択する。
- c) child::text() は、文脈ノードの子テキストノードのすべてを選択する。
- d) child::node() は、文脈ノードの子のすべてを、ノード型を問わずに選択する。
- e) attribute::name は、文脈ノードの name 属性を選択する。
- f) attribute::* は、文脈ノードのすべての属性を選択する。
- g) descendant::para は、文脈ノードの子孫要素 para を選択する。
- h) ancestor::div は、文脈ノードの祖先 div のすべてを選択する。
- i) ancestor-or-self::div は、文脈ノードの祖先 div を選択し、文脈ノードが div 要素である場合には、その文脈ノードも選択する。
- j) descendant-or-self::para は、文脈ノードの子孫要素 para を選択し、文脈ノードが para 要素である場合には、その文脈ノードも選択する。
- k) self::para は、文脈ノードが para 要素である場合にはその文脈ノードを選択し、そうでない場合には何も選択しない。
- l) child::chapter/descendant::para は、文脈ノードの子要素 chapter の子孫要素 para を選択する。
- m) child::* / child::para は、文脈ノードの孫 para のすべてを選択する。
- n) / は、文書のルート（常に文書要素の親）を選択する。
- o) /descendant::para は、文脈ノードと同じ文書にある para 要素のすべてを選択する。
- p) /descendant::olist / child::item は、親 olist をもち、文脈ノードと同じ文書にある item 要素のすべてを選択する。
- q) child::para[position()=1] は、文脈ノードの子の中の最初 para を選択する。

注記 以下では、これを“最初の子 para”などという。

- r) `child::para[position()=last()]`は、文脈ノードの最後の子 `para` を選択する。
- s) `child::para[position()=last()-1]`は、文脈ノードの最後から 2 番目の子 `para` を選択する。
- t) `child::para[position()>1]`は、文脈ノードの最初の子 `para` を除き、文脈ノードの子 `para` のすべてを選択する。
- u) `following-sibling::chapter[position()=1]`は、文脈ノードの次の兄弟 `chapter` を選択する。
- v) `preceding-sibling::chapter[position()=1]`は、文脈ノードの前の兄弟 `chapter` を選択する。
- w) `/descendant::figure[position()=42]`は、文書中の 42 番目の `figure` 要素を選択する。
- x) `/child::doc/child::chapter[position()=5]/child::section[position()=2]`は、`doc` 文書要素の 5 番目の `chapter` の 2 番目の `section` を選択する。
- y) `child::para[attribute::type="warning"]`は、値が `warning` である `type` 属性をもつ、文脈ノードの子 `para` のすべてを選択する。
- z) `child::para[attribute::type='warning'][position()=5]`は、値が `warning` の `type` 属性をもつ、文脈ノードの 5 番目の子 `para` を選択する。
- aa) `child::para[position()=5][attribute::type="warning"]`は、文脈ノードの 5 番目の子 `para` を、それが値 `warning` をとる `type` 属性をもつ場合に、選択する。
- ab) `child::chapter[child::title='Introduction']`は、文脈ノードの子 `chapter` で、文字列値（箇条 5 参照）が `Introduction` に等しい 1 個以上の子 `title` をもつものを選択する。
- ac) `child::chapter[child::title]`は、文脈ノードの子 `chapter` で、1 個以上の子 `title` をもつものを選択する。
- ad) `child::*[self::chapter or self::appendix]`は、文脈ノードの子 `chapter` 及び `appendix` を選択する。
- ae) `child::*[self::chapter or self::appendix][position()=last()]`は、文脈ノードの最後の子 `chapter` 又は `appendix` を選択する。

位置パスには、相対位置パス及び絶対位置パスの2種類がある。

相対位置パスは、“/”で分離された1個以上の位置パスの列から成る。相対位置パスの中のステップは、左から右へ一つに構成される。各ステップは順次、文脈ノードに対して相対的にノードの集合を選択する。ステップの初期列は、次のとおりに後続ステップとともに一つに構成される。ステップの初期列は、文脈ノードに対して相対的にノードの集合を選択する。その集合の各ノードは、次のステップの文脈ノードとして使う。そのステップによって特定されるノードの集合は、一つに統合される。ステップの構成によって特定されるノードの集合は、統合されて一つになる。例えば、`child::div/child::para` は、文脈ノードの子要素 `div` の子要素 `para` を選択する。言い換えると、親 `div` をもつ孫要素 `para` を選択する。

絶対位置パスは、“/” から成り、任意選択で相対位置パスが後続する。“/” はそれ自体で、文脈ノードを含む文書のルートノードを選択する。その後に相対位置パスが続く場合には、その位置パスは、文脈ノードを含む文書のルートノードに対して相対的な相対位置パスによって選択することになるノードの集合を選択する。

位置パスの構文を次に示す。

- ```

[1] LocationPath ::= RelativeLocationPath
 | AbsoluteLocationPath
[2] AbsoluteLocationPath ::= '/' RelativeLocationPath?
 | AbbreviatedAbsoluteLocationPath
[3] RelativeLocationPath ::= Step

```



| RelativeLocationPath '/' Step  
| AbbreviatedRelativeLocationPath

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|                                 |             |
|---------------------------------|-------------|
| LocationPath                    | 位置パス        |
| RelativeLocationPath            | 相対位置パス      |
| AbsoluteLocationPath            | 絶対位置パス      |
| AbbreviatedAbsoluteLocationPath | 短縮絶対位置パス    |
| Step                            | ステップ、位置ステップ |
| AbbreviatedRelativeLocationPath | 短縮相対位置パス    |

## 2.1 位置ステップ

位置ステップは、次の三つの部分をもつ。

- 位置ステップによって選択されるノードと文脈ノードとの木関係を指定する軸。
- 位置ステップによって選択されるノードのノード型及び展開名(箇条 5 参照)を指定するノード試験。
- 位置ステップによって選択されるノードの集合を更に洗練するために、任意の式を用いる 0 個以上の述部。

位置ステップの構文は、二重コロンによって分けられた軸の名前及びノード試験であり、それぞれ角括弧でくくられた 0 個以上の式が続く。例えば `child::para[position()=1]` では、`child` が軸の名前、`para` がノード試験、`[position()=1]` が述部になる。

位置ステップによって選択されるノード集合は、軸及びノード試験から初期ノード集合を生成し、その後そのノード集合を述部のそれぞれによって順次フィルタリングした結果のノード集合とする。

初期ノード集合は、軸によって指定される文脈ノードとの関係をもち、ノード試験によって指定するノード型及び展開名(箇条 5 参照)をもつノードから成る。例えば、位置ステップ `descendant::para` は、文脈ノードの子孫要素 `para` を選択する。ここで `descendant` は、初期ノード集合にある各ノードが文脈の子孫でなければならないことを指定し、`para` は、初期ノード集合にある各ノードが `para` と名付けられる要素でなければならないことを指定する。利用可能な軸は、2.2 で示す。利用可能なノード試験は、2.3 で示す。ノード試験には、その意味が軸に依存するものがある。

初期ノード集合は、最初の述部によってフィルタリングされて、新しいノード集合を生成する。その後この新しいノード集合は、2 番目の述部を使ってフィルタリングされ、以降同様とする。最終ノード集合は、位置ステップが選択するノード集合になる。軸は、各述部の式がどのように評価されるかに影響を与えるので、述部の意味は、軸に関して定義される(2.4 参照)。

位置ステップの構文を次に示す。

[4] Step ::= AxisSpecifier NodeTest Predicate\*  
| AbbreviatedStep  
[5] AxisSpecifier ::= AxisName ':'  
| AbbreviatedAxisSpecifier

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|               |             |
|---------------|-------------|
| Step          | ステップ、位置ステップ |
| AxisSpecifier | 軸、軸指定子      |
| NodeTest      | ノード試験       |
| Predicate     | 述部          |

|                                 |                  |
|---------------------------------|------------------|
| <u>AbbreviatedStep</u>          | 短縮ステップ, 短縮位置ステップ |
| <u>AxisName</u>                 | 軸名               |
| <u>AbbreviatedAxisSpecifier</u> | 短縮軸, 短縮軸指定子      |

## 2.2 軸

次に示す軸が利用できる。

- a) **child** 軸は、文脈ノードの子を含む。
- b) **descendant** 軸は、文脈ノードの子孫を含む。子孫とは、子、子の子、子の子の子など、以降同様とする。そのため、**descendant** 軸が属性ノード又は名前空間ノードを含むことは決してない。
- c) **parent** 軸は、文脈ノードの親（箇条 5 参照）が存在する場合、それを含む。
- d) **ancestor** 軸は、文脈ノードの祖先を含む。文脈ノードの祖先とは、文脈ノードの親、親の親、親の親の親など、以降同様とする。そのため、**ancestor** 軸は、文脈ノードがルートノードでない場合には、常にルートノードを含む。
- e) **following-sibling** 軸は、文脈ノードのすべての後続する兄弟を含む。文脈ノードが属性ノード又は名前空間ノードの場合、**following-sibling** 軸は空とする。
- f) **preceding-sibling** 軸は、文脈ノードのすべての先行する兄弟を含む。文脈ノードが属性ノード又は名前空間ノードの場合、**preceding-sibling** 軸は空とする。
- g) **following** 軸は、文書順で文脈ノードの後にある、文脈ノードと同じ文書の中のすべてのノードを含む。ただし、すべての子孫を除き、属性ノード及び名前空間ノードを除く。
- h) **preceding** 軸は、文書順で文脈ノードの前にある、文脈ノードと同じ文書の中のすべてのノードを含む。ただし、すべての祖先を除き、属性ノード及び名前空間ノードを除く。
- i) **attribute** 軸は、文脈ノードの属性を含む。文脈ノードが要素でない場合には、軸は空になる。
- j) **namespace** 軸は、文脈ノードの名前空間ノードを含む。文脈ノードが要素でない場合には、軸は空になる。
- k) **self** 軸は、（現在の）文脈ノードそれ自体だけを含む。
- l) **descendant-or-self** 軸は、文脈ノード及び文脈ノードの子孫を含む。
- m) **ancestor-or-self** 軸は、文脈ノード及び文脈ノードの祖先を含む。したがって、**ancestor-or-self** 軸は、常にルートノードを含む。

