XPath

Processamento Estruturado de Documentos 2002

By jcr

Motivação

- Todos os processos de transformação/formatação de documentos XML começam por construir uma árvore: a **árvore documental abstracta**
- O XPath permite-nos navegar nessa árvore e manipular os seus elementos

Introdução

- O XPath foi desenvolvido para ser utilizado como valor dum atributo num documento XML.
- A sua sintaxe é uma mistura da linguagem de expressões com a linguagem para a especificação do caminho numa estrutura de directorias como a usada nos sistemas Unix ou Windows
- Adicionalmente, o XPath fornece ainda um conjunto de funções para manipulação de texto, Namespaces, e outras ...

O Modelo de Dados do XPath

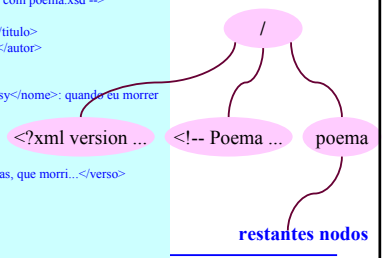
- Do ponto de vista do XPath, um documento XML é uma ADA, uma árvore de nodos.
- Para o XPath há sete tipos de nodos:
  1. **o nodo raiz (um por documento)**
  2. **nodos elemento**
  3. **nodos atributo**
  4. **nodos texto**
  5. **nodos comentário**
  6. **nodos instrução de processamento**
  7. **nodos Namespace**

Exemplo: instância do poema

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Poema anotado de acordo com poema.xsd -->
<poema tipo="soneto">
 <titulo>"Soneto Já Antigo"</titulo>
 <autor>(Álvaro de Campos)</autor>
 <corpo>
  <quadra>
   <verso>Olha, <nome>Daisy</nome>: quando eu morrer
        tu hás-de</verso>
   ...
  </quadra>
  <terno>
   <verso>embora não o saibas, que morri...</verso>
   ...
  </terno>
</corpo>
 <data>(1922)</data>
</poema>
```


Exemplo: nodo raiz

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Poema anotado de acordo com poema.xsd -->
<poema tipo="soneto">
 <titulo>"Soneto Já Antigo"</titulo>
 <autor>(Álvaro de Campos)</autor>
 <corpo>
  <quadra>
   <verso>Olha, <nome>Daisy</nome>: quando eu morrer
        tu hás-de</verso>
   ...
  </quadra>
  <terno>
   <verso>embora não o saibas, que morri...</verso>
   ...
  </terno>
</corpo>
 <data>(1922)</data>
</poema>
```



Exemplo: nodos elemento

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Poema anotado de acordo com poema.xsd -->
<poema tipo="soneto">
 <titulo>"Soneto Já Antigo"</titulo>
 <autor>(Álvaro de Campos)</autor>
 <corpo>
  <quadra>
   <verso>Olha, <nome>Daisy</nome>: quando eu morrer
        tu hás-de</verso>
   ...
  </quadra>
  <terno>
   <verso>embora não o saibas, que morri...</verso>
   ...
  </terno>
</corpo>
 <data>(1922)</data>
</poema>
```

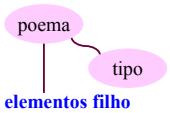
poema
titulo
autor
corpo
quadra
verso
terno
nome
lugar
data

verso = "Olha, Daisy: quando eu morrer ..."

Exemplo: nodos atributo

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Poema anotado de acordo com poema.xsd -->
<poema tipo="soneto">
 <titulo>"Soneto Já Antigo"</titulo>
 <autor>(Álvaro de Campos)</autor>
 <corpo>
  <quadra>
   <verso>Olha, <nome>Daisy</nome>: quando eu morrer
        tu hás-de</verso>
   ...
  </quadra>
  <terno>
   <verso>embora não o saibas, que morri...</verso>
   ...
  </terno>
</corpo>
 <data>(1922)</data>
</poema>
```

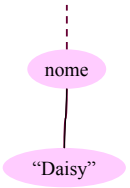
tipo



elementos filho

Exemplo: nodos texto

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Poema anotado de acordo com poema.xsd -->
<poema tipo="soneto">
 <titulo>"Soneto Já Antigo"</titulo>
 <autor>(Álvaro de Campos)</autor>
 <corpo>
  <quadra>
   <verso>Olha, <nome>Daisy</nome>: quando eu morrer
        tu hás-de</verso>
   ...
  </quadra>
  <terno>
   <verso>embora não o saibas, que morri...</verso>
   ...
  </terno>
</corpo>
 <data>(1922)</data>
</poema>
```



Endereçamento

- A sintaxe básica do XPath é muito semelhante à do endereçamento de ficheiros num sistema operativo. Se o endereço começar por / , então estaremos perante um endereço absoluto.

Endereçamento (exemplo1)

/AAA Selecciona o elemento raíz AAA.

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC/>
</AAA>
```


