XML

tutorial 0

bruno cuconato

Laboratório de Políticas Públicas da FGV

XML: o que é

<definition> <term>XML</term>, the Extensible Markup Language, is a W3C-endorsed standard for document markup. It defines a generic syntax used to mark up data with simple, human-readable tags. It provides a standard format for computer documents that is flexible enough to be customized for domains as diverse as web sites, electronic data interchange, vector graphics, genealogy, real estate listings, object serialization, remote procedure calls, voice mail systems, and more.</definition>

XML: o que é

um exemplo

```
<?xml version="1.0"?>
<product barcode="2394287410">
<manufacturer>Verbatim</manufacturer>
<name>DataLife MF 2HD</name>
<quantity>10</quantity>
<size>3.5"</size>
<color>black</color>
<description>floppy disks</description>
</product>
```

- **nomenclatura**: elements, names, markup, tags, attributes, content, root.

XML: o que é

um exemplo

```
<?xml version="1.0"?>
<product barcode="2394287410">
<manufacturer>Verbatim</manufacturer>
<name>DataLife MF 2HD</name>
<quantity>10</quantity>
<size>3.5"</size>
<color>black</color>
<description>floppy disks</description>
</product>
```

- **nomenclatura**: elements, names, markup, tags, attributes, content, root.
- todo documento XML é uma árvore.

XML: o que é

- uma linguagem de *meta*markup.

XML: o que é

- uma linguagem de *meta*markup.
- linguagens como o markdown deste documento descrevem forma:

XML: o que é

- uma linguagem de *meta*markup.
- linguagens como o markdown deste documento descrevem forma:
  - se eu escrevo `*itálico*`, o computador sabe que deve mostrar esse conteúdo como *itálico*.

XML: o que é

- uma linguagem de *meta*markup.
- linguagens como o markdown deste documento descrevem forma:
  - se eu escrevo `*itálico*`, o computador sabe que deve mostrar esse conteúdo como *itálico*.
  - mas se eu escrevo em itálico porque estou usando uma palavra estrangeira ou por que se trata do título de um livro em uma citação, para o computador não há diferença.

XML: o que é

- uma linguagem de *meta*markup.
- linguagens como o markdown deste documento descrevem forma:
  - se eu escrevo `*itálico*`, o computador sabe que deve mostrar esse conteúdo como *itálico*.
  - mas se eu escrevo em itálico porque estou usando uma palavra estrangeira ou por que se trata do título de um livro em uma citação, para o computador não há diferença.
  - exemplos:

XML: o que é

- uma linguagem de *meta*markup.
- linguagens como o markdown deste documento descrevem forma:
  - se eu escrevo `*itálico*`, o computador sabe que deve mostrar esse conteúdo como *itálico*.
  - mas se eu escrevo em itálico porque estou usando uma palavra estrangeira ou por que se trata do título de um livro em uma citação, para o computador não há diferença.
  - exemplos:
    - *astonishing*, *dom casmurro*

estrutura, não conteúdo

em uma linguagem como o XML, essas representações dizem respeito à **estrutura**, e não à *forma*.

> *XML is a structural and semantic markup language, not a presentation language.*

estrutura, não conteúdo

- assim, o exemplo anterior ficaria como:

```
<language code="en">astonishing</language>,
<title>dom casmurro</title>
```

estrutura, não conteúdo

- assim, o exemplo anterior ficaria como:

  ```
  <language code="en">astonishing</language>,
  <title>dom casmurro</title>
  ```

- ao contrário de linguagens como markdown, a sintaxe do XML é infinita: nós podemos criar as **tags** que quisermos, desde que obedeçamos às regras sobre sua formação.[1]

[1] isso quer dizer que as tags do exemplo são arbitrárias.

estrutura, não conteúdo

- o que isso muda?

estrutura, não conteúdo

o fato de não estarmos declarando como queremos que um texto seja mostrado – e sim o que ele representa – nos permite separar estrutura e forma.

estrutura, não conteúdo

- imagine que as normas de citação da ABNT mudaram, e agora títulos devem ser colocados em negrito, e não italicizados.

