



# MÓDULO 3: XML

## EXTENSIBLE MARKUP LANGUAGE



# MOTIVAÇÃO

## Interoperabilidade:

*Troca de informação entre sistemas incompatíveis: têm de existir conversores entre diferentes formatos, e a sua formatação tem de estar documentada*

## Legibilidade e compreensão:

São necessários documentos que possam ser lidos e compreendidos por computadores e por humanos

Têm de ser suficientemente estruturados para que a informação possa ser automaticamente processada;

Porque não usar HTML?

# XML ≠ HTML

Porque não usar HTML?

○ HTML é um padrão para a **estrutura visual** que a informação deve seguir (como a informação deve ser mostrada), **não define:**

○ **significado** dessa informação (meta-dados)

As **relações** que poderão existir entre as informações mostradas

# XML ≠ HTML

Ao passo que o HTML foi desenvolvido para mostrar os dados, o XML foi desenvolvido para **estruturar, transportar e armazenar os dados**

No HTML as *tags* são **pré-definidas**, no XML são **definidas pelo autor** (é por isso que é uma *extensible* markup language, é *extensível*, pode adaptar-se ao domínio dos dados em questão)

# O QUE É O XML?

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  <cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

# VANTAGENS DO XML

É **baseado em tags e estruturado**: os documentos têm uma estrutura em árvore

É **extensível**: as *tags* não são pré-definidas

Contêm **dados e meta-dados**

É **independente do hardware e software**: os dados são guardados no formato de texto (ASCII/UNICODE (e.g. UTF-8))

É **legível** por humanos e computadores

Pode ser automaticamente **validado** (mas só sintaxe e estrutura)



# MÓDULO 3: XML

## DOCUMENTOS XML



# DOCUMENTOS XML (EXEMPLO)

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  <cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

**PRÓLOGO**

(versão e encoding)

**ELEMENTO**

(elemento raiz)

**ATRIBUTO**

(meta-informação)

**DADOS**

(informação)



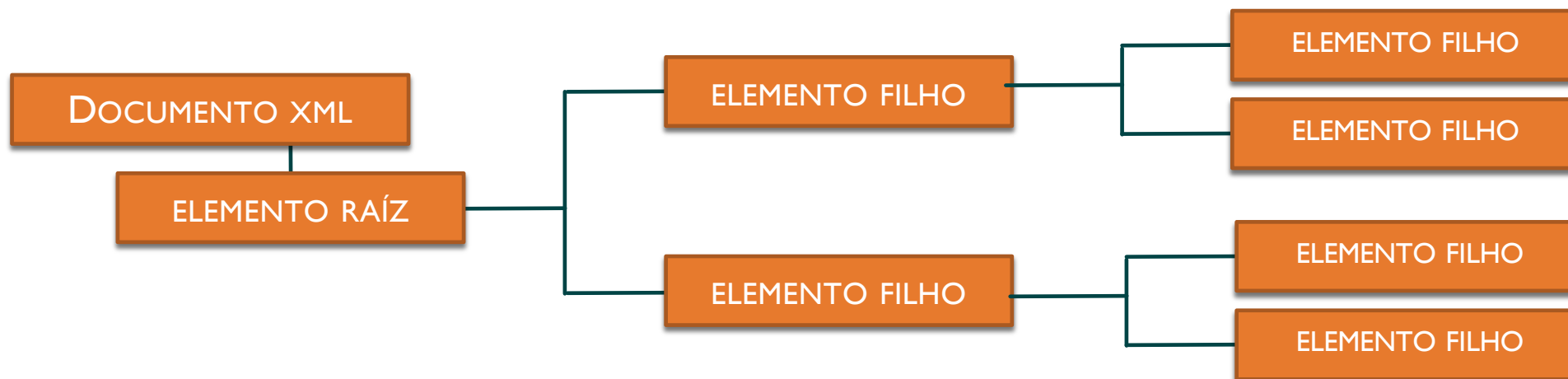
# ESTRUTURA DOS DOCUMENTOS XML

Prólogo: primeira linha do documento, especifica a sua versão e a codificação

Elemento raíz: encapsula todos os outros, é a “raíz” do documento (neste caso, indica que o documento é um **<catalogo>**)

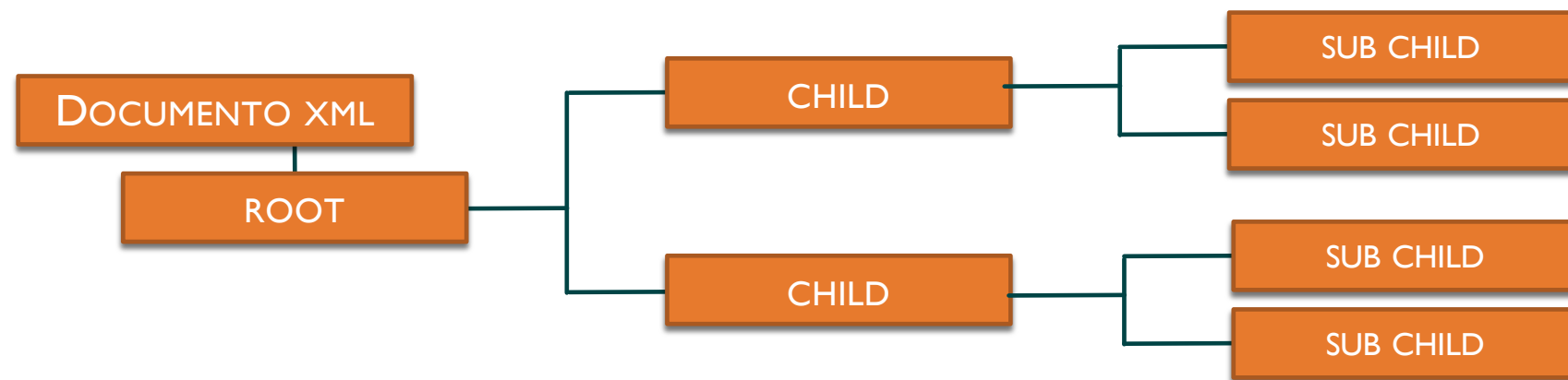
Os restantes elementos são chamados elementos-filhos (**<cd>**, **<titulo>**, **<artista>**, **<ano>**)