Endereçamento (exemplo2)

/AAA/CCC Selecciona os elementos CCC que são filhos do elemento raíz AAA.

```
<AAA>
    <BBB/>
    <CCC>
        <BBB/>
        <BBB/>
        <DDD>
            <BBB/>
        </DDD>
    </CCC>
</AAA>
```

Endereçamento (exemplo3)

/AAA/DDD/BBB

Selecciona os elementos BBB que são filhos de elementos DDD que, por sua vez são filhos do elemento raíz AAA.

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC/>
</AAA>
```


Descendência

Se o endereço começar por //, então todos os elementos no documento que respeitarem a selecção que vem a seguir serão seleccionados.

Descendência (exemplo1)

//BBB Selecciona todos os elementos BBB.

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC/>
</AAA>
```


Descendência (exemplo2)

//DDD/BBB Selecciona todos os elementos BBB filhos de elementos DDD.

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC>
        <DDD>
            <BBB/>
            <BBB/>
        </DDD>
    </CCC>
</AAA>
```


*

O operador * selecciona todos os elementos abrangidos pelo endereço precedente.

* (exemplo1)

/AAA/CCC/DDD/* Selecciona todos os elementos com contexto: /AAA/CCC/DDD.

```
<AAA>
    <XXX>
        <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
        </DDD>
    </XXX>
```

```
<CCC>
    <DDD>
        <BBB/>
        <BBB/>
        <EEE/>
        <FFF/>
    </DDD>
</CCC>
```

```
<CCC>
    <BBB>
        <BBB>
            <BBB/>
        </BBB>
    </BBB>
</CCC>
</AAA>
```

* (exemplo2)

/*/*/BBB Selecciona todos os elementos BBB com 3 gerações ancestrais.

* (exemplo3)

//* Selecciona todos os elementos.

Predicados

- Em XPath uma expressão dentro de [] designa-se por predicado.
- Um predicado visa especificar ainda mais um dado elemento: testanto a sua posição na árvore, o seu conteúdo, ...

- Se a expressão fôr constituída por apenas um número selecciona o elemento pela posição no seu nível.
- O predicado last() testa se o elemento é o último do seu nível.

Predicados (exemplo1)

/AAA/BBB[1] Selecciona o primeiro elemento BBB filho de AAA.

```
<AAA>
    <BBB/>
    <BBB/>
    <BBB/>
    <BBB/>
</AAA>
```


Predicados (exemplo2)

/AAA/BBB[last()] Selecciona o último elemento BBB filho de AAA.

```
<AAA>
    <BBB/>
    <BBB/>
    <BBB/>
    <BBB/>
</AAA>
```


Atributos

Os atributos são especificados pelo prefixo @.

Atributos (exemplo1)

`//BBB[@ident]` Selecciona os elementos BBB que têm o atributo ident especificado.

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB name="bbb"/>
    <BBB/>
</AAA>
```


Atributos (exemplo2)

`//BBB[@nome]` Selecciona os elementos BBB que têm o atributo nome especificado.

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB nome="bbb"/>
    <BBB/>
</AAA>
```


Atributos (exemplo3)

`//BBB[@*]` Selecciona os elementos BBB que têm um atributo especificado.

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB nome="bbb"/>
    <BBB/>
</AAA>
```


Atributos (exemplo4)

`//BBB[not(@*)]` Selecciona os elementos BBB que não têm nenhum atributo.

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB nome="bbb"/>
    <BBB/>
</AAA>
```


Valores de Atributos

- O valor dum atributo pode ser usado como critério de selecção.
- A função normalize-space retira os caracteres brancos inciais e finais duma string e substitui as cadeias brancas por um espaço.

Valores de Atributos (exemplo1)

`//BBB[@ident=“b1”]` Selecciona os elementos BBB que têm o atributo ident com valor igual a b1.

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB name="bbb"/>
    <BBB/>
</AAA>
```

Valores de Atributos (exemplo2)

`//BBB[@nome="bbb"]` Selecciona os elementos BBB que têm o atributo nome com valor igual a bbb.

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB nome="bbb"/>
    <BBB nome=" bbb  "/>
    <BBB/>
</AAA>
```


Valores de Atributos (exemplo3)

`//BBB[normalize-space(@nome)="bbb"]`

Selecciona os elementos BBB que têm o atributo nome com valor igual a bbb (filtrando espaços iniciais e finais).