estrutura, não conteúdo

- imagine que as normas de citação da ABNT mudaram, e agora títulos devem ser colocados em negrito, e não italicizados.
- se o nosso documento-fonte estivesse em markdown, teríamos de fazer as mudanças manualmente.

estrutura, não conteúdo

- imagine que as normas de citação da ABNT mudaram, e agora títulos devem ser colocados em negrito, e não italicizados.
- se o nosso documento-fonte estivesse em markdown, teríamos de fazer as mudanças manualmente.
- se o nosso documento-fonte está em XML, ele fica como está. o que precisa mudar é o programa que lê o XML e o formata para display.

```
if element['type']=='title':
    italicize(element['content'])
```

vira

```
if element['type']='title':
    bold(element['content'])
```

sintaxe

- assim como a maior parte das linguagens de markup, o XML tem uma sintaxe própria que deve ser respeitada.

sintaxe

- assim como a maior parte das linguagens de markup, o XML tem uma sintaxe própria que deve ser respeitada.

  um documento XML que não fere a sintaxe é chamado de ***bem-formado****.*

sintaxe

elementos

- todo elemento teve ter começar com uma start-tag (`<tag-name>`) e terminar com uma end-tag (`</tag-name>`). elementos vazios podem ser escritos como o usual ou como `<tag-name />`.

sintaxe

elementos

- todo elemento teve ter começar com uma start-tag (`<tag-name>`) e terminar com uma end-tag (`</tag-name>`). elementos vazios podem ser escritos como o usual ou como `<tag-name />`.
  - XML é case-sensitive: `<person>` e `</Person>` são coisas diferentes!

sintaxe

elementos

- todo elemento teve ter começar com uma start-tag (`<tag-name>`) e terminar com uma end-tag (`</tag-name>`). elementos vazios podem ser escritos como o usual ou como `<tag-name />`.
  - XML é case-sensitive: `<person>` e `</Person>` são coisas diferentes!
- root, parent, child, e sibling elements: nomenclatura usual de árvores.

sintaxe

elementos

- todo elemento teve ter começar com uma start-tag (`<tag-name>`) e terminar com uma end-tag (`</tag-name>`). elementos vazios podem ser escritos como o usual ou como `<tag-name />`.
  - XML é case-sensitive: `<person>` e `</Person>` são coisas diferentes!
- root, parent, child, e sibling elements: nomenclatura usual de árvores.
  - toda criança tem exatamente um parente.

sintaxe

elementos

- todo elemento teve ter começar com uma start-tag (`<tag-name>`) e terminar com uma end-tag (`</tag-name>`). elementos vazios podem ser escritos como o usual ou como `<tag-name />`.
  - XML é case-sensitive: `<person>` e `</Person>` são coisas diferentes!
- root, parent, child, e sibling elements: nomenclatura usual de árvores.
  - toda criança tem exatamente um parente.
  - sem overlap: `<strong><em>this common example from HTML</strong></em>` não pode!

sintaxe

elementos

- todo elemento teve ter começar com uma start-tag (`<tag-name>`) e terminar com uma end-tag (`</tag-name>`). elementos vazios podem ser escritos como o usual ou como `<tag-name />`.
  - XML é case-sensitive: `<person>` e `</Person>` são coisas diferentes!
- root, parent, child, e sibling elements: nomenclatura usual de árvores.
  - toda criança tem exatamente um parente.
  - sem overlap: `<strong><em>this common example from HTML</strong></em>` não pode!
  - só um elemento raiz por documento!

sintaxe

atributos

- elementos podem ter atributos:

sintaxe

atributos

- elementos podem ter atributos:

  *An **attribute** is a name-value pair attached to the element's start-tag.*

sintaxe

atributos

- elementos podem ter atributos:

  *An **attribute** is a name-value pair attached to the element's start-tag.*

  - os nomes são separados dos valores por um sinal de =, e valores são englobados por " ou '.

sintaxe

atributos

- elementos podem ter atributos:

  *An **attribute** is a name-value pair attached to the element's start-tag.*

  - os nomes são separados dos valores por um sinal de =, e valores são englobados por " ou '.
- escaping de " e ' não é o esperado: `&quot;` e `&apos;`. mas podemos fazer `<publisher name="O'Reilly"/>`.

sintaxe

atributos

- elementos podem ter atributos:

  *An **attribute** is a name-value pair attached to the element's start-tag.*

  - os nomes são separados dos valores por um sinal de =, e valores são englobados por " ou '.
- escaping de " e ' não é o esperado: `&quot;` e `&apos;`. mas podemos fazer `<publisher name="O'Reilly"/>`.
- whitespace é opcional! (`<name first='donald' last = "knuth" />`)

sintaxe

atributos

```
<person>
<name first="Alan" last="Turing"/>
</person>
```

ou

```
<person>
<name>
<first-name>Alan</first-name>
<last-name>Turing</last-name>
</name>
</person>
```

sintaxe

atributos

- qual escolher?
  tanto faz.

sintaxe

atributos

- qual escolher?
  tanto faz.
- qual é melhor?

sintaxe

nomes

- regras para nomes de elementos e de atributos são as mesmas (ufa!)

sintaxe

nomes

- regras para nomes de elementos e de atributos são as mesmas (ufa!)
- podem conter qualquer caractere alfanumérico (inclui UTF-8, se o documento for UTF-8), além de `.-_` (ponto, hífen, underscore) – mas nenhum outro símbolo de pontuação.

sintaxe

nomes

- regras para nomes de elementos e de atributos são as mesmas (ufa!)
- podem conter qualquer caractere alfanumérico (inclui UTF-8, se o documento for UTF-8), além de `.-_` (ponto, hífen, underscore) – mas nenhum outro símbolo de pontuação.
- os nomes não podem começar com números, hífen ou ponto, nem com a string `XML`, que é de uso reservado.

sintaxe

referências

- nós vimos que em nomes de atributos potencialmente precisaríamos de escapar apóstrofos e aspas com `&quot;` e `&apos;`. essas strings são chamadas de **entity references**, e não são as únicas:

sintaxe

referências

- nós vimos que em nomes de atributos potencialmente precisaríamos de escapar apóstrofos e aspas com `&quot;` e `&apos;`. essas strings são chamadas de **entity references**, e não são as únicas:
  - `<`: “less than”, ou <.

sintaxe

referências

- nós vimos que em nomes de atributos potencialmente precisaríamos de escapar apóstrofos e aspas com `&quot;` e `&apos;`. essas strings são chamadas de **entity references**, e não são as únicas:
  - `<`: “less than”, ou <.
  - `&`: “ampersand”, ou &.

sintaxe

referências

- nós vimos que em nomes de atributos potencialmente precisaríamos de escapar apóstrofos e aspas com `&quot;` e `&apos;`. essas strings são chamadas de **entity references**, e não são as únicas:
  - `<`: “less than”, ou <.
  - `&`: “ampersand”, ou &.
  - `>`: “greater than”, ou >.

sintaxe

toda ocorrência desses 3 símbolos no conteúdo de um elemento deve ser escapada substituindo o caractere em questão pela sua entity reference.

sintaxe

comentários

```
<!-- um comentário simples. -->
```

- podem aparecer em qualquer lugar do documento.

sintaxe

comentários

```
<!-- um comentário simples. -->
```

- podem aparecer em qualquer lugar do documento.
- não podem conter `--` nem terminar em `--->` (três hífens).

sintaxe

processing instruction

- ao contrário do comentário, que instrui humanos, as instruções de processamento instruem máquinas

sintaxe

processing instruction

- ao contrário do comentário, que instrui humanos, as instruções de processamento instruem máquinas
- exemplo:

  ```
  <?xml-stylesheet href="person.css" type="text/css"?>
  ```

  diz ao XML parser que ele deve usar o arquivo `person.css` para estilizar o documento XML que contém essa declaração.

sintaxe

the XML declaration

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

- **se houver, deve ser a primeira linha do documento!**

sintaxe

the XML declaration

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

- **se houver, deve ser a primeira linha do documento!**
- versão: 1.0, apesar da 1.1 existir (ver link)

sintaxe

the XML declaration

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

- **se houver, deve ser a primeira linha do documento!**
- versão: 1.0, apesar da 1.1 existir (ver link)
- encoding: assume UTF-8 se ausente.