A estrutura dos ficheiros é **necessariamente hierárquica**



# ESTRUTURA EM ÁRVORE (XML TREE)

```
<root>
  <child>
    <subchild>...</subchild>
    <subchild>...</subchild>
    <subchild>...</subchild>
  </child>
</root>
```



Estrutura em árvore, com um **elemento raiz** e os seus **descendentes**;

As relações entre elementos são descritas pelos termos **pai** (*parent*), **filho** (*child*) e **irmão** (*sibling*);

Todos os elementos podem ter **conteúdo** (dados) e **atributos** (meta-dados)

# ESTRUTURA EM ÁRVORE (EXEMPLO)

```
<?xml version="1.0" encoding="UTF-8"?>
```

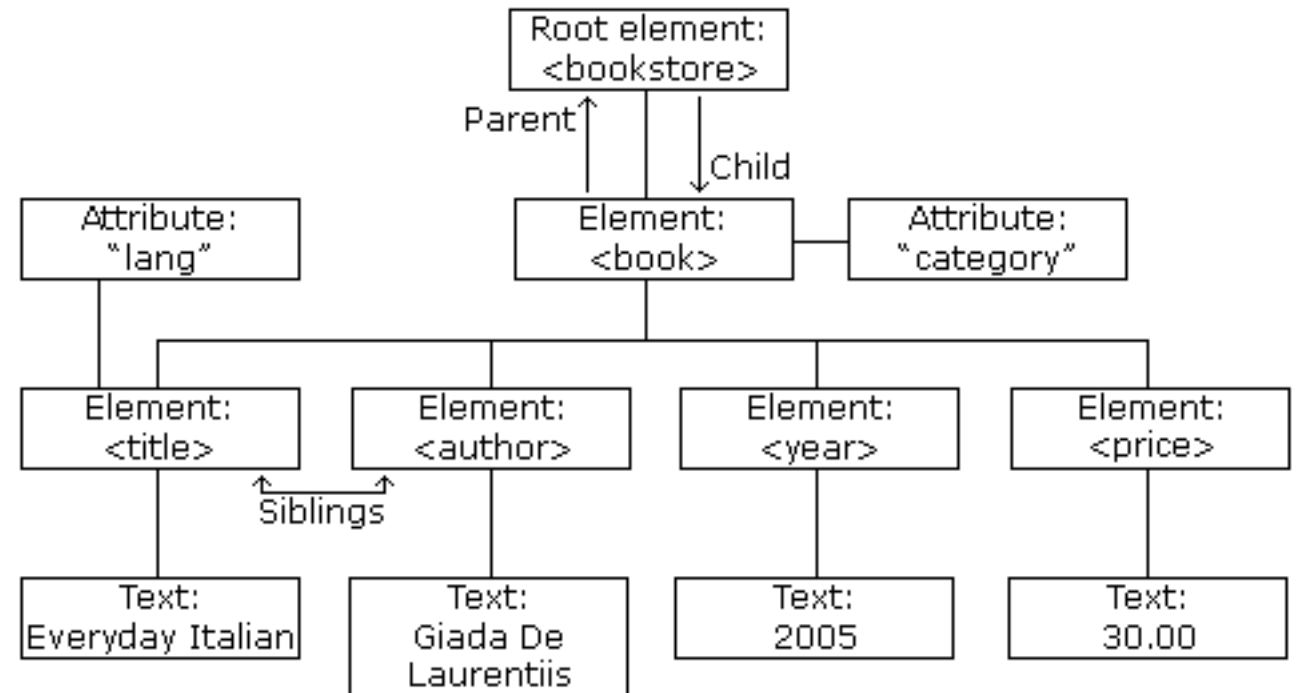
```
<bookstore>
```

```
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
```

```
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
```

```
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
```

```
</bookstore>
```



# REGRAS SINTÁTICAS DO XML (1/3)

O prólogo é obrigatório

Todos os elementos devem ter uma *tag* de abertura e de fecho:

Têm de ser escritas da mesma forma

`<titulo>`A kind of Magic`</titulo>`

Elementos sem conteúdo podem ser representados por uma única *tag* (`<titulo/>`)

O xml é *case-sensitive*: `<titulo>`  $\neq$  `<TITULO>`

Todos os atributos devem estar entre aspas: `<cd id="01">`

Todos os elementos têm de ser encadeados da forma correcta:

`<cd> <titulo> </cd> </titulo>` está incorrecto

Tem de existir sempre um elemento raiz

Os espaços em branco são preservados

## REGRAS SINTÁTICAS DO XML (2/3)

Tags XML (elementos):

Não podem começar por números ou sinais de pontuação (devem começar por uma letra ou underscore)

Não podem começar pelas letras xml: **xml**, **XML**, **XmL**, etc...

Podem conter números, letras, hífen, underscores, pontos **mas não espaços em branco!**

Comentários

Como no HTML: `<!-- Comentário -->`

# REGRAS SINTÁTICAS DO XML (2/3)

## Caracteres especiais

Os caracteres `<` e `&` são ilegais: é necessário usar as entidades de referência (`&lt;` e `&amp;`);

Existem outros 3 que não são ilegais mas que convém transformar:

- `>` para `&gt;`;

- `'` para `&apos;` (apóstrofo)

- `"` para `&quot;` (aspas)



# MÓDULO 3: XML

## ELEMENTOS E ATRIBUTOS



# ELEMENTOS XML

Um elemento xml é tudo o que se encontra entre as tags de abertura e fecho do elemento (inclusivé):

`<titulo>...</titulo>`

Os elementos xml podem conter: texto, atributos e outros elementos

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  <cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

`<titulo>`, `<artista>` e `<ano>` têm conteúdo textual;

`<catalogo>` e `<cd>` incluem outros elementos;

`<cd>` tem um atributo (`id="children"`).