```
<AAA>
    <BBB ident="b1"/>
    <BBB ident="b2"/>
    <BBB nome="bbb"/>
    <BBB nome=" bbb  "/>
    <BBB/>
</AAA>
```


Funções: count

A função count dá como resultado o número de elementos resultantes da aplicação do selector que lhe fôr passado como argumento.

count (exemplo1)

`//*[count(BBB)=2]` Selecciona todos os elementos que tenham dois filhos BBB.

```
<AAA>
    <CCC>
        <BBB/>
        <BBB/>
        <BBB/>
    </CCC>
    <DDD>
        <BBB/>
        <BBB/>
    </DDD>
    <EEE>
        <CCC/>
        <DDD/>
    </EEE>
</AAA>
```


count (exemplo2)

`//*[count(*)=2]` Selecciona todos os elementos que tenham dois filhos.

```
<AAA>
    <CCC>
        <BBB/>
        <BBB/>
        <BBB/>
    </CCC>
    <DDD>
        <BBB/>
        <BBB/>
    </DDD>
    <EEE>
        <CCC/>
        <DDD/>
    </EEE>
</AAA>
```


count (exemplo3)

`//*[count(*)=3]` Selecciona todos os elementos que tenham três filhos.

```
<AAA>
    <CCC>
        <BBB/>
        <BBB/>
        <BBB/>
    </CCC>
    <DDD>
        <BBB/>
        <BBB/>
    </DDD>
    <EEE>
        <CCC/>
        <DDD/>
    </EEE>
</AAA>
```

Funções: name

- A função name retorna o nome do elemento seleccionado.
- A função starts-with recebe dois argumentos do tipo string e retorna verdadeiro se o primeiro argumento inicia com o segundo.
- A função contains recebe dois argumentos do tipo string e retorna verdadeiro se o primeiro argumento contém o segundo.

name (exemplo1)

`//*[name()= BBB]`

Selecciona todos os elementos que tenham nome igual a BBB.

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```


name (exemplo2)

`//*[starts-with(name(), 'B')]`

Selecciona todos os elementos que tenham nome iniciado por B.

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```


name (exemplo3)

`//*[contains(name(),'C')]`

Selecciona todos os elementos cujo nome contém a letra C.

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```


Funções: string-length

- A função string-length retorna o número de caracteres na string argumento.
- Para os operadores relacionais é necessário usar as seguintes substituições:
  - < para <
  - > para >

string-length (exemplo1)

`//*[string-length(name())=3]`

Selecciona todos os elementos que tenham o nome constituído por 3 caracteres.

```
<AAA>
  <Q/>
  <SSSS/>
  <BB/>
  <CCC/>
  <DDDDDDDD/>
  <EEEE/>
</AAA>
```

string-length (exemplo2)

`//*[string-length(name()) < 3]`

Selecciona todos os elementos que tenham o nome constituído por menos de 3 caracteres.

```
<AAA>
    <Q/>
    <SSSS/>
    <BB/>
    <CCC/>
    <DDDDDDDD/>
    <EEEE/>
</AAA>
```


string-length (exemplo3)

`//*[string-length(name()) > 3]`

Selecciona todos os elementos que tenham o nome constituído por mais de 3 caracteres.

```
<AAA>
    <Q/>
    <SSSS/>
    <BB/>
    <CCC/>
    <DDDDDDDD/>
    <EEEE/>
</AAA>
```


Combinação de endereços

- Vários selectores poderão ser combinados com o operador '|' com o significado de serem alternativos.

Combinação de end. (exemplo1)

`//BBB | //CCC`

Selecciona todos os elementos BBB e CCC.

```
<AAA>
    <BBB/>
    <CCC/>
    <DDD>
        <CCC/>
    </DDD>
    <EEE/>
</AAA>
```


Combinação de end. (exemplo2)

`//BBB | /AAA/EEE`

Selecciona todos os elementos BBB e os elementos EEE filhos de AAA.

```
<AAA>
    <BBB/>
    <CCC/>
    <DDD>
        <CCC/>
    </DDD>
    <EEE/>
</AAA>
```


Combinação de end. (exemplo3)

`//BBB | /AAA/EEE | /AAA | //DDD/CCC`

O número de combinações é ilimitado.

```
<AAA>
    <BBB/>
    <CCC/>
    <DDD>
        <CCC/>
    </DDD>
    <EEE/>
</AAA>
```

“Axis”: travessia da árvore

- O operador ‘::’ permite indicar o tipo de travessia que se faz à árvore documental.
- Por omissão, é utilizado o “axis” child (child::) o que leva a uma travessia dos filhos e por aí adiante.
- Os outros tipos de “axis” são:
  1. descendant
  2. parent
  3. ancestor
  4. following-sibling
  5. preceding-sibling
  6. following
  7. preceding
  8. descendant-or-self
  9. ancestor-or-self

child:: (exemplo1)

/AAA /child::AAA Selecciona os elementos AAA filhos da raíz.

