8

XML

Fundamentos de XPath

Helder da Rocha
(helder@argonavis.com.br)

argonavis

O que é XPath

- XPath é uma linguagem usada para **localizar** informações em um documento XML
  - Serve para **navegar pelos nós** e localizar dados
  - É usada por várias outras tecnologias do XML, como XSLT, Xquery, Xpointer, XML Schema, bancos de dados e linguagens que fazem mapeamento com XML
- XPath opera sobre o **XML processado**
  - O arquivo-fonte usado pelo XPath não tem entidades (por exemplo: ã) ou blocos CDATA
  - O processador resolve todas as entidades antes do processamento com XPath, e todas as entidades e seções CDATA são convertidas em XML e texto

Exemplo de XPath

- *Uma expressão XPath é um caminho na árvore-fonte que resulta em um*
  - *valor (número, texto, booleano),*
  - *objeto (elemento, atributo, nó de texto) ou*
  - *conjunto de objetos*

- *Caminhos absolutos*

```
/contato/@codigo
/contato/telefone/@tipo
/contato/telefone/numero/text()
```

- *Relativos ao contexto* `/contato` :

```
@codigo (ou ./@codigo)
telefone/@tipo (ou ./telefone/@tipo)
telefone/numero/text()
```

- *Expressões XPath são usadas dentro de atributos XML*
  - *Usadas em XSLT, XLink, XQuery e XPointer*

Modelo de dados

- *XPath trata um documento XML como uma **árvore de nós** (similar, mas não igual, à àrvore DOM)*
  - *DOM opera sobre o documento XML cru, e pode representar entidades e blocos CDATA*
- *Os nós usados pelo XPath podem ser de sete tipos*
  - *Raiz (só há um desses)*
  - *Elemento*
  - *Atributo*
  - *Texto*
  - *Namespace*
  - *Instrução de processamento*
  - *Comentário*

Tipos e valores

- *Cada nó, ao ser processado, produz um **valor** que tem um **tipo***
- *O conteúdo de um nó processado **pode** ser representado pelo **conteúdo de texto** do nó*
  - *Todos os nós de texto que não estiverem em atributos*
- *O **valor** do nó pode conter um dos quatro* tipos de conteúdo a seguir*
  - *uma estrutura de outros nós (**node-set**)*
  - *um escalar numérico (**number**)*
  - *um string (**string**)*
  - *um valor booleano (**boolean**)*

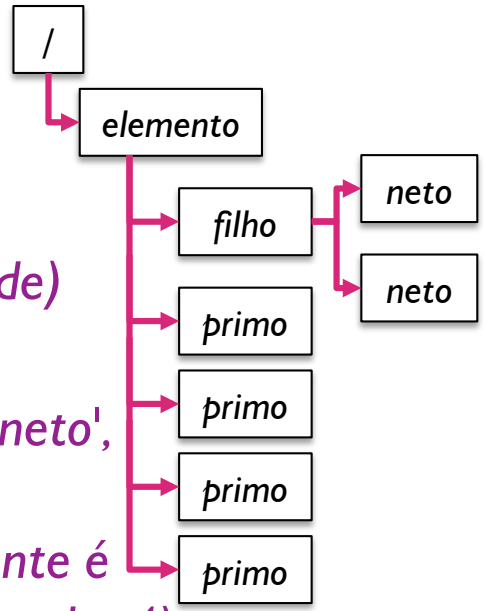
** A abordagem de XPath neste curso está restrita a XPath 1.0; XPath 2.0 suportará muito mais tipos (todos os tipos do XML Schema).*

Expressões XPath 1.0

- Os **tipos de expressão** suportadas por XPath são relacionados aos tipos de dados
  - operações sobre nós da árvore-fonte (caminhos)
  - operações sobre texto
  - operações booleanas
  - operações numéricas
- E cada tipo de expressão devolve um resultado que pode ser um dos quatro **tipos de dados**
  - um conjunto de nós (node-set)
  - um texto (string)
  - um valor booleano (boolean)
  - um número (number)

Caminhos (location paths)