**Nota:** Os atributos também é o utilizador que decide! Não têm de ser "id"!!! Por exemplo, podíamos associar cada cd ao seu estilo musical, escolhendo um atributo:

`<cd style="rock">` ou `<cd style="pop">`



# NOMEAÇÃO DOS ELEMENTOS XML: BOAS PRÁTICAS

Nomes descritivos: `<nome>`, `<ultimaConsulta>`, `<sistema_saude>`

Nomes curtos e simples: `<medico_familia>` em vez de `<nome_medico_de_familia>`

Evitar os seguintes:

- (algum software pode pensar que é uma subtracção): `<medico-familia>` (médico - familia ?!)
- . (algum software pode pensar que estamos a falar de uma propriedade de um objecto):  
`<medico.familia>`
- : (são reservados para os namespaces)

é à também podem geral alguns problemas (podem não ser suportadas pelas aplicações que usam o xml)

# NOMEAÇÃO DOS ELEMENTOS XML: ESTILOS MAIS USADOS

<ultimaconsulta>

<ULTIMACONSULTA>

<ultima\_consulta>

<UltimaConsulta>

<ultimaConsulta>

O importante é **escolher um estilo e mantê-lo** (ser consistente!)

Se os documentos XML estiverem relacionados com uma base de dados, devem ser usadas as regras de nomeação da base de dados.

# “EXTENSIBLE” MARKUP ELEMENTS: ELEMENTOS EXTENSÍVEIS

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
</catalogo>
```

Podem ser adicionados e/ou removidos elementos aos documentos xml, que as aplicações funcionam igualmente bem: **os documentos xml são extensíveis**

Catálogo de CDs:

Título: A kind of Magic  
Artista: Queen  
Ano: 1986

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <preco>20 euros</preco>
  </cd>
</catalogo>
```

Catálogo de CDs:

Título: A kind of Magic  
Artista: Queen  
Ano: (none)

# ATRIBUTOS XML

Os atributos contêm dados relacionados com um elemento em particular;  
Devem ser colocados entre aspas ou plicas:

`<cd id="01">`, `<livro categoria='romance'>`

No caso do valor do atributo conter aspas, pode proceder-se das seguintes formas:

`<agente nome='James "007" Bond'>`

`<agente nome="James &quote;007&quote; Bond">`

# ATRIBUTOS XML E METADADOS

Alguns atributos informação adicional sobre os elementos, mas normalmente essa informação não faz parte dos dados (ex: id)

```
<cd id="01">
  <titulo>A kind of Magic</titulo>
</cd>
<cd id="02">
  <titulo>The Slim Shady LP</titulo>
</cd>
```

Aqui o id serve para identificar diferentes cds, não é propriamente parte da informação acerca do cd...

```
<peessoa genero="feminino">
  <nome>Maria das Couves</nome>
</peessoa>
<peessoa genero="masculino">
  <nome>Jorge Alves</nome>
</peessoa>
```

Aqui, o género deverá ser atributo? Faz parte da caracterização de uma pessoa (são dados, não metadados?). Por isso podemos considerar passá-lo para elemento!

Metadados devem ser guardados como atributos, e dados devem ser guardados como elementos

# ELEMENTOS OU ATRIBUTOS?

Não existe uma “regra exacta” para decidir quando usar atributos ou quando usar elementos (*é deixado à consideração do autor*)

O mais consensual é usar **elementos para dados** e **atributos para informação que não é relevante para os dados**

```
<peessoa genero="feminino">  
  <nome>Maria das Couves</nome>  
</peessoa>  
<peessoa genero="masculino">  
  <nome>Jorge Alves</nome>  
</peessoa>
```

```
<peessoa>  
  <genero>feminino</genero>  
  <nome>Maria das Couves</nome>  
</peessoa>  
<peessoa>  
  <genero>masculino</genero>  
  <nome>Jorge Alves</nome>  
</peessoa>
```

Em muitos casos a informação pode ser representada quer sob a forma de atributos ou elementos... Mas, se possível, os atributos “devem ser evitados” (*usar bom-senso!*)

# DEVEMOS EVITAR ATRIBUTOS XML?

Os atributos são muito úteis em HTML. Em XML, são mais difíceis de ler e manter. Os atributos:

Não podem conter **valores múltiplos** (os elementos podem);

Não podem conter **estruturas em árvores** (os elementos podem)

Não são facilmente **extensíveis** (os elementos são)

Não descrevem **estruturas e relações** (os elementos sim)

São **mais difíceis de manipular** no código



```
<email dia="12" mes="11" ano="2002"  
to="Jorge" from="Pedro" subject="Aulas"  
mensagem="Temos aulas hoje?">  
</email>
```

# ELEMENTOS VERSUS ATRIBUTOS (EXEMPLO)

```
<email data="12/11/2002">  
  <to>Jorge</to>  
  <from>Pedro</from>  
  <subject>Aulas</subject>  
  <mensagem>Temos aulas hoje?</mensagem>  
</email>
```

```
<email>  
  <data>12-11-2002</data>  
  <to>Jorge</to>  
  <from>Pedro</from>  
  <subject>Aulas</subject>  
  <mensagem>Temos aulas hoje?</mensagem>  
</email>
```

```
<email>  
  <data>  
    <dia>12</dia>  
    <mes>11</mes>  
    <ano>2002</ano>  
  </data>  
  <to>Jorge</to>  
  <from>Pedro</from>  
  <subject>Aulas</subject>  
  <mensagem>Temos aulas hoje?</mensagem>  
</email>
```



# MÓDULO 3: XML

## EXERCÍCIOS

# EXERCÍCIO I

```
<?xml version="1.0" ?>
<email>
<to>Raquel</to>
<_bcc>Rita</_bcc>
<from>Pedro</From>
<subject>Filmes<subject>

<!-- Vou mandar para as duas -->
<mensagem>Hey! Queres ir ver o Marley & Eu ao DV?<mensagem/>
```

- O código mostrado tem erros. Corrija-os e verifique se o documento está bem formado em <https://validator.w3.org>
- Visualize o documento num browser. O resultado é o que esperava?