sintaxe

the XML declaration

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

- **se houver, deve ser a primeira linha do documento!**
- versão: 1.0, apesar da 1.1 existir (ver link)
- encoding: assume UTF-8 se ausente.
- standalone: `no` se o documento contiver DDT externa. se ausente, `no` é assumido.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.
2. Elements may nest but may not overlap.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.
2. Elements may nest but may not overlap.
3. There must be exactly one root element.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.
2. Elements may nest but may not overlap.
3. There must be exactly one root element.
4. Attribute values must be quoted.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.
2. Elements may nest but may not overlap.
3. There must be exactly one root element.
4. Attribute values must be quoted.
5. An element may not have two attributes with the same name.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.
2. Elements may nest but may not overlap.
3. There must be exactly one root element.
4. Attribute values must be quoted.
5. An element may not have two attributes with the same name.
6. Comments and processing instructions may not appear inside tags.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

sintaxe

resumo para checagem de well-formedness[2]

1. Every start-tag must have a matching end-tag.
2. Elements may nest but may not overlap.
3. There must be exactly one root element.
4. Attribute values must be quoted.
5. An element may not have two attributes with the same name.
6. Comments and processing instructions may not appear inside tags.
7. No unescaped $<$ or & signs may occur in the character data of an element or attribute.

[2] `xmllint` é um programa comum em GNU/Linux que checa e valida arquivos XML.

validação

- além da sintaxe presente em toda linguagem, o XML também permite que um documento seja validado por um esquema.

validação

- além da sintaxe presente em toda linguagem, o XML também permite que um documento seja validado por um esquema.

  se um documento XML obedece às regras de um dado esquema, ele é considerado ***válido*** *naquele esquema.*

validação

- há mais de uma forma de validar documentos XML.

validação

- há mais de uma forma de validar documentos XML.
  - podemos usar as DTDs (*Document Type Definitions*) ou linguagens específicas para isso, como a XML Schema.[3]

[3] mais sobre as formas de validação em outro tutorial.

validação

e para quê isso serve?

validação

- a validação é útil para prevenir erros, ou definir qual subconjunto da linguagem XML deve ser usado para a representação de um certo domínio

validação

- a validação é útil para prevenir erros, ou definir qual subconjunto da linguagem XML deve ser usado para a representação de um certo domínio
- no caso do DOliberto, podemos imaginar que usaremos algum tipo de validação para garantir que toda publicação do DO tem uma URN válida.

validação

- a validação é útil para prevenir erros, ou definir qual subconjunto da linguagem XML deve ser usado para a representação de um certo domínio
- no caso do DOliberto, podemos imaginar que usaremos algum tipo de validação para garantir que toda publicação do DO tem uma URN válida.
- num exemplo mais avançado, podemos garantir que toda publicação que versa sobre uma nomeação deve conter algumas tags fundamentais, como `<nomeado>`, `<autoridade>`, `<cargo>`, `<órgão>`, etc.

validação

- a validação é útil para prevenir erros, ou definir qual subconjunto da linguagem XML deve ser usado para a representação de um certo domínio
- no caso do DOliberto, podemos imaginar que usaremos algum tipo de validação para garantir que toda publicação do DO tem uma URN válida.
- num exemplo mais avançado, podemos garantir que toda publicação que versa sobre uma nomeação deve conter algumas tags fundamentais, como `<nomeado>`, `<autoridade>`, `<cargo>`, `<órgão>`, etc.
  - além disso, podemos garantir que um `<nomeado>` tem de ser uma `<pessoa>`, ou que um `<órgão>` tem de ser um órgão válido, etc.

dever de casa

- ler sobre DTDs (capítulo 3 do XML in a nutshell)

dever de casa

- ler sobre DTDs (capítulo 3 do XML in a nutshell)
  - mas o LexML usa XML Schema [docs] [arquivos]

dever de casa

- ler sobre DTDs (capítulo 3 do XML in a nutshell)
  - mas o LexML usa XML Schema [docs] [arquivos]
- qual usar para o DOliberto?

Conclusão

Perguntas?

bibliografia

- XML in a Nutshell (ISBN 978-0-596-00764-5)

bibliografia

- XML in a Nutshell (ISBN 978-0-596-00764-5)
- annotated XML specification

Contato

LABFGV

site, facebook, github.