**注記** **ancestor**, **descendant**, **following**, **preceding** 及び **self** の軸は、（属性ノード及び名前空間ノードを無視して）文書を区分する。これらは重なり合うことはなく、一緒になって文書中のすべてのノードを含む。

軸の構文を次に示す。

```
[6] AxisName ::= 'ancestor'
 | 'ancestor-or-self'
 | 'attribute'
 | 'child'
 | 'descendant'
 | 'descendant-or-self'
 | 'following'
 | 'following-sibling'
 | 'namespace'
```

|  |                     |
|--|---------------------|
|  | 'parent'            |
|  | 'preceding'         |
|  | 'preceding-sibling' |
|  | 'self'              |

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|          |    |
|----------|----|
| AxisName | 軸名 |
|----------|----|

### 2.3 ノード試験

どの軸にも主ノード型がある。軸が要素を含むことができる場合、主ノード型は要素とする。そうではない場合、軸が含むことができるノードの型とする。したがって、次のとおりとなる。

- a) attribute 軸については、主ノード型は、attribute とする。
- b) namespace 軸については、主ノード型は、namespace とする。
- c) その他の軸については、主ノード型は、element とする。

QName であるノード試験が真であるのは、ノードの型(箇条 5 参照)が、主ノード型であって、QName によって指定される展開名に等しい展開名をもつ場合に限る。例えば、child::para は、文脈ノードの子要素 para を選択する。文脈ノードが子要素 para をもたない場合は、ノードの空集合を選択することになる。attribute::href は、文脈ノードの href 属性を選択する。文脈ノードが href 属性をもたない場合は、ノードの空集合を選択することになる。

ノード試験の中の QName は、式の文脈からの名前空間宣言を使って展開名に展開される。これは、xmlns で宣言される既定の名前空間が使われないことを除いて、開始タグ及び終了タグにおける要素型名について展開がなされるのと同じ方法とする。QName が接頭辞をもたないとき、名前空間 URI はヌルとする。これは属性名が展開されるのと同じ方法とする。QName が、式の文脈の中で名前空間宣言のない接頭辞をとる場合には、誤りとする。

ノード試験 \* は、主ノード型のどのノードについても真とする。例えば、child::\* は文脈ノードのすべての子要素を選択することになり、attribute::\* は文脈ノードのすべての属性を選択することになる。

ノード試験は、NCName:\* という形式をもつことができる。この場合、接頭辞は、QName をもつ場合と同じ方法で、文脈名前空間宣言を使って展開される。式の文脈の中の接頭辞について、名前空間宣言がない場合には、誤りとする。ノード試験は、名前の局所部分とは無関係に、展開名が接頭辞の展開先の名前空間 URI をもつ主要な型のどのノードについても真になる。

ノード試験 text() は、どのテキストノードについても真とする。例えば、child::text() は、文脈ノードの子テキストノードを選択することになる。同様に、ノード試験 comment() は、どのコメントノードについても真とし、ノード試験 processing-instruction() はどの処理命令についても真とする。processing-instruction() 試験は、Literal である引数をもってもよい。この場合、Literal の値に等しい名前をもつどの処理命令についても真とする。

ノード試験 node() は、どの型をもつどのノードについても真とする。

ノード試験の構文を次に示す。

```
[7] NodeTest ::= NameTest
 | NodeType '(' ')'
 | 'processing-instruction' '(' Literal ')'
```

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|          |       |
|----------|-------|
| NodeTest | ノード試験 |
|----------|-------|

|                       |      |
|-----------------------|------|
| <u>NameTest</u> ..... | 名前試験 |
| <u>NodeType</u> ..... | ノード型 |
| <u>Literal</u> .....  | リテラル |

## 2.4 述部

軸は、順方向軸か逆方向軸かのどちらかとする。文脈ノード、又は文書順（箇条 5 参照）に文脈ノードの後にあるノードだけしか含まない軸は、順方向軸とする。文脈ノード、又は文書順に文脈ノードの前にあるノードだけしか含まない軸は、逆方向軸とする。したがって、**ancestor** 軸、**ancestor-or-self** 軸、**preceding** 軸、**preceding-sibling** 軸は、逆方向軸になる。その他の軸は、すべて順方向軸になる。ただし、**self** 軸は常に最大で 1 個のノードしか含まないので、順方向軸でも逆方向軸でも差異はない。軸に関するノード集合のメンバの近接度位置(**proximity position**)は、軸が順方向軸の場合は文書順に、逆方向軸の場合は逆文書順に、順序付けられたノード集合におけるそのノードの位置と定義する。最初の位置は、1 とする。

述部は、軸に関してノード集合をフィルタリングして、新しいノード集合を作る。フィルタリングされるノード集合の中の各ノードに対して、**PredicateExpr** は、そのノードを文脈ノードとし、そのノード集合のノード数を文脈サイズとし、軸に関してノード集合におけるノードの近接度位置を文脈位置として、評価される。**PredicateExpr** がそのノードについて真と評価される場合、そのノードは、新しいノード集合に含まれる。そうでない場合には、それは含まれない。

**PredicateExpr** は、**Expr** を評価してその結果を論理型に変換することによって、評価される。結果が数値型である場合には、その結果は、数値型が文脈位置に等しい場合には真に変換され、それ以外の場合には偽に変換される。結果が数値型でない場合には、結果は、**boolean** 関数への呼出しによるものとして変換される。したがって、**para[3]** という位置パスは、**para[position()=3]** と等価になる。

述部の構文を次に示す。

- [8]     **Predicate**            ::= '[ PredicateExpr ]'
- [9]     **PredicateExpr**       ::= **Expr**

注記 構文規則内の構成子は、（存在する場合には）本文中の次の日本語と対応する。

|                            |     |
|----------------------------|-----|
| <u>Predicate</u> .....     | 述部  |
| <u>PredicateExpr</u> ..... | 述部式 |
| <u>Expr</u> .....          | 式   |

## 2.5 短縮構文

短縮構文を使った位置パスの例を次に示す。

### 例

- a) **para** は、文脈ノードの子要素 **para** を選択する。
- b) **\*** は、文脈ノードの子要素のすべてを選択する。
- c) **text()** は、文脈ノードの子テキストノードのすべてを選択する。
- d) **@name** は、文脈ノードの **name** 属性を選択する。
- e) **@\*** は、文脈ノードのすべての属性を選択する。
- f) **para[1]** は、文脈ノードの最初の子 **para** を選択する。
- g) **para[last()]** は、文脈ノードの最後の子 **para** を選択する。
- h) **\*/para** は、文脈ノードの孫 **para** のすべてを選択する。
- i) **/doc/chapter[5]/section[2]** は、**doc** の 5 番目の **chapter** の 2 番目の **section** を選択する。
- j) **chapter//para** は、文脈ノードの子要素 **chapter** の子孫要素 **para** を選択する。

- k) `//para` は、文書ルートの子孫 `para` のすべてを選択し、したがって、文脈ノードと同じ文書の中の `para` 要素のすべてを選択する。
- l) `//olist/item` は、親 `olist` をもち、文脈ノードと同じ文書にある `item` 要素のすべてを選択する。
- m) `.` (ドット一つ) は、文脈ノードを選択する。
- n) `./para` は、文脈ノードの子孫要素 `para` を選択する。
- o) `..` (ドット二つ) は、文脈ノードの親を選択する。
- p) `./@lang` は、文脈ノードの親の `lang` 属性を選択する。
- q) `para[@type="warning"]` は、値 `warning` をとる `type` 属性をもつ、文脈ノードの子 `para` のすべてを選択する。
- r) `para[@type="warning"][5]` は、値 `warning` をとる `type` 属性をもつ、文脈ノードの 5 番目の子 `para` を選択する。
- s) `para[5][@type="warning"]` は、文脈ノードの 5 番目の子 `para` を、それが値 `warning` をとる `type` 属性をもつときに、選択する。
- t) `chapter[title="Introduction"]` は、文字列値 (箇条 5 参照) が `Introduction` に等しい 1 個以上の子 `title` をもつ、文脈ノードの子 `chapter` を選択する。
- u) `chapter[title]` は、1 個以上の子 `title` をもつ、文脈ノードの子 `chapter` を選択する。
- v) `employee[@secretary and @assistant]` は、文脈ノードの子 `employee` で、`secretary` 属性及び `assistant` 属性の両方をもつものすべてを選択する。

最も重要な短縮形は、`child::` を位置ステップから省けることとする。実効的には、`child` は既定の軸とする。例えば、`div/para` という位置パスは、`child::div/child::para` に関する短縮になる。