## EXERCÍCIO 2

| Número de Aluno | Nome              | Cadeira         | Nota |
|-----------------|-------------------|-----------------|------|
| 1234            | Zé Pedro          | Programação Web | 12   |
| 2345            | Petra Ovelha      | Web Design      | 14   |
| 3456            | Hugo Matias       | POO             | 8    |
| 4567            | Ricardo Gromêncio | POO             | 3    |

- a) Escreva o código xml correspondente à tabela.
- b) Verifique se está bem formado em <https://validator.w3.org>

# MÓDULO 2: XML

## VALIDAÇÃO

# VALIDAÇÃO DE DOCUMENTOS XML (SINTAXE E ESTRUTURA)

Um documento xml **bem formado** deve seguir as regras:

- Ter um elemento raiz

- Ter uma *tag* de fecho

- Tags* de abertura e fecho escritas da mesma forma (xml é *case-sensitive*)

- Estar devidamente encadeados

- Ter os valores dos atributos entre aspas ou plicas

## ○ XML não permite erros:

A especificação do W3C indica que as aplicações devem parar de correr se o documento XML tiver erros; erros no XML interrompem as aplicações

No HTML isto não acontece, é possível visualizar documentos com erros - ex: *tags* de fecho em falta

# VALIDAÇÃO DE DOCUMENTOS XML (SINTAXE E ESTRUTURA)

O W3C disponibiliza um serviço de validação do código XML produzido:  
<https://validator.w3.org>

# VALIDAÇÃO DE DOCUMENTOS XML (DTD E XSD)

## Bem formado $\neq$ Válido!

Um documento XML **válido** é um documento XML **bem formado** e que está de acordo com as regras de um *document type definition*

**DTD** - *Document Type Definition (original)*

**XML Schema (XSD)** - *Uma alternativa ao DTD baseada em XML*

São os *document type definition* que definem as regras e os elementos e atributos permitidos para um dado documento XML

# DTD (EXEMPLO)

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
</catalogo>
```

```
<!DOCTYPE catalogo [
  <!ELEMENT catalogo (cd*)>
  <!ELEMENT cd (titulo, artista, ano?)>
  <!ATTLIST cd CDATA#REQUIRED>
  <!ELEMENT titulo (#PCDATA)>
  <!ELEMENT artista (#PCDATA)>
  <!ELEMENT ano (#PCDATA)>
]>
```

catalogo é o elemento raiz

catalogo tem 0 ou mais cd

Cada cd tem um elemento titulo, um elemento artista e 0 ou 1 elemento ano, por esta ordem

Cada cd tem obrigatoriamente um atributo chamado id, cujo valor é textual (CDATA)

titulo, artista e ano são conteúdos textuais (#PCDATA)



# XSD (EXEMPLO)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="catalogo">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="cd" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="cd">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="titulo" />
        <xs:element ref="artista" />
        <xs:element minOccurs="0" maxOccurs="1" ref="ano" />
      </xs:sequence>
      <xs:atributo name="id" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="titulo" type="xs:string"/>
  <xs:element name="artista" type="xs:string"/>
  <xs:element name="ano" type="xs:string"/>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
</catalogo>
```

# MÓDULO 3: XML

## VISUALIZAÇÃO

# VISUALIZAÇÃO DE DOCUMENTOS XML

Os ficheiros XML, por si só, “**não fazem nada**”. São “apenas” uma forma de estruturar, armazenar e transportar informação. **É necessário escrever código para os enviar, receber, guardar, ou mostrar!**

Podem ser vistos no browser, mas não como os ficheiros HTML!

```
▼<catalogo>
  ▼<cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  ▼<cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

# VISUALIZAR DOCUMENTOS XML COM CSS

Os documentos XML não têm informação acerca de como a informação deve ser visualizada. Mas podemos usar ficheiros CSS para definir a forma como deverão ser mostradas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Com CSS -->
<?xml-stylesheet type="text/css" href="catalogo.css"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  <cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

```
catalogo {
    background-color: navy;
}
cd {
    display: block;
    margin-bottom: 40px;
}
artista {
    display: block;
    color: green;
}
ano {
    display: block;
    color: white;
}
titulo {
    color: pink;
}
```



# VISUALIZAR DOCUMENTOS XML COM XSLT

Também é possível visualizar os documentos XML usando XSLT (eXtensible Stylesheet Language Transformations). As XSLT transformam os documentos XML em HTML (mas são um pouco mais complexas...)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Com XSL -->
<?xml-stylesheet type="text/xsl" href="catalogo.xsl"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  <cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
  <html>
  <body>
    <h2>Os meus CDs</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Título</th>
        <th>Artista</th>
      </tr>
      <xsl:for-each select="catalogo/cd">
        <tr>
          <td><xsl:value-of select="titulo"/></td>
          <td bgcolor="yellow"><xsl:value-of select="artista"/></td>
          <td><xsl:value-of select="ano"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

## Os meus CDs

| Título            | Artista | Ano  |
|-------------------|---------|------|
| A kind of Magic   | Queen   | 1986 |
| The Slim Shady LP | Eminem  | 1999 |

# MÓDULO 3: XML

## PARSING

# PARSERS DE DOCUMENTOS XML

Servem para ler, criar e manipular documentos XML:

**SimpleXMLParser (PHP)** é baseado na estrutura de árvore do XML: se conhecermos a estrutura do documento (elementos, atributos, conteúdo textual), é muito simples manipular os documentos

O PHP é diferente das tecnologias do lado do cliente, é executado do lado do servidor! Precisamos por isso de um web host com PHP

Podemos instalar um web server como o **Apache** localmente (fazendo do nosso PC um servidor). O mais fácil é instalar o **XAMPP**!

# XAMPP: INSTALAÇÃO

Ir a <https://www.apachefriends.org/index.html>

Fazer a transferência e seguir as instruções

Ir ao painel de comandos do xampp (deve abrir após instalação), e fazer “Start” ao Apache Web Server

Ir à directoria onde os ficheiros foram instalados, e procurar a pasta *htdocs* (é nesta pasta que precisamos de colocar os ficheiros .php)

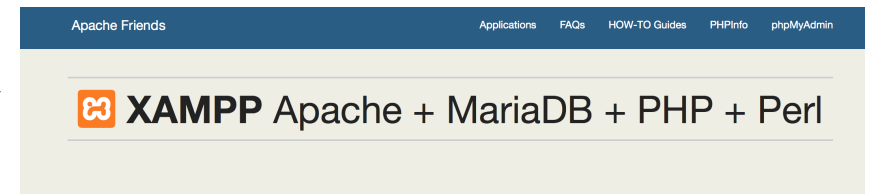
## Testar:

No browser, ir a localhost

Criar um ficheiro **exemplo.php** e testar:

Guardar na pasta **htdocs** e chamar no browser **localhost/exemplo.php**