- Um **caminho** é uma seqüência de passos de navegação na árvore-fonte (documento-fonte)
  - Todo caminho resulta em um nó (**node**) ou conjunto de nós (**node-set**)
- O resultado de um caminho produz um contexto
  - O **nó de contexto**: expressões seguintes relativas ao contexto
  - Se for um node-set, o processamento de **cada nó do conjunto** também introduz um contexto: o **nó corrente**
  - Todo contexto tem um **tamanho** e uma **posição**
- Caminhos podem ser absolutos ou relativos
  - absolutos: começam no **nó raiz** (iniciam com "/")
  - relativos: começam no **nó do contexto** (context node)
- Exemplos:
  - `/raiz/elemento/filho/neto` : contexto é node-set 'neto', que tem tamanho 2
  - `../../primo[3]` : caminho relativo a 'neto', nó corrente é elemento <primo> na posição 3 (node-set tem tamanho 4)



Passos

- *Um caminho contém uma* ***seqüência de passos***
- *Cada passo pode ter três partes*
  - *um* ***eixo****: descreve a direção a ser tomada, e se representa um namespace, atributo ou elemento*
    - *ancestor, sibling, descendant, child, etc.*
    - *attribute, namespace*
  - *um* ***teste****: que seleciona um conjunto de nós*
    - *nome do nó, tipo do nó*
  - *um* ***predicado*** *opcional: que reduz o conjunto com base em características dos nós*
    - *atributos, valores de atributos, posição*
- *Sintaxe*
  - `eixo::teste[predicado]`

Eixos

- Há treze eixos
  - // para navegar entre elementos
  - / para navegar por atributos
  - / para navegar por namespace
- Eixos que representam elementos
  - ancestor, ancestor-or-self
  - child, self, parent
  - descendant, descendant-or-self
  - following, preceding
  - following-sibling, preceding-sibling
- Eixo que representa um atributo
  - attribute
- Eixo que representa um namespace
  - namespace

Atalhos

- *Pode-se usar **símbolos** em vez dos **nomes** de alguns eixos mais comuns*
  - `descendant-or-self::` `//`
  - `self::` `.`
  - `parent::` `..`
  - `attribute::` `@`
  - `child::` *(ausência de eixo)*
- *Ex:*
  - **`child::filho`** *é a mesma coisa que* **`filho`** *ou* **`./filho`** *(*`self::node()/child::filho`*)*
  - **`descendant-or-self::neto`** *é o mesmo que* **`//neto`**
  - **`attribute::id`** *é o mesmo que* **`@id`**
  - **`parent::node()/filho`** *é o mesmo que* **`../filho`**

Testes (eixo::teste)

- *Um teste **restringe** os resultados de um eixo*
- **`eixo::node()`**
  - *qualquer **nó** (inclusive comentários, nós de texto e instruções de processamento)*
- **`eixo::*`**
  - *qualquer elemento, atributo ou namespace*
  - *Ex:* **`attribute::*`** *(qualquer atributo)*
- **`eixo::`***`nome`*
  - *Onde **nome** é o nome de um elemento, atributo ou namespace*
  - *Ex:* **`child::table`** *(o elemento filho table)*
  - *Ex:* **`self::table`** *(o elemento corrente é table)*
  - *Ex:* **`parent::table`** *(o elemento pai é table)*

Testes (2)

- **eixo::text()**
  - *qualquer nó de texto*
- **eixo::comment()**
  - *qualquer nó de comentário*
- **eixo::processing-instruction()**
  - *qualquer instrução de processamento*
- **eixo::processing-instruction('alvo')**
  - *instrução* **<?alvo ... ?>**
- *Nem todos os eixos podem ser usados com estes testes (ex: parent ou attribute não podem)*

Exemplos de passos simples

- `child::node()` *ou* `node()`
- `child::*` *ou* `*`
- `parent::node()` *ou* `..`
- `parent::elemento`
- `self::node()` *ou* `.`
- `child::comment()` *ou* `comment()`
- `preceding-sibling::*`
- `following-sibling::node()`
- `following::elemento`
- `ancestor-or-self::elemento`

inclui elementos, comentários, etc.

