

# **XML PATH LANGUAGE (XPATH) 2.0**

---

Engenharia de Dados e Conhecimento

2019/2020

# O que é XPath?

- XPath é uma linguagem **não-XML** para a identificação de determinadas partes de documentos XML.
- XPath permite escrever expressões que se referem, por exemplo:
  - ao primeiro elemento de um documento,
  - ao sétimo filho da pessoa descrita do terceiro elemento
  - o atributo ID da pessoa descrita no primeiro elemento cujo conteúdo é a string "Fred Jones"
  - etc.

# Para que serve?

- Habitualmente, uma expressão XPath refere **nós** de um documento XML pela sua posição absoluta, posição relativa, tipo de conteúdo ou ainda por vários outros critérios.
- Uma expressão XPath também pode representar números, *strings* ou valores lógicos.
  - Isso permite ao XSLT (que falaremos de seguida!) realizar cálculos simples como numeração, referências cruzadas de figuras, tabelas e equações.
  - A manipulação de *strings* em XPath e XSLT permite, por exemplo, transformar o título de uma notícia em maiúsculas ou extrair os dois últimos dígitos de um ano.

# Onde se utiliza?

- XSLT usa expressões XPath para combinar e seleccionar determinados elementos num documento de entrada para que sejam copiados para o documento de saída ou para tratamento adicional.
- XPointer usa expressões XPath para identificar um ponto em particular ou parte de um documento XML.
- O Schema usa expressões XPath para definir restrições de singularidade e co-ocorrência.
- XForms depende de XPath para associar os controlos de um formulário de dados, expressar restrições sobre os valores inseridos pelo utilizador e calcular os valores que dependam de outros valores.

# Características

- Uma das características interessantes sobre a utilização de XPath em combinação com o XSLT é que as consultas e as transformações podem ser executadas por aplicações sem que estas tenham conhecimento prévio da estrutura do documento em que trabalham.
- Esta é uma diferença clara relativamente aos sistemas de informação anteriores, como os RDBMS, em que o esquema das tabelas tem de ser conhecido antes de qualquer consulta poder ser executada.

# Características

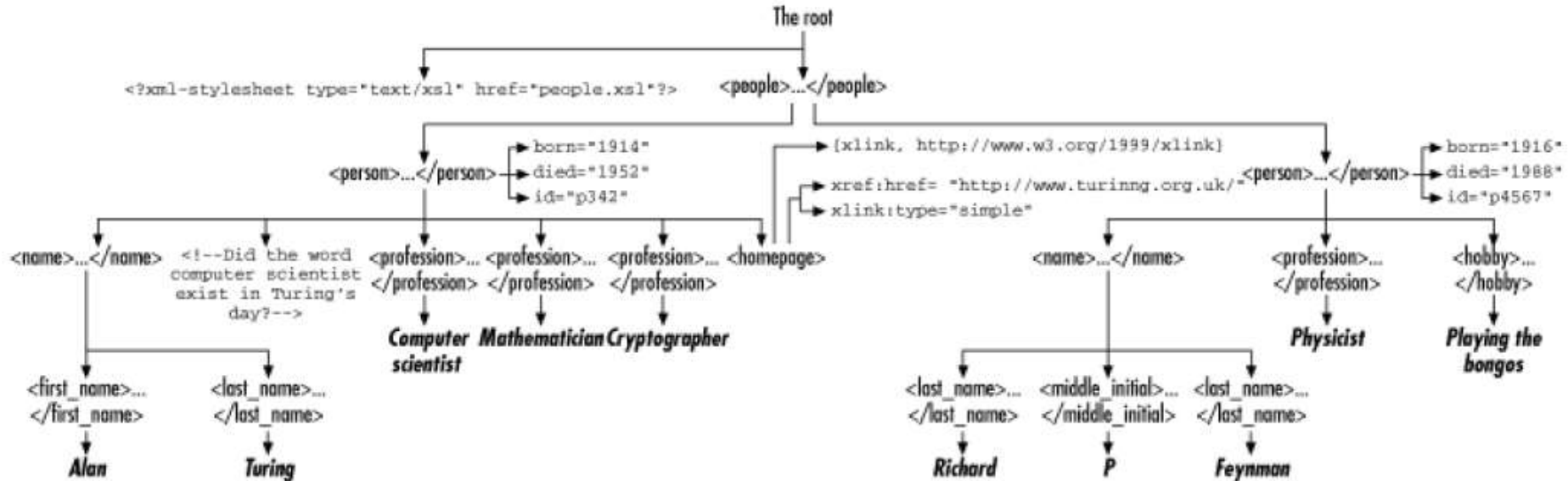
- Mesmo que isso funcione bem em muitas circunstâncias, há duas áreas em que podem ser obtidas melhorias se a estrutura dos documentos for conhecida.
  - **Otimização** – não é crucial para pequenos documentos, mas se o tamanho do documento aumenta muito, qualquer otimizador necessita de saber “onde melhorar”.
    - A primeira peça de informação para essa tarefa é a estrutura dos documentos.
  - **Comparações e ordenações** – As ordenações (numéricas ou por *string*) são indicadas nas *stylesheets* XSLT e as comparações são sempre feitas letra a letra. Selecionar ou comparar datas com diferentes fusos horários, é praticamente impossível nessas condições.
    - Assim, um *schema* com informações sobre o tipo de dados pode ajudar muito.