```
<?php  
echo "O meu primeiro script PHP";  
?>
```





# SIMPLEXML: CRIAÇÃO DE DOCUMENTOS XML (EXEMPLO)

```
<?php
echo "<?xml version='1.0' encoding='UTF-8'?>";
echo "<email>";
echo "<from>Jani</from>";
echo "<to>Tove</to>";
echo "<mensagem>Remember me this weekend</mensagem>";
echo "</email>";
?>
```

```
▼<email>
  <from>Jani</from>
  <to>Tove</to>
  <mensagem>Remember me this weekend</mensagem>
</email>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<email>
  <from>Jani</from>
  <to>Tove</to>
  <mensagem>Remember me this weekend</mensagem>
</email>
```

```
<?xml version='1.0' encoding='UTF-8'?>
<email>
<from>Jani</from>
<to>Tove</to>
<mensagem>Remember me this weekend</mensagem>
</email>
```

# SIMPLEXML: TRANSFORMAÇÃO DE DOCUMENTOS XML (EXEMPLO)

```
<?php
// Fazer load do ficheiro xml (catalogo.xml)
$xml = simplexml_load_file("catalogo.xml");

// Criar um novo ficheiro com as alterações pretendidas
echo "\n";
echo "<?xml version='1.0' encoding='UTF-8'?>\n";
echo "<catalogo>\n";

foreach($xml->children() as $cd)
{
    // Adicionar um "child element" <proprietario>
    $cd->addChild("proprietario", "Miriam Santos");

    // Fazer echo ao resto do ficheiro:
    echo "<cd id='" . $cd->attributes()->id . "'>\n";
    echo "<titulo>". $cd->titulo . "</titulo>\n";
    echo "<artista>". $cd->artista . "</artista>\n";
    echo "<ano>". $cd->ano . "</ano>\n";
    echo "<proprietario>". $cd->proprietario . "</proprietario>\n";
    echo "</cd>\n";
}
echo "</catalogo>";

?>
```

```
<?xml version='1.0' encoding='UTF-8'?>
<catalogo>
<cd id='01'>
<titulo>A kind of Magic</titulo>
<artista>Queen</artista>
<ano>1986</ano>
<proprietario>Miriam Santos</proprietario>
</cd>
<cd id='02'>
<titulo>The Slim Shady LP</titulo>
<artista>Eminem</artista>
<ano>1999</ano>
<proprietario>Miriam Santos</proprietario>
</cd>
</catalogo>
```

# SIMPLEXML: LEITURA E DISPLAY DE DOCUMENTOS XML (EXEMPLO)

```
<?php
// Fazer load do ficheiro xml (catalogo.xml)
$xml = simplexml_load_file("catalogo.xml");
echo "<h2>O meu catálogo de cds</h2><br />";

foreach($xml->children() as $cd)
{

    echo "CD : ".$cd->attributes()->id."<br />";
    echo "Título : ".$cd->titulo."<br />";
    echo "Artista : ".$cd->artista."<br />";
    echo "Ano : ".$cd->ano."<br />";
    echo "<hr/>";