属性に関する短縮形もある。`attribute::` は、`@` に短縮できる。例えば、`para[@type="warning"]` という位置パスは、`child::para[attribute::type="warning"]` の短縮であって、`warning` に等しい値をとる `type` 属性をもつ子 `para` を選択する。

`//` は、`/descendant-or-self::node()` の短縮とする。例えば、`//para` は、`/descendant-or-self::node()/child::para` の短縮であるので、文書中のどの `para` 要素をも選択する。文書要素ノードはルートノードの子であるので、文書要素である `para` 要素でさえ `//para` によって選択されることになる。`div//para` は、`child::div/descendant-or-self::node()/child::para` の短縮であるので、子 `div` の子孫 `para` のすべてを選択することになる。

**注記** `//para[1]` という位置パスは、位置パス `/descendant::para[1]` と同じものを意味しない。後者は、最初の子孫要素 `para` を選択する。前者は、その親の最初の子 `para` であるような子孫要素 `para` のすべてを選択する。

`.` (ドット) という位置ステップは、`self::node()` の短縮とする。これは、`//` との組合せで特に有用である。例えば、`//para` という位置パスは、`self::node()/descendant-or-self::node()/child::para` の短縮であるので、文脈ノードの子孫要素 `para` のすべてを選択することになる。

同様に、`..` という位置ステップは、`parent::node()` の短縮とする。例えば、`../title` は、`parent::node()/child::title` の短縮であるので、文脈ノードの親の子 `title` を選択することになる。

短縮形の構文を次に示す。

- [10] `AbbreviatedAbsolutePath ::= '/' RelativeLocationPath`
- [11] `AbbreviatedRelativeLocationPath ::= RelativeLocationPath '/' Step`
- [12] `AbbreviatedStep ::= '.'`

| '...'

[13] AbbreviatedAxisSpecifier ::= '@'?

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|                                 |                  |
|---------------------------------|------------------|
| AbbreviatedAbsolutePath         | 短縮絶対位置パス         |
| RelativeLocationPath            | 相対位置パス           |
| AbbreviatedRelativeLocationPath | 短縮相対位置パス         |
| Step                            | ステップ, 位置ステップ     |
| AbbreviatedStep                 | 短縮ステップ, 短縮位置ステップ |
| AbbreviatedAxisSpecifier        | 短縮軸, 短縮軸指定子      |

### 3 式

#### 3.1 基本

VariableReference は、文脈における変数束縛の集合の中で、変数名が束縛されている値に評価される。変数名が、式の文脈における変数束縛の集合の中で、いかなる値にも束縛されていない場合には、誤りとする。

括弧は、グループ化を行うために使用してよい。

式の構文を次に示す。

[14] Expr ::= OrExpr

[15] PrimaryExpr ::= VariableReference

| '(' Expr ')'

| Literal

| Number

| FunctionCall

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|                   |            |
|-------------------|------------|
| Expr              | 式          |
| OrExpr            | or 式       |
| PrimaryExpr       | 基本式        |
| VariableReference | 変数参照       |
| Literal           | リテラル       |
| Number            | 数, 数値, 数値型 |
| FunctionCall      | 関数呼出し      |

#### 3.2 関数呼出し

FunctionCall 式は、式評価文脈関数ライブラリの中の関数を識別するために FunctionName を使用し、Argument の各々を評価し、各引数を関数が要求する型に変換し、最後に、関数を呼び出してその変換された引数を関数に渡すことによって、評価される。引数の個数が間違っている場合、又は引数が要求される型に変換できない場合には、誤りとする。FunctionCall 式の結果は、呼び出される関数が返す結果とする。

引数が文字列型に変換される場合には、string 関数の呼出しによって行われるのと同様に変換される。引数が数値型に変換される場合には、number 関数の呼出しによって行われるのと同様に変換される。引数が論理型に変換される場合には、boolean 関数の呼出しによって行われるのと同様に変換される。ノード集合型ではない引数は、ノード集合に変換できない。

関数呼出しの構文を次に示す。

[16] `FunctionCall` ::= `FunctionName '(' ( Argument ( ' Argument )' )* )? ' )'`

[17] `Argument` ::= `Expr`

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|                           |       |
|---------------------------|-------|
| <code>FunctionCall</code> | 関数呼出し |
| <code>FunctionName</code> | 関数名   |
| <code>Argument</code>     | 引数    |
| <code>Expr</code>         | 式     |

### 3.3 ノード集合型 (node-set)

位置パスは、式として使用できる。その式は、パスによって選択されるノードの集合を返す。

| 演算子は、その演算対象（オペランド）の和集合を計算するが、演算対象は、ノード集合でなければならない。

`Predicate` は、位置パスで使用するのと同じ方法で、式をフィルタリングするために使用する。フィルタリングされる式がノード集合へと評価されない場合には、誤りとする。`Predicate` は、子軸に関してノード集合をフィルタリングする。

注記 `Predicate` の意味は、どの軸を適用するかに決定的に依存する。例えば、`preceding::foo[1]` は、逆文書順で最初の `foo` 要素を返す。これは、`[1]` という述部に適用される軸が先行する軸になることによる。反対に、`(preceding::foo)[1]` は、文書順で最初の `foo` 要素を返す。これは、`[1]` という述部に適用される軸が子軸になることによる。

/ 演算子及び // 演算子は、式及び相対的位置パスを構成する。式がノード集合へと評価されない場合、誤りとする。/ 演算子は、/ を位置パスで使用するのと同じ方法で構成を行う。位置パスにおけるのと同様に、// は、`/descendant-or-self::node()` の短縮形とする。

ノード集合へと変換可能なオブジェクトの型は存在しない。

ノード集合の式の構文を次に示す。

[18] `UnionExpr` ::= `PathExpr`  
| `UnionExpr` `'` `PathExpr`

[19] `PathExpr` ::= `LocationPath`  
| `FilterExpr`  
| `FilterExpr` `'/'` `RelativeLocationPath`  
| `FilterExpr` `'//'` `RelativeLocationPath`

[20] `FilterExpr` ::= `PrimaryExpr`  
| `FilterExpr` `Predicate`

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

|                                   |        |
|-----------------------------------|--------|
| <code>UnionExpr</code>            | 合併式    |
| <code>PathExpr</code>             | パス式    |
| <code>LocationPath</code>         | 位置パス   |
| <code>FilterExpr</code>           | フィルタ式  |
| <code>RelativeLocationPath</code> | 相対位置パス |
| <code>PrimaryExpr</code>          | 基本式    |
| <code>Predicate</code>            | 述部     |

### 3.4 論理型 (boolean)

論理型のオブジェクトは、二つの値、真(true)及び偽(false)、の一つをもつことができる。

or 式は、各々の演算対象を評価し、その値を boolean 関数への呼出しを行ったものとして論理型に変換することによって評価する。結果は、(演算対象の) いずれかの値が真の場合に真とし、そうでない場合は偽とする。右側の演算対象は、左側の演算対象が真と評価される場合には、評価しない。

and 式は、各々の演算対象を評価し、その値を boolean 関数への呼出しを行ったものとして論理型に変換することによって評価する。結果は、(演算対象の) 両方の値が真の場合に真とし、そうでない場合は偽とする。右側の演算対象は、左側の演算対象が偽と評価される場合には、評価しない。

(RelationalExpr ではない) EqualityExpr 又は (AdditiveExpr ではない) RelationalExpr は、二つの演算対象を評価した結果のオブジェクトを比較することによって評価する。結果のオブジェクトの比較は、次の三つの段落で定義する。最初に、ノード集合を含む比較をノード集合を含まない比較を用いて定義する。これは、=, !=, <=, <, >= 及び > に対して統一的に定義する。2 番目に、ノード集合を含まない比較を = 及び != に対して定義する。3 番目に、ノード集合を含まない比較を <=, <, >= 及び > に対して定義する。

比較する両方のオブジェクトがノード集合の場合、その比較は、あるノードが最初のノード集合の中に存在し、更に、あるノードが 2 番目のノード集合の中にも存在して、その二つのノードの文字列値 (箇条 5 参照) に関して比較を実行した結果が、真の場合に限り、真とする。比較する一つのオブジェクトがノード集合であって他方が数値の場合、その比較は、あるノードがノード集合の中に存在して、そのノードの文字列値を number 関数を用いて数値に変換した結果と、それと比較する数値に関する比較を実行した結果とが、真の場合に限り、真とする。比較するオブジェクトの一つがノード集合であって他方が文字列の場合、その比較は、あるノードがノード集合の中に存在し、そのノードの文字列値及び他方の文字列に関して比較を実行した結果が、真の場合に限り、真とする。比較する一つのオブジェクトがノード集合であって他方が論理型の場合、その比較は、その論理型と、ノード集合を boolean 関数を用いて論理型に変換した結果とに関して比較を実行した結果が、真の場合に限り、真とする。

比較するオブジェクトのいずれもがノード集合ではなく、演算子が = 又は != の場合、それらのオブジェクトは、次に示すとおり共通型に変換しその後比較することによって比較する。比較するオブジェクトの少なくとも一つが論理型の場合、比較する各々のオブジェクトは、boolean 関数を適用するものとして論理型に変換する。そうでない場合であって、比較するオブジェクトの少なくとも一つが数値の場合、比較する各々のオブジェクトは、number 関数を適用するものとして数値に変換する。そうでない場合、比較するオブジェクトの両方は、string 関数を適用するものとして文字列に変換する。= の比較は、オブジェクトが等しい場合に限り、真とする。!= の比較は、オブジェクトが等しくない場合に限り、真とする。数値は、IEEE 754 [IEEE 754]に従う等しさで比較する。二つの論理型は、両方が真又は両方が偽のいずれかの場合に限り、等しい。二つの文字列は、それらが UCS 文字の同じ列から成る場合に限り、等しい。