# Estrutura de um documento XML

- Um documento XML é percebido como sendo uma árvore de nós.
  - Há exatamente **um nó raiz** que contém todos os outros nós.
  - Alguns nós podem conter zero, um ou mais nós.
- **XPath é uma linguagem para selecionar nós e conjuntos de nós dessa árvore.**
- Na linguagem XPath há 7 tipos de nós:
  - element,
  - attribute,
  - text,
  - namespace,
  - processing-instruction,
  - comment,
  - document (root) nodes.

# Considerando o exemplo seguinte...

- Este exemplo mostra os sete tipos de elementos.





# O XML do documento de exemplo

```
1 <?xml version="1.0"?>
2 <?xml-stylesheet type="application/xml" href="people.xsl"?>
3 <!DOCTYPE people [
4     <!ATTLIST homepage xlink:type CDATA #FIXED "simple" xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
5     <!ATTLIST person id ID #IMPLIED>
6 ]>
7 <people>
8   <person born="1912" died="1954" id="p342">
9     <name>
10       <first_name>Alan</first_name>
11       <last_name>Turing</last_name>
12     </name>
13     <!-- Did the word computer scientist exist in Turing's day? -->
14     <profession>computer scientist</profession>
15     <profession>mathematician</profession>
16     <profession>cryptographer</profession>
17     <homepage xlink:href="http://www.turing.org.uk/" />
18   </person>
19   <person born="1918" died="1988" id="p4567">
20     <name>
21       <first_name>Richard</first_name>
22       <middle_initial>*</middle_initial>
23       <last_name>Feynman</last_name>
24     </name>
25     <profession>physicist</profession>
26     <hobby>Playing the bongoes</hobby>
27   </person>
28 </people>
```

# Terminologia XPath – Nós (nodes)

- Exemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person born="1918" died="1988" id="p4567">
  <name>
    <first_name>Richard</first_name>
    <middle_initial> </middle_initial>
    <last_name>Feynman</last_name>
  </name>
  <profession>physicist</profession>
  <hobby>Playing the bongoes</hobby>
</person>
```

**<person>** (document node)

**<first\_name>Richard</first\_name>** (element node)

**born="1918"** (attribute node)

**Richard** (element atomic value)

**1918** (attribute atomic value)

} Os atomic values não possuem filhos!

# Contexto

- Um dos conceitos mais importantes em XPath é o **contexto**.
- Tudo o que fazemos em XPath é interpretado relativamente ao contexto.
- Pensando no documento XML exemplo como uma hierarquia de directórios dum sistema de arquivos, podemos pensar que **person** é um directório na raiz do sistema de arquivos.
- O directório **person**, por sua vez, contém directórios chamados **name**, **profession** e **hobby**.
- Neste exemplo, o contexto é o directório actual. Se fôssemos à linha de comandos para executar o comando “`dir *.js`”, os resultados variariam de acordo com o directório actual.
- **Da mesma forma, os resultados da avaliação de uma expressão XPath variam de acordo com o contexto.**

# Relação entre os nós

- Parent - cada elemento e atributo tem um parent (pai).
- Exemplo:

```
<person born="1918" died="1988" id="p4567">  
  <name>  
    <first_name>Richard</first_name>  
    <middle_initial> &#x50;</middle_initial>  
    <last_name>Feynman</last_name>  
  </name>  
  <profession>physicist</profession>  
  <hobby>Playing the bongoes</hobby>  
</person>
```

Neste exemplo, o elemento **name** é pai de first\_name, middle\_initial e last\_name.

# Relação entre os nós

- Children - os nós podem ter zero, um ou mais filhos.
- Exemplo:

```
<person born="1918" died="1988" id="p4567">  
  <name>  
    <first_name>Richard</first_name>  
    <middle_initial>#x50;</middle_initial>  
    <last_name>Feynman</last_name>  
  </name>  
  <profession>physicist</profession>  
  <hobby>Playing the bongoes</hobby>  
</person>
```

Os elementos **name**, **profession** e **hobby** são todos filhos do elemento person

# Relação entre os nós

- Siblings - são todos aqueles elementos que possuem o mesmo pai
- Exemplo:

```
<person born="1918" died="1988" id="p4567">
  <name>
    <first_name>Richard</first_name>
    <middle_initial>#x50;</middle_initial>
    <last_name>Feynman</last_name>
  </name>
  <profession>physicist</profession>
  <hobby>Playing the bongoes</hobby>
</person>
```

**name**, **profession** e **hobby** são siblings (têm como pai o elemento person), embora sejam elementos de tipo diferente na sua estrutura:

- name é um elemento composto;
- profession e hobby são elementos simples do tipo String.

# Relação entre os nós

- Ancestors - os ancestors são os nós pais, os nós avós, os nós bisavós, etc... de um elemento
- Exemplo:

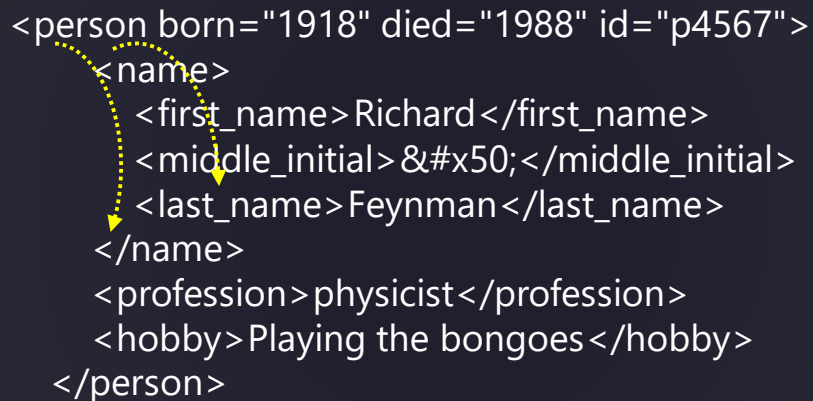
```
<person born="1918" died="1988" id="p4567">  
  <name>  
    <first_name>Richard</first_name>  
    <middle_initial> &#x50;</middle_initial>  
    <last_name>Feynman</last_name>  
  </name>  
  <profession>physicist</profession>  
  <hobby>Playing the bongoes</hobby>  
</person>
```

<first\_name> tem como ancestors <name>  
e <person>

# Relação entre os nós

- Descendants - são todos os nós filhos, netos, etc ... de um dado nó
- Exemplo:

```
<person born="1918" died="1988" id="p4567">  
  <name>  
    <first_name>Richard</first_name>  
    <middle_initial> &#x50;</middle_initial>  
    <last_name>Feynman</last_name>  
  </name>  
  <profession>physicist</profession>  
  <hobby>Playing the bongoes</hobby>  
</person>
```

A yellow dotted arrow originates from the opening tag of the root node, `<person>`, and points to the opening tag of the child node, `<name>`, illustrating the parent-child relationship.

Os descendants de `<person>` são `<name>`,  
`<first_name>`, `<middle_initial>`, `<last_name>`,  
`<profession>` e `<hobby>`



# Terminologia XPath – Nós (nodes)

- Apesar de documento (nó raiz), elementos, e atributos serem responsáveis por 90% ou mais do conteúdo dos documentos XML, ainda existem mais quatro tipos de nós: **nós de namespace**, **nós de texto**, **nós de instrução de processamento** e nós de **comentário**.
- Os nós de namespace são tratados de forma explícita.
- Os outros três tipos de nós possuem testes de nós especiais para os encontrar que são os seguintes:
  - `comment()`
  - `text ()`
  - `processing-instruction()`

# XPath Syntax – Selecção de Nós

Expressão Path	Resultado
person	selecciona todos os elementos person, dentro do contexto
/people	Selecciona o <b>root element</b> people Nota: Se o path começar com um slash ( / ) isso representa sempre um <b>path absoluto</b> para um elemento!
people/person	Selecciona todos os elementos person que sejam filhos do elemento people
//people	Selecciona todos os elementos people <u>independentemente do ponto onde se encontrem</u> no documento
people//last_name	Selecciona todos os elementos last_name que sejam <u>descendentes</u> do elemento people, independentemente do ponto onde se encontrem no documento
//@id	Selecciona todos os atributos chamados id

# Exemplos

- Como representar o nome dos cursos?
  - //curso/nome
- Como representar o guid?
  - //curso/guid
  - ou
  - //curso/@guid

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2
3  <=cursos>
4
5  <=curso guid="96">
6      <guid>96</guid>
7      <codigo>450/2007</codigo>
8      <nome>Administração e Gestão Pública</nome>
9      <grau>Mestrado</grau>
10     <bolonha>1</bolonha>
11     <vagas>40</vagas>
12     <template>0</template>
13 </curso>
14
15 <=curso guid="96">
16     <guid>54</guid>
17     <codigo>450/2007</codigo>
18     <nome>Administração Pública</nome>
19     <grau>Licenciatura</grau>
20     <bolonha>1</bolonha>
21     <vagas>60</vagas>
22     <template>0</template>
23 </curso>
24
25 <=curso guid="104">
26     <guid>104</guid>
27     <codigo>0300-0003</codigo>
28     <nome>Administração Pública</nome>
29     <grau>Licenciatura</grau>
30     <bolonha>0</bolonha>
31     <vagas>0</vagas>
32     <template>0</template>
33 </curso>
34
35 <=curso guid="155">
36     <guid>155</guid>
```

# XPath Syntax – Predicados

Expressão Path	Resultado
/people/person[1]	Selecciona o <u>primeiro</u> elemento person filho do elemento people
/people/person[last()]	Selecciona o <u>último</u> elemento person filho do elemento people
/people/person[last()-1]	Selecciona o <u>penultimo</u> elemento person filho do elemento people
/people/person[position()<3]	Selecciona os dois primeiros elementos person filhos do elemento people
//person[@id]	Selecciona todos os elementos person que possuam o atributo id
//person[@id='p4567']	Selecciona todos os elementos person que possuam o atributo id com um valor igual a 'p4567'
/people/person[@born>1918]	Selecciona os todos os elementos person filhos do elemento people com um atributo born com valor maior que 1918
/people/ person[@born>1918]/last_name	Seleciona todos os elementos last_name filhos do elemento person e netos do elemento people com um atributo born com valor maior que 1918

# XPath Syntax

- Seleção de nós desconhecidos

Expressão Path	Resultado
/people/*	Selecciona os todos os elementos filhos do elemento people
//*	Selecciona os todos os elementos no documento
//person[@*]	Selecciona os todos os elementos person que possuam um atributo

# XPath Syntax

- Expressões de Localização

Expressão Path	Resultado
child::name	Selecciona todos os elementos name filhos do corrente nó
attribute::id	Selecciona todos os atributos id no corrente nó
child::*	Selecciona todos os filhos do corrente nó
attribute::*	Selecciona todos atributos no corrente nó
child::text()	Selecciona todos os nós de texto filhos do corrente nó
child::node()	Selecciona todos os nós filhos do corrente nó
descendant::name	Selecciona todos os descendentes name do corrente nó
ancestor::name	Selecciona todos os ascendentes name do corrente nó
ancestor-or-self::name	Selecciona todos os ascendentes name do corrente nó e o próprio
child::* / child::last_name	Selecciona todos elementos last_name netos do corrente nó

# XPath Syntax - Operadores

- |
- +
- -
- \*
- div
- !=
- <
- <=
- >
- >=
- or
- and
- mod