}
?>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <cd id="01">
    <titulo>A kind of Magic</titulo>
    <artista>Queen</artista>
    <ano>1986</ano>
  </cd>
  <cd id="02">
    <titulo>The Slim Shady LP</titulo>
    <artista>Eminem</artista>
    <ano>1999</ano>
  </cd>
</catalogo>
```

## O meu catálogo de cds

CD : 01  
Título : A kind of Magic  
Artista : Queen  
Ano : 1986

---

CD : 02  
Título : The Slim Shady LP  
Artista : Eminem  
Ano : 1999

# REFERÊNCIAS

Tutoriais da W3Schools:

<http://www.w3schools.com/xml/default.asp>

[http://www.w3schools.com/xml/xml\\_parser.asp](http://www.w3schools.com/xml/xml_parser.asp)

[http://www.w3schools.com/php/php\\_ref\\_simplexml.asp](http://www.w3schools.com/php/php_ref_simplexml.asp)

[http://www.w3schools.com/php/php\\_xml\\_parsers.asp](http://www.w3schools.com/php/php_xml_parsers.asp)

# UTILIZAÇÕES DE XML

## Webservice

O que é?

Qualquer peça de software que está disponível através da internet e usa um sistema de mensagens XML padronizado

HTML = utilizador-máquina

XML/SOAP = máquina-máquina

# UTILIZAÇÕES DE XML

## Webservice

- Aproveitar as características da Web

  - Mesma infraestrutura

  - Mesmo modelo

  - Acessível a qualquer um

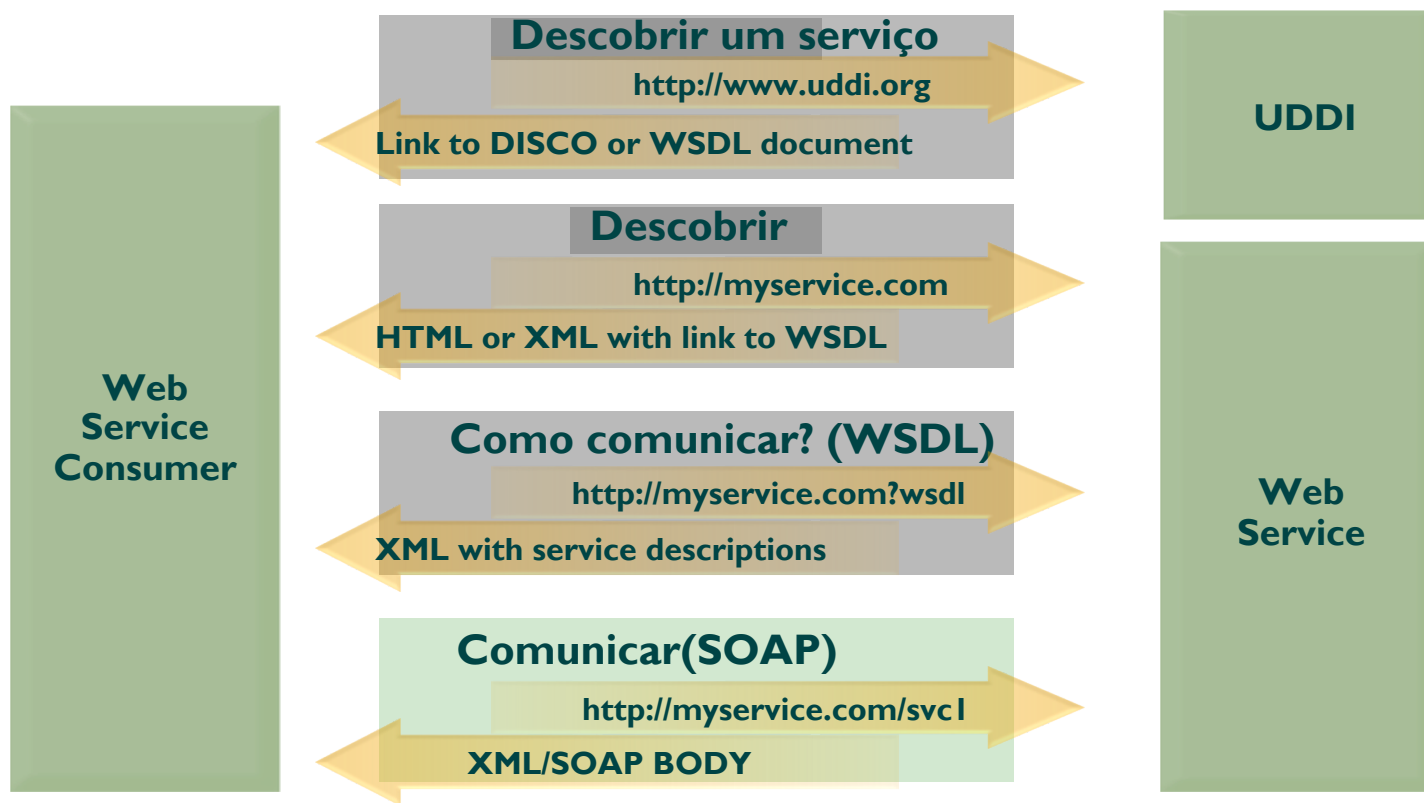
- Aplicações distribuídas verdadeiramente escaláveis

  - Sem estado e levemente associadas

  - Ambas acessíveis através da Internet e Intranet

# UTILIZAÇÕES DE XML

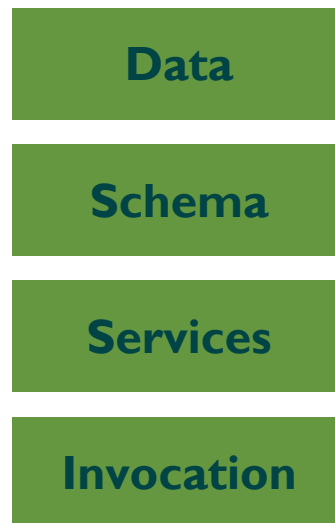
## Webservice – como funciona?



# UTILIZAÇÕES DE XML

## Webservices com .NET Mapeamento bidirecional

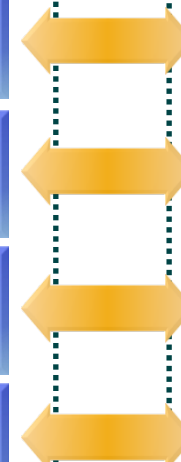
### Conceitos de aplicação



### Web



### Programas





# UTILIZAÇÕES DE XML

Webservice – como utilizar?

Existem 3 maneiras de consumir um Webservice:

- Utilizando o método HTTP-POST

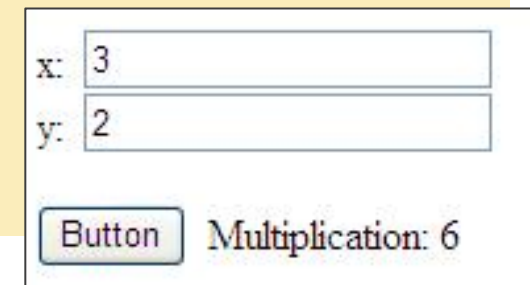
- Utilizando XMLHttp que irá usar SOAP para chamar o serviço

- Utilizando a classe gerada WSDL

# UTILIZAÇÕES DE XML

## Webservice – como utilizar? Utilizando o método HTTP-POST

```
<html>
  <body>
    <form action="http://localhost/.../Service.asmx/Multiply" method="POST">
      <input name="x"></input>
      <input name="y"></input>
      <input type="submit" value="Enter"> </input>
    </form>
  </body>
</html>
```



A screenshot of a web form. It contains two input fields: the first is labeled 'x:' and contains the value '3'; the second is labeled 'y:' and contains the value '2'. Below these fields is a button labeled 'Button'. To the right of the button, the text 'Multiplication: 6' is displayed, indicating the result of the calculation.

## Webservice – como utilizar?

Utilizando XMLHttp que irá usar SOAP para chamar

```
public class OrderProcessor
{
    public void SubmitOrder(PurchaseOrder order) {...}
}
```

```
public class PurchaseOrder
{
    public string ShipTo;
    public string BillTo;
    public string Comment;
    public Item[] Items;
    public DateTime OrderDate;
}
```

```
public class OrderProcessor
{
    [WebMethod]
    public void SubmitOrder(PurchaseOrder order) {...}
}