**注記** \$x がノード集合に束縛されている場合、\$x="foo" は、not(\$x!="foo") と同じことを意味しない。

\$x="foo" は、\$x の中のあるノードが文字列値 foo をもつ場合に限り、真になる。not(\$x!="foo")

は、\$x の中のすべてのノードが文字列値 foo をもつ場合に限り、真になる。

比較するオブジェクトのいずれもがノード集合でなく、演算子が <=, <, >= 又は > の場合、それらのオブジェクトは、両方を数値に変換し IEEE 754 に従って数値を比較することによって比較する。< の比較は、最初の数値が 2 番目の数値よりも小さい場合に限り、真とする。<= の比較は、最初の数値が 2 番目の数値以下の場合に限り、真とする。> の比較は、最初の数値が 2 番目の数値よりも大きい場合に限り、



真とする。>= の比較は、最初の数値が 2 番目の数値以上の場合に限り、真とする。

**注記** XPath 式が XML 文書に現れる場合、< 及び <= 演算子は、XML 1.0 の規則に従って、例えば、&lt; 及び &lt;= を用いて引用されなければならない。次の例では、test 属性の値が、XPath 式になっている。

```
<xsl:if test="@value < 10">...</xsl:if>
```

論理型の式の構文を次に示す。

- [21] OrExpr ::= AndExpr  
                  | OrExpr 'or' AndExpr
- [22] AndExpr ::= EqualityExpr  
                  | AndExpr 'and' EqualityExpr
- [23] EqualityExpr ::= RelationalExpr  
                  | EqualityExpr '=' RelationalExpr  
                  | EqualityExpr '!=' RelationalExpr
- [24] RelationalExpr ::= AdditiveExpr  
                  | RelationalExpr '<' AdditiveExpr  
                  | RelationalExpr '>' AdditiveExpr  
                  | RelationalExpr '<=' AdditiveExpr  
                  | RelationalExpr '>=' AdditiveExpr

**注記 1** この文法による効果は、次に示す優先順位の順序による。優先順位の順序は、より低い優先順位のことを先（上及び左）に示してある。

```
or
and
=, !=
<=, <, >=, >
```

さらに、演算子は、すべて、左結合とする。例えば、 $3 > 2 > 1$  は、 $(3 > 2) > 1$  と等価であって、偽と評価する。

**注記 2** 構文規則内の構成子は、（存在する場合には）本文中の次の日本語と対応する。

|                      |        |
|----------------------|--------|
| OrExpr               | or 式   |
| AndExpr              | and 式  |
| EqualityExpr         | 等号類式   |
| RelationalExpr       | 関係式    |
| RelativeLocationPath | 相対位置パス |
| AdditiveExpr         | 加減式    |

### 3.5 数値型 (number)

数値は、浮動小数点数値を表現する。数値は、倍精度 64 ビット形式の IEEE 754 値[IEEE 754]をもつことができる。これらには、“非数” (Not-a-Number, NaN) 値、正の無限大及び負の無限大、並びに正の 0 及び負の 0 を含む。IEEE 754 規定の主要な要約については、[JLS]の 4.2.3 を参照する。

数値演算子は、その演算対象を、number 関数の呼出しを行うものとして、数値に変換する。

+ 演算子は、加算を実行する。

2 項の - 演算子は、減算を実行する。単項の - 演算子は、負数を示す。-0 は、負の 0 と評価されるこ

とに注意。

**注記** XML は名前の中に - を許しているので、通常、- 演算子は、空白によって先行される必要がある。例えば、foo-bar は、foo-bar と名前が付けられた子要素を含むノード集合に評価される。

foo - bar は、最初の foo 子要素の文字列値を数値に変換した結果と、最初の bar 子要素の文字列値を数値に変換した結果との差に評価する。

\* 演算子は、IEEE 754 に従った浮動小数点の乗算を実行する。結果が NaN でない場合、演算対象の両方が同じ符号をもつとき及びそのときに限り、結果が正となることに注意。

div 演算子は、IEEE 754 に従った浮動小数点の除算を実行する。結果が NaN でない場合、演算対象の両方が同じ符号をもつとき及びそのときに限り、結果が正となることに注意。

mod 演算子は、切捨て(truncating)除算からの剰余を返す。次に例を示す。

**例**

5 mod 2 は、1 を返す。

5 mod -2 は、1 を返す。

-5 mod 2 は、-1 を返す。

-5 mod -2 は、-1 を返す。

**注記 1** これは、Java 及び ECMAScript における % 演算子と同じになっている。

**注記 2** これは、IEEE 754 の剰余演算子とは同じではない。IEEE 754 の剰余演算子は、丸め(rounding)除算からの剰余を返す。

数値型の式の構文を次に示す。

- ```
[25]  AdditiveExpr      ::=  MultiplicativeExpr
                                |  AdditiveExpr '+' MultiplicativeExpr
                                |  AdditiveExpr '-' MultiplicativeExpr

[26]  MultiplicativeExpr ::=  UnaryExpr
                                |  MultiplicativeExpr MultiplyOperator UnaryExpr
                                |  MultiplicativeExpr 'div' UnaryExpr
                                |  MultiplicativeExpr 'mod' UnaryExpr

[27]  UnaryExpr        ::=  UnionExpr
                                |  '-' UnaryExpr
```

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

AdditiveExpr	加減式
MultiplicativeExpr	乗除式
UnaryExpr	単項式
MultiplyOperator	乗算演算子

3.6 文字列型 (string)

文字列は、0 個以上の文字の列から成る。ここで、文字とは、XML の規格(JIS X 4159)のとおり定義される。したがって、XPath における単一の文字とは、単一の対応する Unicode スカラ値をもつ単一の Unicode 抽象文字([Unicode]参照)に相当する。これは、16 ビットの Unicode 符号値と同じではない。U+FFFF よりも大きな Unicode スカラ値をもつ抽象文字のための Unicode 符号化文字表現は、16 ビットの Unicode 符号値の対(サロゲートペア, surrogate pair)になる。多くのプログラム言語では、文字列は、16 ビットの Unicode 符号値の列によって表現される。それらの言語での XPath の実装は、サロゲートペアが単一の

XPath 文字として正しく取り扱われることを確実にを行うように注意しなければならない。

注記 Unicode では、Unicode 抽象文字の異なる列から成るにもかかわらず、同一として取り扱われることが望ましい二つの文字列が存在する可能性がある。例えば、幾つかのアクセント付き文字は、事前に構成した形式又は分解した形式のいずれかで表現してよい。そのために、XPath 式における文字及び XML 文書における文字の両方が正準形式に正規化していない場合には、XPath 式は、期待しない結果を返すことがある ([Character Model]参照)。

3.7 字句構造

トークン化を行う場合、最も長くて解釈が可能なトークンを常に返す。

読みやすさのために、空白は、文法によって明示的には許されていない場合であっても、式の中で使用してもよい。すなわち、ExprWhitespace を、ExprToken の前後の式内に自由に追加してもよい。ただし、ExprToken の中間には追加してはならない。

ExprToken 文法をあいまいにしないために、次の特殊なトークン化規則を指定した順序で適用しなければならない。

- a) 先行するトークンが存在し、その先行するトークンが @, ::, (, [, , (コンマ) 又は Operator の一つではない場合、* は、MultiplyOperator として認識しなければならない、NCName は、OperatorName として認識しなければならない。
- b) 挿入されているかもしれない ExprWhitespace の後の QName に続く文字が ((1 バイト文字符号の開き丸括弧) となる場合、そのトークンは、NodeType 又は FunctionName として認識しなければならない。
- c) 挿入されているかもしれない ExprWhitespace の後の NCName に続く二つの文字が :: となる場合、そのトークンは、AxisName として認識しなければならない。
- d) そうでない場合には、トークンを、MultiplyOperator, OperatorName, NodeType, FunctionName 又は AxisName として認識してはならない。

呼出し側の言語は、JIS X 4159 及び JIS X 4158 の字句規則に従うか、又はその代わりに[XML 1.1]及び[XML Names 1.1]の字句規則に従うかを選択してよい。式の字句構造を次に示す。

```
[28] ExprToken ::= '(' | ')' | '[' | ']' | ':' | '.' | ',' | '@' | ';' | ':'
                | NameTest
                | NodeType
                | Operator
                | FunctionName
                | AxisName
                | Literal
                | Number
                | VariableReference
```

```
[29] Literal ::= '"' ['^']* '"'
                | "'" ['^']* "'"
```

```
[30] Number ::= Digits ('.' Digits)?
                | '!' Digits
```

```
[31] Digits ::= [0-9] +
```

```
[32] Operator ::= OperatorName
```

- | MultiplyOperator
| '/' | '//' | '|' | '+' | '-' | '=' | '!=' | '<' | '<=' | '>' | '>='
- [33] OperatorName ::= 'and' | 'or' | 'mod' | 'div'
- [34] MultiplyOperator ::= '*'
- [35] FunctionName ::= QName - NodeType
- [36] VariableReference ::= '\$' QName
- [37] NameTest ::= '*'
| NCName ':' '*'
| QName
- [38] NodeType ::= 'comment'
| 'text'
| 'processing-instruction'
| 'node'
- [39] ExprWhitespace ::= S (JIS X 4159 で定義。)