apenas elementos

Predicados

- *Expressão XPath entre colchetes, com resultado* ***booleano***
- *Usada para filtrar resultado de* ***um passo***
  - *Opera* ***no contexto do node-set*** *do passo*
- `child::astro[child::orbita]` *ou* `astro[orbita]`
  - *Predicado será true() se existir astro/orbita*
- `produto[@preco="1.99"]`
  - *true() se produto tiver atributo preco contendo 1.99*
- `produto[attribute::nome='livro']/attribute::preco`
  - *predicado restringe node-set* ***produto*** *(primeiro passo)*
  - *expressão (caminho de dois passos) retorna atributo* ***preco***
- `child::*[position() != last()]`
- `preceding::node()[1]`
- `parent::livro[@idioma="pt" | @idioma="en"]`
- `parent::node()[../usados]/child::livro[@ebook][@id='869']`
  - *predicado duplo (restrição do tipo 'and') no segundo passo*
  - *pai de <livro> precisa estar dentro de um bloco <usados>*

Exemplos de caminhos equivalentes

(1)

- `parent::*/following-sibling::paragrafo`
- `../following-sibling::paragrafo`

(2)

- ```
  descendant-or-self::capitulo[position()=1]/
                      child::secao[position()=4]/
                  child::paragrafo[position()=1]
  ```
- `//capitulo[1]/secao[4]/paragrafo[1]`

(3)

- `self::elemento | descendant::elemento`
- `//elemento`
- `descendant-or-self::elemento`

(4)

- ```
  /child::cursos/
   child::curso/
   child::topicos/
   child::item[position()=5]
  ```
- `/cursos/curso/topicos/item[5]`

Expressões booleanas (I)

- *Operadores de comparação*
  - `a = b` — *igualdade*
  - `a != b` — *diferença*
  - `a < b` — *a menor que b**
  - `a > b` *ou* `a > b` — *a maior que b**
  - `a <= b` — *a menor ou igual a b**
  - `a >= b` *ou* `a >= b` — *a maior ou igual a b**

* Se ***não*** *forem usadas dentro de um documento XML, as expressões podem ser escrita sem os escapes: a < b, a > b, a <= b, a >= b*

Expressões booleanas (2)

- *Operadores booleanos*
  - `expr1 and expr2` — *E lógico*
  - `expr1  or expr2` — *OU lógico*
  - `not (expressao)` — *Negação*
  - `true()` — *verdadeiro*
  - `false()` — *falso*

Funções de node-set

- **count**(node-set)
  - *conta o número de elementos de um conjunto*
  - *exemplos:*
    - `count(parent::*)` *retorna 1 (um pai)*
    - `count(child::*)` *retorna no. de filhos*
- **last()**
  - *Retorna o número com a posição do último elemento do conjunto de nós correntes*
- **position()**
  - *o número com a posição do nó corrente dentro do conjunto de nós correntes*

Funções do nó de contexto

- `id('id')`
  - *retorna um identificador unívoco para o nó de contexto.*
- `local-name()`
  - *o nome local (sem o prefixo) do nó de contexto*
- `name()`
  - *o nome qualificado (com prefixo de namespace)*
- `namespace-uri()`
  - *URI do namespace do nó de contexto*

Expressões numéricas

`a + b`	*soma*
`a - b`	*subtração*
`a * b`	*multiplicação*
`a` **`div`** `b`	*divisão*
`a` **`mod`** `b`	*resto*
`round``(expressão)`	*arredondamento*
`floor``(expressão)`	*piso (arr. p/ baixo)*
`ceiling``(expressão)`	*teto (arr. p/ cima)*
`sum``(a, b, ..., n)`	*somatório*

Expressões de string

- **concat**(str1, str2, ..., strn)
  - *concatena vários strings*
- **substring**(str, inicio, fim)
  - *retorna um fragmento do string procurado*
- **substring-after**(str, str_buscado)
  - *começa no fim de **str** e termina depois de **str_buscado***
- **substring-before**(str, str_buscado)
  - *começa no início de **str** e termina antes de **str_buscado***

Expressões de string (2)

- **normalize-space**(str)
  - *remove espaços em branco desnecessários e remove espaços antes e depois (trim)*
- **translate**(str, str_buscado, str_substit)
  - *troca todas as ocorrencias de **str_buscado** com **str_substit** em **str***
- **format-number**(num, mascara)
- **format-number**(num, mascara, locale)
  - *retorna um string contendo **num**, formatado de acordo com a **máscara** (e opcionalmente de acordo com o **locale** especificado)*

Expressões de teste de string

- *Retornam valor booleano*
  - `starts-with ( str, substr )`
  - `contains ( str, substr )`
- *Retornam inteiro*
  - `string-length ( str )`