[XmlRoot("Order", Namespace="urn:acme.b2b-schema.v1")]
public class PurchaseOrder
{
    [XmlElement("shipTo")] public string ShipTo;
    [XmlElement("billTo")] public string BillTo;
    [XmlElement("comment")] public string Comment;
    [XmlElement("items")] public Item[] Items;
    [XmlAttribute("date")] public DateTime OrderDate;
}
```

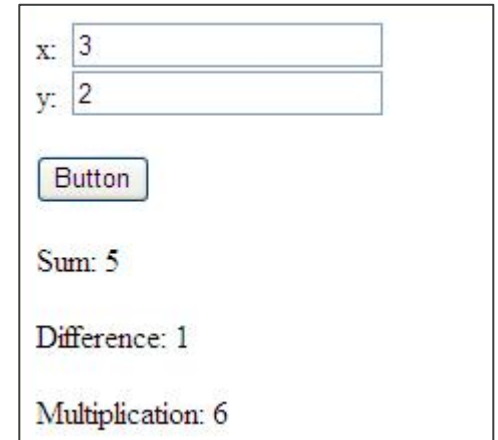
```
PurchaseOrder po = new PurchaseOrder();
po.ShipTo = "Anders Hejlsberg";
po.BillTo = "Bill Gates";
po.OrderDate = DateTime.Today;
...
OrderProcessor.SubmitOrder(po);
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope>
  <soap:Body>
    <SubmitOrder>
      <Order date="20010703">
        <shipTo>Anders Hejlsberg</shipTo>
        <billTo>Bill Gates</billTo>
        <comment>Overnight delivery</comment>
        <items>
          <productId>17748933</productId>
          <description>Dom
Perignon</description>
        </items>
      </Order>
    </SubmitOrder>
  </soap:Body>
</soap:Envelope>
```

# UTILIZAÇÕES DE XML

## Webservice – como utilizar? Utilizando a classe gerada WSDL

```
<?php
define('CALCULATOR_WSDL_URL', 'https://ssl.INNOSYSTEMS.net/cgi-bin/xml/xml.cgi?WSDL');
$wsdl = array();
$wsdl[CalculatorWsdClass::WSDL_URL] = CALCULATOR_WSDL_URL;
$calculatorServiceCalculate->calculateProduct(new CalculatorStructCalculateProduct($x, $y)
?>
```



A screenshot of a web-based calculator interface. It features two input fields at the top: 'x:' with the value '3' and 'y:' with the value '2'. Below these is a button labeled 'Button'. Underneath the button, the results of calculations are displayed: 'Sum: 5', 'Difference: 1', and 'Multiplication: 6'.

# WEBSERVICE - EXEMPLO

<http://www.webservicex.net/globalweather.asmx?op=GetWeather>

## GetWeather

Get weather report for all major cities around the world.

## Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

| Parameter                             | Value                |
|---------------------------------------|----------------------|
| CityName:                             | <input type="text"/> |
| CountryName:                          | <input type="text"/> |
| <input type="button" value="Invoke"/> |                      |

## SOAP 1.1

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