```
<AAA>
    <BBB/>
    <CCC/>
</AAA>
```


child:: (exemplo2)

/AAA/BBB /child::AAA/child::BBB

Selecciona os elementos BBB filhos de AAA.

```
<AAA>
    <BBB/>
    <CCC/>
</AAA>
```


child:: (exemplo3)

/AAA/BBB /child::AAA/BBB

O operador pode ser colocado em evidência.

```
<AAA>
    <BBB/>
    <CCC/>
</AAA>
```


descendant:: (exemplo1)

/descendant::* Selecciona os descendentes da raíz, logo todos os nodos.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```


descendant:: (exemplo2)

/AAA/BBB/descendant::* Selecciona os descendentes de AAA/BBB.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

descendant:: (exemplo3)

`//CCC/descendant::*`

Selecciona os nodos que têm CCC como ancestral.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```


descendant:: (exemplo4)

`//CCC/descendant::*/DDD`

Selecciona os nodos DDD que têm CCC como ancestral.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```


parent:: (exemplo1)

`//DDD/parent::*`

Selecciona os elementos pai de nodos DDD.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```


ancestor:: (exemplo1)

`/AAA/BBB/DDD/CCC/EEE/ancestor*`

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

Selecciona os ancestrais de ...EEE.

ancestor:: (exemplo2)

`//FFF/ancestor*`

Selecciona os ancestrais de FFF.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
```

```
<CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```


following-sibling:: (exemplo1)

`/AAA/BBB/following-sibling::*`

Selecciona os irmãos à direita do nodo BBB.

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
```

```
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

following-sibling:: (exemplo2)

```
//CCC/following-sibling::*
```

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```


preceding-sibling:: (exemplo1)

```
/AAA/XXX/preceding-sibling::*
```

Selecciona os irmãos à esquerda do nodo XXX.

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
    <FFF>
      <GGG/>
    </FFF>
  </DDD>
</XXX>
<CCC>
  <DDD/>
</CCC>
</AAA>
```


preceding-sibling:: (exemplo2)

```
//CCC/preceding-sibling::*
```

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
    <FFF>
      <GGG/>
    </FFF>
  </DDD>
</XXX>
<CCC>
  <DDD/>
</CCC>
</AAA>
```


following:: (exemplo1)

```
/AAA/XXX/following::*
```

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
      <DDD>
        <EEE/>
      </DDD>
    </ZZZ>
    <FFF>
      <GGG/>
    </FFF>
  </BBB>
```

```
<XXX>
  <DDD>
    <EEE/>
    <DDD/>
    <CCC/>
    <FFF/>
    <FFF>
      <GGG/>
    </FFF>
  </DDD>
</XXX>
```

```
<CCC>
  <DDD/>
</CCC>
</AAA>
```


following:: (exemplo2)

```
//ZZZ/following::*
```

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
      <DDD>
        <EEE/>
      </DDD>
    </ZZZ>
    <FFF>
      <GGG/>
    </FFF>
  </BBB>
```

```
<XXX>
  <DDD>
    <EEE/>
    <DDD/>
    <CCC/>
    <FFF/>
    <FFF>
      <GGG/>
    </FFF>
  </DDD>
</XXX>
```

```
<CCC>
  <DDD/>
</CCC>
</AAA>
```


preceding:: (exemplo1)

```
/AAA/XXX/preceding::*
```

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
```

```
    <FFF>
      <GGG/>
    </FFF>
  </DDD>
</XXX>
<CCC>
  <DDD/>
</CCC>
</AAA>
```

preceding:: (exemplo2)

```
//GGG/preceding::*
```

descendant-or-self:: (exemplo1)

```
/AAA/XXX/descendant-or-self::*
```

descendant-or-self :: (exemplo2)

```
//CCC/descendant-or-self::*
```

ancestor-or-self:: (exemplo1)

```
/AAA/XXX/DDD/EEE/ancestor-or-self::*
```

ancestor-or-self :: (exemplo2)

```
//GGG/ancestor-or-self::*
```

Exercício

- Pegando na árvore do poema e centrando a referência na primeira quadra: quadra[1], calcule os seguintes conjuntos de nodos:
  - quadra[1]/ancestor*
  - quadra[1]/descendant*
  - quadra[1]/preceding*
  - quadra[1]/following*
  - quadra[1]/self*