注記 構文規則内の構成子は、(存在する場合には)本文中の次の日本語と対応する。

ExprToken	式トークン
NameTest	名前試験
NodeType	ノード型
Operator	演算子
FunctionName	関数名
AxisName	軸名
Literal	リテラル
Number	数, 数値, 数値型
VariableReference	変数参照
Digits	数字
OperatorName	演算子名
MultiplyOperator	乗算演算子
QName	修飾名
NCName	名前空間名
ExprWhitespace	式空白

4 主要関数ライブラリ

箇条 4 は、XPath 実装が式を評価するために使用する関数ライブラリの中に常に含めなければならない関数を定義する。

関数ライブラリの中の各関数は、関数の原型(prototype)を用いて指定する。この原型は、返却型、関数の名前及び引数型を与える。引数型の後に疑問符(“?”)がある場合には、その引数は任意選択とする。そうでない場合には、その引数は必ず(須)とする。

4.1 ノード集合関数

関数 数値型 last()

last 関数は、式評価文脈から文脈サイズに等しい数値を返す。

関数 数値型 position()

position 関数は、式評価文脈から文脈位置に等しい数値を返す。

関数 数値型 count(ノード集合型)

count 関数は、引数であるノード集合の中のノード数を返す。

関数 ノード集合型 id(オブジェクト型)

id 関数は、一意 ID (5.2.1 参照) によって要素を選択する。id への引数がノード集合型の場合、結果は、引数であるノード集合の中の各ノードの文字列値に id を適用した結果の和集合とする。id への引数がそれ以外の型の場合、その引数は、string 関数への呼出しによるものとして文字列に変換する。その文字列は、空白で区切られたトークンのリストに分ける。空白は、JIS X 4159 の生成規則 S に、JIS X 4159 で定義するとおりにマッチする文字の列とする。結果は、リストの中のトークンに等しい一意 ID をもつ要素であって、文脈ノードと同じ文書の中にある要素を含むノード集合とする。

a) id("foo") は、一意 ID foo をもつ要素を選択する。

b) id("foo")/child::para[position()=5] は、一意 ID foo をもつ要素の子である 5 番目の para を選択する。

関数 文字列型 local-name(ノード集合型 ?)

local-name 関数は、引数であるノード集合の中の文書順 (箇条 5 参照) で最初のノードの展開名の局所部分を返す。引数のノード集合が空か、又は最初のノードが展開名をもたない場合には、空文字列が返される。引数が省略された場合には、文脈ノードを唯一のメンバとするノード集合を既定の引数として用いる。

関数 文字列型 namespace-uri(ノード集合型 ?)

namespace-uri 関数は、引数であるノード集合の中の文書順で最初のノードの展開名の名前空間 URI を返す。引数のノード集合が空、最初のノードが展開名をもたない、又は展開名の名前空間 URI がヌルの場合には、空文字列が返される。引数が省略された場合には、文脈ノードを唯一のメンバとするノード集合を既定の引数として用いる。

注記 namespace-uri 関数が返す文字列は、要素ノード及び属性ノード以外については空になる。

関数 文字列型 name(ノード集合型 ?)

name 関数は、引数であるノード集合の中の文書順で最初のノードの展開名を表現する QName を含む文字列を返す。QName は、展開名で表現するノード上の実効的な名前空間宣言に関するその展開名を表現しなければならない。通常は、これは、XML ソースに出現した QName になる。複数の接頭辞を同じ名前空間に関連付ける複数の名前空間宣言がそのノード上に存在する場合には、そうなる必要はない。しかし、実装は、ノードのその表現の中に元々の接頭辞についての情報を含めてもよい。この場合、実装は、返す文字列が常に XML ソースの中で使用している QName と確実に同じになるようにできる。引数であるノード集合が空か、又は最初のノードが展開名をもたない場合、空文字列を返す。引数を省略した場合には、文脈ノードを唯一のメンバとするノード集合を既定の引数として用いる。

注記 name 関数が返す文字列は、要素ノード及び属性ノード以外については、local-name 関数が返す文字列と同じになる。

4.2 文字列型関数**関数 文字列型 string(オブジェクト型 ?)**

string 関数は、次のとおりに、オブジェクトを文字列に変換する。

- a) ノード集合は、そのノード集合の中の文書順で最初のノードの文字列値を返すことによって、文字列に変換する。ノード集合が空の場合には、空文字列を返す。
- b) 数値は、次の文字列に変換する。
 - 1) NaN は、文字列 NaN に変換する。
 - 2) 正の 0 は、文字列 0 に変換する。
 - 3) 負の 0 は、文字列 0 に変換する。
 - 4) 正の無限大は、文字列 Infinity に変換する。
 - 5) 負の無限大は、文字列 -Infinity に変換する。
 - 6) 数値が整数の場合、その数値は、10 進小数点も先行する 0 もない Number として、10 進形式で表現する。数値が負の場合には、マイナス記号 (-) を最初に付ける。
 - 7) そうでない場合には、数値は、10 進小数点の前に少なくとも一つの数字をもち、10 進小数点の後に少なくとも一つの数字をもつ 10 進小数点を含んだ Number として、10 進形式で表現する。数値が負の場合には、マイナス記号 (-) を最初に付ける。10 進小数点の直前の必ず (須) の一つの数字以外は、10 進小数点の前に先行する 0 が存在してはならない。10 進小数点の後の必ず (須) の一つの数字以外に、その数値をすべての他の IEEE 754 数値から一意に区別するために必要とする、より多くの、しかし必要な個数だけの数字が存在しなければならない。
- c) 論理型の偽の値は、文字列 false に変換する。論理型の真の値は、文字列 true に変換する。
- d) これらの基本型以外の型のオブジェクトは、その型に依存する方法で、文字列に変換する。引数を省略した場合には、文脈ノードを唯一のメンバとするノード集合を既定の引数として用いる。

注記 string 関数は、利用者への表示のために数値を文字列に変換することを意図してはいない。

JIS X 4169 における format-number 関数及び xsl:number 要素が、この機能を提供する。

関数 文字列型 concat(文字列型, 文字列型, 文字列型*)

concat 関数は、その引数の (文字列の) 連結を返す。

注記 concat 関数における記法 “*” の意味は、W3C の XPath の原勧告及び Errata において定義されていない。XPath の開発者、利用者、既存の XPath の処理系などから調査した結果では、3 番目以降の引数として“文字列型”を 0 個以上とってよいことを示すと思われる。例えば、五つの文字列を連結したい場合には、concat に五つの文字列型の引数を与えてよいことを示す。最初及び 2 番目の引数には “*” が付いていないことに注意する。すなわち concat は、少なくとも二つの引数をもつ。

関数 論理型 starts-with(文字列型, 文字列型)

starts-with 関数は、最初の引数の文字列が 2 番目の文字列の引数で開始する場合に、真を返す。そうでない場合に、偽を返す。2 番目の引数の文字列が空文字列の場合、真を返す。

関数 論理型 contains(文字列型, 文字列型)

contains 関数は、最初の引数の文字列が 2 番目の引数の文字列を含む場合に、真を返す。そうでない場合に、偽を返す。2 番目の引数の文字列が空文字列の場合、真を返す。

関数 文字列型 substring-before(文字列型, 文字列型)

substring-before 関数は、最初の引数の文字列の中における 2 番目の引数の文字列の最初の出現に先行する最初の引数の文字列の部分文字列を返す。最初の引数の文字列が 2 番目の引数の文字列を含まない場合には、空文字列を返す。例えば、substring-before("1999/04/01","/") は、1999 を返す。2 番目の引数の文字列が空文字列の場合、空文字列を返す。

関数 文字列型 substring-after(文字列型, 文字列型)

substring-after 関数は、最初の引数の文字列の中における 2 番目の引数の文字列の最初の出現に続く最初の引数の文字列の部分文字列を返す。最初の引数の文字列が 2 番目の引数の文字列を含まない場合には、空文字列を返す。例えば、substring-after("1999/04/01","/") は、"04/01" を返し、substring-after("1999/04/01","19") は、"99/04/01" を返す。2 番目の引数の文字列が空文字列の場合、最初の引数の文字列を返す。

関数 文字列型 substring(文字列型, 数値型, 数値型 ?)

substring 関数は、2 番目の引数で指定した位置から開始する 3 番目の引数で指定した長さをもつ最初の引数の文字列の部分文字列を返す。例えば、substring("12345",2,3) は、"234" を返す。3 番目の引数を指定しない場合には、2 番目の引数で指定した位置から開始しその文字列の最後まで続く（最初の引数の文字列の）部分文字列を返す。例えば、substring("12345",2) は、"2345" を返す。

より正確には、文字列（3.6 参照）における各々の文字は、一つの数値的な位置をもつ。すなわち、最初の文字の位置は 1 とし、2 番目の文字の位置は 2 とし、以下同様とする。

注記 これは、Java 及び ECMAScript とは異なっている。これらでは、String.substring メソッドが、最初の文字の位置を 0 として扱う。