```
POST /globalweather.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.webserviceX.NET/GetWeather"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetWeather xmlns="http://www.webserviceX.NET">
      <CityName>string</CityName>
      <CountryName>string</CountryName>
    </GetWeather>
  </soap:Body>
</soap:Envelope>
```

## PROPRIEDADES AVANÇADAS: XPATH

- XPath é uma sintaxe utilizada para selecionar parte de um documento XML
- caminho que o Xpath descreve para os elementos é similar ao que o Sistema operativo utiliza para os seus ficheiros
- XPath é em si uma pequena linguagem de programação: tem funções, testes e expressões
- É um padrão W3C
- Não é escrito em XML mas é extremamente utilizado em XSLT

# PROPRIEDADES AVANÇADAS: XPATH

## Sistema operativo

/ = the root directory

/users/dave/foo =  
the (one) file named  
foo in dave in users  
foo = the (one) file named  
foo in the current directory

. = the current directory

.. = the parent directory

/users/dave/\* = all the files  
in /users/dave

## XPath

/library = the root element (if named library )

/library/book/chapter/section = every section  
element in a chapter in every book in the library

section = every section element that is a child of  
the current element

. = the current element

.. = parent of the current element

/library/book/chapter/\* = all the  
elements in /library/book/chapter

# PROPRIEDADES AVANÇADAS: XPATH

Um caminho que começa por `/` representa um caminho absoluto, que se inicia no topo do documento

Exemplo: `/email/message/header/from`

De referir que o caminho absoluto pode indicar mais do que um elemento

Uma barra por si só significa o documento inteiro

O caminho que não começa por `/` representa um caminho a partir do elemento atual

Exemplo: `header/from`

Um caminho que começa por `//` pode começar de qualquer lado

Exemplo: `//header/from` seleciona cada elemento `from` que é filho do `header`

Isto pode ser dispendioso, uma vez que envolve pesquisar no documento inteiro



# PROPRIEDADES AVANÇADAS: XPATH

Um número em parênteses retos seleciona um filho particular correspondente

Exemplo: `/library/book[1]` seleciona o primeiro **book** da **library**

Exemplo: `//chapter/section[2]` seleciona a segunda **section** de cada **chapter** do documento XML

Exemplo: `//book/chapter[1]/section[2]`

Apenas elementos correspondentes são selecionados; por exemplo se o livro contiver ambas **sections** e **exercises**, a última é ignorada na contagem das **sections**

A função **last()** em `[ ]` seleciona o último nó correspondente à pesquisa

Exemplo: `/library/book/chapter[last()]`

Podemos também utilizar aritmética simples

Exemplo para o penúltimo elemento: `/library/book/chapter[last()-1]`

# PROPRIEDADES AVANÇADAS: XPATH

Um asterisco significa todos os elementos deste nível

Exemplo: `/library/book/chapter/*` seleciona cada filho de cada `chapter` de cada `book` na `library`

Exemplo: `//book/*` seleciona cada filho de cada `book` (`chapters`, `tableOfContents`, `index`, etc.)

Exemplo: `/*/*/*/paragraph` seleciona cada `paragraph` que tem exatamente 3 antecessor

Exemplo: `//*` seleciona cada elemento em todo o documento

# PROPRIEDADES AVANÇADAS: XPATH

Podemos seleccionar elementos por si mesmos ou elementos que têm certos atributos

Um atributo consiste num par nome-valor, p.e. `<chapter num="5">`

Para seleccionar por atributo, utilizar o prefixo `@`

Exemplo: `@num` ira seleccionar cada atributo chamado `num`

Exemplo: `//@*` ira seleccionar cada atributo chamado , *em qualquer lado* no documento

Para seleccionar elementos que tenham um determinado atributo, basta colocar o nome do atributo nos parênteses retos `[ ]`

Exemplo: `//chapter[@num]` vai seleccionar cada elemento `chapter` (em todo o documento que tenha um atributo chamado `num`)

# PROPRIEDADES AVANÇADAS: XPATH

`//chapter[@num]` seleciona cada elemento **chapter** com um atributo **num**

`//chapter[not(@num)]` seleciona cada elemento **chapter** que *não* tem um atributo **num**

`//chapter[@*]` seleciona cada elemento **chapter** que tem qualquer atributo

`//chapter[not(@*)]` seleciona cada elemento **chapter** sem atributos

# PROPRIEDADES AVANÇADAS: XPATH

`//chapter[@num='3']` seleciona cada elemento **chapter** com um atributo **num** e valor **3**

`//chapter[not(@num)]` seleciona cada elemento **chapter** que *não* tem um atributo **num**

`//chapter[@*]` seleciona cada elemento **chapter** que tem *qualquer* atributo

`//chapter[not(@*)]` seleciona cada elemento **chapter** sem atributos

A função **normalize-space()** pode ser usada para remover espaços no princípio e fim da palavra para comparações

Exemplo: `//chapter[normalize-space(@num)="3"]`

# PROPRIEDADES AVANÇADAS: XPATH

Um eixo (axis) é um conjunto de nós relativos a um determinado nó; **X::Y** significa “escolher **Y** do eixo **X**”

**self::** é o conjunto dos nós atuais

**self::node()** é o nó atual

**child::** é o normal por omissão, então **/child::X** é equivalente a **/X**

**parent::** é o pai do nó atual

**ancestor::** são todos os antecessores do nó atual, incluindo raiz

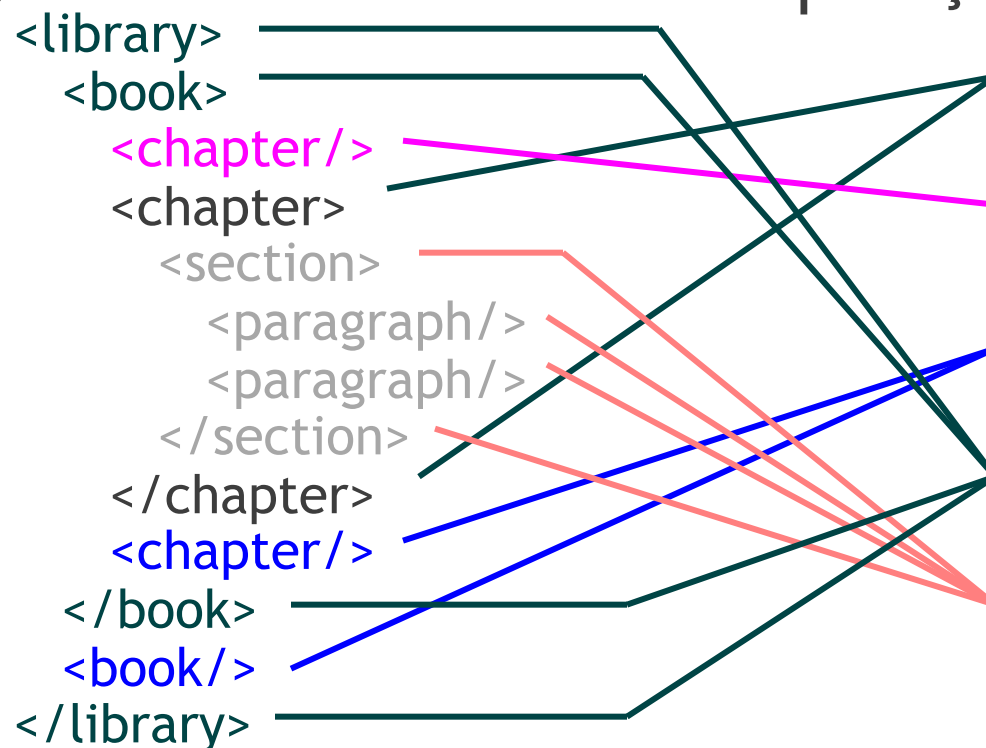
**descendant::** são todos os descendentes do nó atual

**preceding::** tudo antes do nó atual no documento inteiro XML

**following::** tudo depois do nó atual no documento inteiro XML

# PROPRIEDADES AVANÇADAS: XPATH

Começando num determinado nó, os eixos **self**, **preceding**, **following**, **ancestor**, e **descendant** formam a partição de todos os nós



//chapter[2]/self::\*

//chapter[2]/preceding::\*

//chapter[2]/following::\*

//chapter[2]/ancestor::\*

//chapter[2]/descendant::\*

# PROPRIEDADES AVANÇADAS: XPATH

## Expressões aritméticas

**+** add

**-** subtract

**\*** multiply

**div** (not **/**) divide

**mod** modulo (remainder)



# PROPRIEDADES AVANÇADAS: XPATH

= “equal to” (de notar que não é ==)

!= “not equal to”

Pode não ser assim tão simples!

*value = node-set* vai ser verdadeiro se *node-set* contém qualquer nó com um valor que coincide com *value*

*value != node-set* vai ser verdadeiro se *node-set* contém qualquer nó com um valor que não coincide com *value*

Portanto,

*value = node-set* e *value != node-set* podem ser verdadeiros simultaneamente!

# PROPRIEDADES AVANÇADAS: XPATH

○ XPath contém algumas funções que operam sobre conjuntos de nós, números ou strings:

**count(*elem*)** conta o número de elementos selecionados

Exemplo: `//chapter[count(section)=1]` seleciona os **chapters** com exatamente dois filhos **section**

**name()** retorna o nome do elemento

Exemplo: `//*[name()='section']` é similar a `//section`

**starts-with(*arg1*, *arg2*)** testa se *arg1* começa com *arg2*

Exemplo: `//*[starts-with(name(), 'sec']`

**contains(*arg1*, *arg2*)** testa se *arg1* contém *arg2*

Exemplo: `//*[contains(name(), 'ect']`