返す部分文字列は、文字の位置が、2 番目の引数の丸められた値以上になる文字を含み、更に、3 番目の引数を指定した場合には、文字の位置が、2 番目の引数の丸められた値と 3 番目の引数の丸められた値との合計よりも小さな文字（だけ）を含む。これらに使用される比較及び加算は、規定 IEEE 754 の規則に従う。丸めは、round 関数の呼出しによるものとして行われる。次の例は、さまざまな通常ではない場合を示す。

例

```
substring("12345", 1.5, 2.6) は、"234" を返す。
substring("12345", 0, 3) は、"12" を返す。
substring("12345", 0 div 0, 3) は、"" を返す。
substring("12345", 1, 0 div 0) は、"" を返す。
substring("12345", -42, 1 div 0) は、"12345" を返す。
substring("12345", -1 div 0, 1 div 0) は、"" を返す。
```

関数 数値型 string-length(文字列型 ?)

string-length 関数は、文字列（3.6 参照）の中の文字の個数を返す。引数を省略した場合には、文字列に変換した文脈ノード、言い換えると、文脈ノードの文字列値、を既定の引数として用いる。

関数 文字列型 normalize-space(文字列型 ?)

normalize-space 関数は、引数の文字列を、先行する空白及び後続する空白を取り除き空白文字の列を一つのスペースに置き換えることによって正規化された空白をもつ文字列にして返す。空白文字は、JIS X 4159 における S の生成規則によって規定するものと同じとする。引数を省略した場合には、文字列に変換された文脈ノード、言い換えると、文脈ノードの文字列値、を既定の引数として用いる。

関数 文字列型 translate(文字列型, 文字列型, 文字列型)

translate 関数は、2 番目の引数の文字列の中の文字の、最初の引数の文字列の中での出現を、3 番目の引数の文字列の中の対応する位置にある文字で置き換えて返す。例えば、translate("bar","abc","ABC") は、文字列 BAr を返す。（2 番目の引数の文字列が 3 番目の引数の文字列よりも長いために、）3 番目の引数の文字列の中の対応する位置に文字がない、2 番目の引数の文字列の中の文字が存在する場合

には、最初の引数の文字列の中のその文字の出現は取り除かれる。例えば、`translate` ("`--aaa--`", "`abc`", "`ABC`") は、"`AAA`" を返す。一つの文字が 2 番目の引数の文字列の中に一回よりも多く出現する場合には、最初の出現が、その置換する文字を決定する。3 番目の引数の文字列が 2 番目の引数の文字列よりも長い場合には、余分な文字は無視する。

注記 `translate` 関数は、すべての言語における大文字及び小文字の変換のための十分な解ではない。

XPath の将来の版は、大文字及び小文字の変換のための追加関数を提供するかもしれない。

4.3 論理型関数

関数 論理型 `boolean`(オブジェクト型)

`boolean` 関数は、次のとおりに、その引数を論理型に変換する。

- a) 数値は、正の 0、負の 0 又は NaN のいずれでもない場合に限り、真とする。
- b) ノード集合は、空でない場合に限り、真とする。
- c) 文字列は、その長さが 0 でない場合に限り、真とする。
- d) これらの基本型以外の型であるオブジェクトは、その型に依存する方法で論理型に変換する。

関数 論理型 `not`(論理型)

`not` 関数は、その引数が偽の場合に真を返し、そうでない場合に偽を返す。

関数 論理型 `true`()

`true` 関数は、真を返す。

関数 論理型 `false`()

`false` 関数は、偽を返す。

関数 論理型 `lang`(文字列型)

`lang` 関数は、`xml:lang` 属性によって指定した文脈ノードの言語が引数の文字列によって指定した言語と同じ又はその部分言語(sublanguage)になっているかどうか依存して、真又は偽を返す。文脈ノードの言語は、その文脈ノードの `xml:lang` 属性の値によって決定するか、又は文脈ノードが `xml:lang` 属性をもたない場合には、その文脈ノードの `xml:lang` 属性をもつ直近の祖先における `xml:lang` 属性の値によって決定する。それらの属性が存在しない場合には、`lang` 関数は、偽を返す。それらの属性が存在する場合には、その属性値が大文字及び小文字を無視した引数に等しいとき、又は - で開始する添え字が存在し、その属性値の添え字を無視し大文字及び小文字を無視した引数にその属性値が等しいとき、真を返す。

例

`lang("en")` は、文脈ノードが次の五つの要素のいずれかである場合に、真を返す。

```
<para xml:lang="en"/>
<div xml:lang="en"><para/></div>
<para xml:lang="EN"/>
<para xml:lang="en-us"/>
```

4.4 数値型関数

関数 数値型 `number`(オブジェクト型 ?)

`number` 関数は、次のとおりに、その引数を数値に変換する。

- a) 任意選択の空白、任意選択のマイナス記号、`Number` 及び空白がこの順序で続く文字列は、(IEEE 754 の直近への丸め規則に従って、) その文字列が表現する数学的な値に最も近い IEEE 754 の数値に変換する。それ以外の文字列は、NaN に変換する。

- b) 論理型の真は、1 に変換する。論理型の偽は、0 に変換する。
- c) ノード集合は、最初に `string` 関数への呼出しによるものとして文字列に変換し、次に文字列引数と同じ方法で変換する。
- d) これらの基本型以外の型のオブジェクトは、その型に依存する方法で数値に変換する。

引数を省略した場合には、文脈ノードを唯一のメンバとするノード集合を既定の引数として用いる。

注記 `number` 関数は、XML 文書中の要素が言語中立のフォーマットで数値データを表現する型になっていない場合、その要素に出現する数値データを変換するために使用しないほうがよい。言語中立のフォーマットの数値データは、通常は、利用者に表示するために言語特有のフォーマットに変換される。さらに、`number` 関数は、その要素が使用する言語中立のフォーマットが `Number` のための XPath 構文と矛盾する場合には、使用できない。

関数 数値型 `sum`(ノード集合型)

`sum` 関数は、引数のノード集合の中の各ノードに対して、そのノードの文字列値を数値に変換した結果の合計を返す。

関数 数値型 `floor`(数値型)

`floor` 関数は、その引数より大きくない整数である最大の（正の無限大に最も近い）数値を返す。引数が NaN の場合、NaN を返す。引数が正の無限大の場合、正の無限大を返す。引数が負の無限大の場合、負の無限大を返す。引数が正の 0 の場合、正の 0 を返す。引数が負の 0 の場合、負の 0 を返す。引数が 0 よりも大きい 1 よりも小さい場合、正の 0 を返す。

関数 数値型 `ceiling`(数値型)

`ceiling` 関数は、その引数より小さくない整数である最小の（負の無限大に最も近い）数値を返す。引数が NaN の場合、NaN を返す。引数が正の無限大の場合、正の無限大を返す。引数が負の無限大の場合、負の無限大を返す。引数が正の 0 の場合、正の 0 を返す。引数が負の 0 の場合、負の 0 を返す。引数が 0 よりも小さい -1 よりも大きい場合、負の 0 を返す。

関数 数値型 `round`(数値型)

`round` 関数は、その引数に最も近い整数の数値を返す。そのような二つの数値が存在する場合には、正の無限大に最も近いものを返す。引数が NaN の場合には、NaN を返す。引数が正の無限大の場合には、正の無限大を返す。引数が負の無限大の場合には、負の無限大を返す。引数が正の 0 の場合には、正の 0 を返す。引数が負の 0 の場合には、負の 0 を返す。引数が 0 よりも小さいが、-0.5 以上の場合には、負の 0 を返す。

注記 これらの最後の二つの場合については、`round` 関数を呼び出した結果は、0.5 を加えてその次に `floor` 関数を呼び出した結果と同じではない。

5 データモデル

XPath は、木としての XML 文書进行操作する。箇条 5 では、どのようにして XPath が XML 文書を木としてモデル化するかを示す。このモデルは、単に概念的なものであって、何らかの特別な実装を強制するわけではない。このモデルの、XML 情報集合 [XML Infoset] に対する関係は、**附属書 B** に示す。

XPath によって操作される XML 文書は、XML 名前空間の規格 **JIS X 4158** に適合しなければならない。木はノードを含む。ノードには次の七つの型がある。

- a) ルートノード
- b) 要素ノード

- c) テキストノード
- d) 属性ノード
- e) 名前空間ノード
- f) 処理命令ノード
- g) コメントノード

すべてのノード型に対して、その型のノードに対する文字列値を決定する方法がある。あるノード型に対しては、文字列値はノードの一部とし、別のノード型に対しては、文字列値は子孫ノードの文字列値から計算する。

注記 要素ノード及びルートノードの場合は、ノードの文字列値は、DOM の `nodeValue` メソッド ([DOM]参照) が返す文字列と同じではない。

展開名をもつノード型もある。展開名は、局所部分と名前空間 URI とから成る対とする。局所部分は、文字列とする。名前空間 URI は、ヌル又は文字列のいずれかとする。XML 文書の中の名前空間宣言において指定した名前空間名は、[RFC3986] (及び[RFC3987]) において定義する URI (及び／又は IRI) を参照する。このことは、それが素片識別子をもつことができ、相対的になることが可能なことを意味する。展開名の名前空間 URI 構成要素は、その接頭辞が、(素片識別子の有無にかかわらず) 相対 URI である名前空間名をもつ名前空間宣言によって宣言される QName から、展開名が展開される場合、実装依存とする。それら展開名の名前空間 URI 構成要素の値に依存する XPath 式は、相互運用可能ではない。二つの展開名は、同一の局所部分をもつ場合であって、両者が共にヌルの名前空間 URI をもつか、又は両者が共に等しい非ヌルの名前空間 URI をもつかのいずれかの場合に、等しくなる。

文書内のすべてのノードに対して定義され、一般実体展開後の文書の XML 表現において、各ノードの XML 表現の最初の文字が出現する順序に対応する、文書順という順序付けがある。これによって、ルートノードは、最初のノードになる。要素ノードは、その子よりも前に出現する。したがって、文書順は、要素ノードを、(実体展開後の) XML における開始タグの出現順序に順序付ける。要素の属性ノード及び名前空間ノードは、要素の子よりも前に出現する。名前空間ノードは、属性ノードの前に出現する。複数の名前空間ノードの相対的順序は、実装依存とする。複数の属性ノードの相対的順序は、実装依存とする。逆文書順は、文書順の逆とする。

ルートノード及び要素ノードは、順序付けられた子ノードリストをもつ。複数のノードが子を共有することは決してない。あるノードがもう一つのノードと同じでない場合には、あるノードの子は、もう一つのノードの子のどれとも同じにならない。ルートノード以外のすべてのノードは、親をただ一つだけもち、親は要素ノード又はルートノードのいずれかとする。ルートノード又は要素ノードは、その子ノードのそれぞれの親とする。ノードの子孫は、ノードの子及びノードの子の子孫とする。

5.1 ルートノード

ルートノードは、木のルートとする。ルートノードは、木のルートとして以外は出現しない。文書要素の要素ノードは、ルートノードの子とする。ルートノードは、前書きの中及び文書要素末尾の後に出現する、処理命令及びコメントに対する処理命令ノード及びコメントノードも、子としてもつ。

ルートノードの文字列値は、ルートノードのすべてのテキストノード子孫の文字列値の、文書順における連結とする。

ルートノードは、展開名をもたない。

5.2 要素ノード

文書内のあらゆる要素に対して要素ノードが存在する。要素ノードは、XML 名前空間 (JIS X 4158 参照)

に従い、タグの中で指定する要素の QName を展開して計算される展開名をもつ。QName が接頭辞をもたず、適用可能な既定の名前空間が存在しない場合には、要素の展開名の名前空間 URI はヌルとする。

注記 JIS X 4158 の A.3 の記法では、展開名の局所部分は、ExpEType 要素の type 属性に対応する。

展開名の名前空間 URI は、ExpEType 要素の ns 属性に対応し、ExpEType 要素の ns 属性が省略された場合には、ヌルとする。

要素ノードの子は、その内容のための、要素ノード、コメントノード、処理命令ノード及びテキストノードとする。内部実体及び外部実体のどちらへの実体参照も一緒に展開する。文字参照は解決する。

要素ノードの文字列値は、要素ノードのすべてのテキストノード子孫の文字列値の、文書順における連結とする。

5.2.1 一意 ID

要素ノードは、一意の識別子(ID)をもってもよい。これは、DTD において ID 型として宣言する属性の値とする。一つの文書中のどの二つの要素も同じ一意 ID をもってはならない。XML プロセサが、(文書が妥当でない場合だけに可能だが、) 一つの文書中の二つの要素が同じ一意 ID をもっていると報告した場合、文書順で二番目の要素は一意 ID をもたないものとして扱わなければならない。

注記 文書が DTD をもたない場合、文書中のどの要素も一意 ID をもたないことになる。

5.3 属性ノード

各要素ノードは、関連付けられた属性ノードの集合をもち、要素は、これらの属性ノードそれぞれの親とする。ただし、属性ノードは、親である要素の子とはしない。

注記 これは、DOM とは異なる。DOM は、属性をもつ要素を属性の親としては扱わない ([DOM]参照)。

複数の要素が属性ノードを共有することは決してない。ある要素ノードがもう一つの要素ノードと同じノードでない場合、ある要素ノードの属性ノードはどれも、もう一つの要素ノードの属性ノードと同じノードにならない。

注記 = 演算子は、二つのノードが同じ値をもつかどうかを試験し、それらが同じノードであるかどうかは試験しない。したがって、二つの異なる要素の属性を、たとえそれらが同じノードでなくとも、= を使用して等しいかどうか比較してよい。

既定値の属性も指定した属性と同一に扱う。ある属性を DTD 内でその要素型に対して宣言したが、既定値を #IMPLIED と宣言し、しかもその属性をその要素に関して指定しない場合には、その要素の属性集合は、その属性に対するノードを含まない。

xml:lang, xml:space などの属性は、別の子孫要素上の同じ属性のインスタンスで上書きしない場合には、その属性が、その属性をもつ要素の子孫であるすべての要素に適用される。しかし、これは、属性ノードが木の中のどこに現れるかに影響しない。要素は、その要素の開始タグ若しくは空要素タグにおいて明示的に指定した属性だけに対して、又は DTD において既定値付きで明示的に宣言した属性だけに対して、属性ノードをもつ。

属性ノードは、展開名及び文字列値をもつ。展開名は、XML 名前空間の規格 JIS X 4158 に従い、XML 文書中のタグで指定される QName を展開して計算される。属性名の名前空間 URI は、その属性の QName が接頭辞をもたない場合には、ヌルになる。

注記 JIS X 4158 の A.3 の記法では、展開名の局所部分は ExpAName 要素の name 属性に対応する。

展開名の名前空間 URI は、ExpAName 要素の ns 属性に対応し、ExpAName 要素の ns 属性が省略された場合には、ヌルとする。

属性ノードは、文字列値をもつ。文字列値は、XML の規格(JIS X 4159)によって指定する、正規化した値とする。正規化した値が長さ 0 の文字列である属性も特別扱いされず、文字列値が長さ 0 の文字列である属性ノードとなるだけとする。

注記 既定値属性を、外部 DTD 又は外部パラメタ実体の中で宣言していることもある。XML の規格は、XML プロセサに対して、それが妥当性検証していない場合には、外部 DTD 又は外部パラメタを読むことを要求しない。外部 DTD 又は外部パラメタ実体で宣言された既定値属性の値を XPath 木が含むことを前提とするスタイルシートなどは、妥当性検証を行わない XML プロセサとともに動作しなくてもよい。

名前空間を宣言する属性 (JIS X 4158 参照) に対応する属性ノードは存在しない。

5.4 名前空間ノード

各要素は、関連付けられた名前空間ノードの集合をもつ。要素の有効範囲内にある明示した名前空間接頭辞 (これは、XML 名前空間の規格 JIS X 4158 によって暗黙に宣言する、xml 接頭辞を含む。) それぞれに対して一つ、及び既定の名前空間が要素の有効範囲内に存在する場合には、それに対して一つとする。要素は、これらの名前空間ノードそれぞれの親とするが、名前空間ノードは、親である要素の子とはしない。複数の要素は、名前空間ノードを共有しない。ある要素ノードがもう一つの要素ノードと同じでない場合、ある要素ノードの名前空間ノードは、どれももう一つの要素ノードの名前空間ノードと同じにはならない。このことは、要素が次の属性に対して名前空間ノードをもつということを意味している。

- a) 名前が xmlns: で始まり、その要素自体又はより近い祖先要素がその接頭辞を空値で再宣言していない場合には、その値が空でない、要素及び直近の祖先要素上のあらゆる属性。

注記 接頭辞を宣言しないことは、[XML Names 1.1] に適合する文書だけで起こりえる。

- b) その要素又は祖先が xmlns 属性をもち、直近のそのような要素の xmlns 属性の値が空でない場合には、xmlns 属性。

注記 xmlns="" 属性は、既定の名前空間の“取消しを宣言する” (JIS X 4158 参照)。

名前空間ノードは、展開名をもつ。局所部分は、名前空間接頭辞とし (これは、名前空間ノードが既定の名前空間に対するものの場合、空になる。)、名前空間 URI は、常にヌルとする。

名前空間ノードの文字列値は、その名前空間接頭辞に束縛されている名前空間 URI とする。XML 文書の中の名前空間宣言に出現している名前空間名が (素片識別子の有無にかかわらず) 相対 URI となる場合、文字列値は、実装依存とする。それら名前空間ノードの文字列値に依存する XPath 式は、相互運用可能ではない。

5.5 処理命令ノード

文書型宣言内部に出現する処理命令を除いて、あらゆる処理命令に対応して処理命令ノードが存在する。

処理命令は、展開名をもつ。局所部分は、処理命令のターゲットであって、名前空間 URI はヌルとする。処理命令ノードの文字列値は、ターゲット及び空白に続く処理命令の部分とする。終了の ?> は含まない。

注記 XML 宣言は、処理命令とはしない。したがって、XML 宣言に対応する処理命令ノードは存在しない。

5.6 コメントノード

文書型宣言内部に出現するコメントを除いて、あらゆるコメントに対してコメントノードが存在する。

コメントの文字列値は、開きの <!-- 又は閉じの --> を含まない、コメントの内容とする。

コメントノードは、展開名をもたない。

5.7 テキストノード

文字データは、テキストノードにまとめる。できるだけ多くの文字データを各テキストノード内にまとめる。テキストノードは、直前又は直後にテキストノードの兄弟をもつことはない。テキストノードの文字列値は、文字データとする。テキストノードは、必ず少なくとも 1 文字のデータをもつ。

CDATA セクション内にある各文字は、文字データとして扱う。そのため、ソース文書内の `<![CDATA[<]]>` は `<` と同じに扱う。両者共に、木のテキストノードでは 1 個の `<` 文字となる。したがって、CDATA セクションは、`<![CDATA[` 及び `]]>` が除去され、`<` 及び `&` の出現がそれぞれ `<` 及び `&` で置き換えられたものとして扱われる。文書要素の外の空白は、テキストノードを生成しない。

注記 `<` 文字を含むテキストノードが XML として書き出されるとき、`<` 文字は、例えば、`<` を使う、又は CDATA セクションの中を含めるとして、別扱いしなければならない。

コメント、処理命令及び属性値の内部の文字は、テキストノードを生成しない。外部実体内の行末は、XML の規格(JIS X 4159)で指定されるとおりに `#xA` に正規化する。

テキストノードは、展開名をもたない。

6 適合性

XPath は、主に、他の規定で利用できる構成要素として意図されている。したがって、XPath の実装の適合性のための判定基準を示すには、([XPath]及び JIS X 4169 といった) XPath を使用する規定に左右されることになるので、XPath の独立した実装のための適合性判定基準を定義することはしない。

附属書 A

(規定)

文献

序文

この附属書の **A.1** は、引用規格について規定する。また、**A.2** は、参考文献について記載するものである。規定の一部ではない。

A.1 引用規格 (規定)

IEEE 754

Institute of Electrical and Electronics Engineers. IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754-1985.

RFC3986

T. Berners-Lee, R. Fielding, L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. IETF RFC 3986.

注記 <http://www.ietf.org/rfc/rfc3986.txt> を参照。

RFC3987

M. Duerst, M. Suignard. Internationalized Resource Identifiers (IRIs). IETF RFC 3987.

注記 <http://www.ietf.org/rfc/rfc3987.txt> を参照。

XML

JIS X 4159:2005 拡張可能なマーク付け言語 (XML) 1.0

注記 World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation. が、この規格に一致している。

<http://www.w3.org/TR/2004/REC-xml-20040204/>を参照。

XML 1.1

World Wide Web Consortium. Extensible Markup Language (XML) 1.1. W3C Recommendation.

注記 <http://www.w3.org/TR/xml11/> を参照。

XML Names

JIS X 4158:2005 XML 名前空間

注記 World Wide Web Consortium. Namespaces in XML. W3C Recommendation. が、この規格に一致している。

<http://www.w3.org/TR/REC-xml-names> を参照。

XML Names 1.1

World Wide Web Consortium. Namespaces in XML 1.1. W3C Recommendation.

注記 <http://www.w3.org/TR/xml-names11/> を参照。

A.2 参考文献 (参考)

Character Model

World Wide Web Consortium. Character Model for the World Wide Web. W3C Working Draft.

注記 <http://www.w3.org/TR/WD-charmod> を参照。

DOM

World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification. W3C Recommendation.

注記 <http://www.w3.org/TR/REC-DOM-Level-1> を参照。

JLS

J. Gosling, B. Joy, and G. Steele. The Java Language Specification.

注記 <http://java.sun.com/docs/books/jls/index.html> を参照。

ISO/IEC 10646

JIS X 0221:1995 国際符号化文字集合(UCS)－第 1 部 体系及び基本多言語面

注記 1 **ISO/IEC 10646-1:1993**, Information technology－Universal Multiple-Octet Coded Character Set (UCS)－Part 1: Architecture and Basic Multilingual Plane. International Standard. が、この規格に一致している。

注記 2 **JIS X 0221:1995** は、2000 年に **ISO/IEC 10646-1** の第 2 版が発行され、この規格に一致した **JIS X 0221-1:2001** に置き換えられている。2003 年に **ISO/IEC 10646, Information technology－Universal Multiple-Octet Coded Character Set (UCS)**が発行され、この規格、**Amendment 1 (2005)**及び **Amendment 2 (2006)**に対応した **JIS** の作成が進められている。

TEI

C.M. Sperberg-McQueen, L. Burnard Guidelines for Electronic Text Encoding and Interchange.

注記 <http://etext.virginia.edu/TEI.html> を参照。

Unicode

Unicode Consortium. The Unicode Standard.

注記 <http://www.unicode.org/unicode/standard/standard.html> を参照。

XML Infoset

World Wide Web Consortium. XML Information Set. W3C Recommendation.

注記 <http://www.w3.org/TR/xml-infoset> を参照。

XPointer

World Wide Web Consortium. XML Pointer Language (XPointer). W3C Working Draft.

注記 <http://www.w3.org/TR/WD-xptr> を参照。

XQL

J. Robie, J. Lapp, D. Schach. XML Query Language (XQL).

注記 <http://www.w3.org/TandS/QL/QL98/pp/xql.html> を参照。

XSLT

JIS X 4169:2007, XSL 変換 (XSLT) 1.0

注記 World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation. が、この規格に一致している。

附属書 B

(参考)

XML 情報集合との対応付け

序文

この附属書は、規定の一部ではない。

XPath データモデルにおけるノードは、XML 情報集合[XML Infoset]によって提供される次の情報項目から導き出すことができる。

注記 附属書 B における次の記述で、“[~~×~~特性”は、XML 情報集合[XML Infoset]において定義された特性を参照する記法であって、文献参照ではないことに注意する。

B.1 ルートノード

XPath データモデルのインスタンスは、ただ一つのノードを含み、それは、XML 情報集合における一意の文書情報項目に対応する。

- a) ルートノードの子は、[children]特性の中に見出される情報項目であって、あらゆる文書型宣言情報項目を省略したものに対応するノードとする。

B.2 要素ノード

要素ノードは、要素情報項目に対応する。

- a) 要素ノードの子は、[children]特性の中に出現する情報項目に対応するノードとする。この対応は、連続する文字情報項目という子が一つのテキストノードに合体されるので、1 対 1 対応ではない。XPath データモデルは、すべての一般実体が展開されることを要求するので、非展開実体参照情報項目という子は、決して存在しない。
- b) 要素ノードの属性は、[attributes]特性の中に出現する属性情報項目に対応するノードとする。
- c) 要素ノードの名前空間は、[in-scope namespaces]特性の中に出現する名前空間情報項目に対応するノードとする。
- d) 要素ノードの拡張名の局所部分は、[local name]特性に対応する。要素ノードの拡張名の名前空間 URI は、[namespace name]特性に対応する。
- e) 要素ノードの一意識別子 (ID) は、対応する属性が存在する場合には、ID の[attribute type]特性をもつ [attributes]特性の中の属性情報項目の[normalized value]特性に対応する。そうでない場合には、ヌルとする。
- f) 要素ノードの親は、[parent]特性に対応する。

B.3 属性ノード

属性ノードは、属性情報項目に対応する。名前空間宣言は、属性としてはモデル化しない。

- a) 属性ノードの拡張名の局所部分は、[local name]特性に対応する。属性ノードの拡張名の名前空間 URI は、[namespace name]特性に対応する。
- b) 属性ノードの文字列値は、[normalized value]特性に対応する。
- c) 属性ノードの親は、[owner element]特性に対応する。

B.4 テキストノード

テキストノードは、一つ以上の連続する文字情報項目の列に対応する。

- a) テキストノードの文字列値は、文字情報項目の各々の連結された[character code]特性に対応する。
- b) テキストノードの親は、連続する文字情報項目の任意の一つの[parent]特性に対応する。連続する文字は、常に、同じ親をもつ。

B.5 処理命令ノード

処理命令ノードは、処理命令情報項目に対応する。文書型宣言情報項目の子となる処理命令のための処理命令ノードは存在しない。

- a) 処理命令ノードの拡張名の局所部分は、[target]特性に対応する。処理命令ノードの拡張名の名前空間URIは、ヌルとする。
- b) 処理命令ノードの文字列値は、[content]特性に対応する。
- c) 処理命令ノードの親は、[parent]特性に対応する。

B.6 コメントノード

コメントノードは、コメント情報項目に対応する。

- a) コメントノードの文字列値は、[content]特性に対応する。
- b) コメントノードの親は、コメント情報項目の[parent]特性に対応する。

B.7 名前空間ノード

名前空間ノードは、名前空間情報項目に対応する。

- a) 名前空間ノードの拡張名の局所部分は、[prefix]特性に対応する。名前空間ノードの拡張名の名前空間URIは、ヌルとする。
- b) 名前空間ノードの文字列値は、[namespace name]特性に対応する。
- c) 名前空間ノードの親は、このノードが出現する名前空間の集まりの中の要素ノードとする。

B.8 XML 情報集合適合性

この規定は、XML 情報集合[XML Infoset]に適合する。次の情報項目は、データモデルのインスタンスを構成するために情報集合生成器によって開示しなければならない。

- a) [children]特性をもつ文書情報項目。
- b) [children]特性、[attributes]特性、[in-scope namespaces]特性、[local name]特性、[namespace name]特性、及び[parent]特性をもつ要素情報項目。
- c) [namespace name]特性、[local name]特性、[normalized value]特性、[owner element]特性及び[attribute type]特性をもつ属性情報項目。
- d) [character code]特性及び[parent]特性をもつ文字情報項目。
- e) [target]特性、[content]特性及び[parent]特性をもつ処理命令情報項目。
- f) [content]特性及び[parent]特性をもつコメント情報項目。
- g) [prefix]特性及び[namespace name]特性をもつ名前空間情報項目。

情報集合生成器によって利用可能なその他の情報項目及び特性は、無視する。