Disciplina: Programação Cliente-servidor

Aula 5: XML e AJAX

Apresentação

Em termos gerais, os arquivos XML apresentam regras de escrita específicas que foram muito bem aceitas pelo mercado, e hoje trabalhamos com diversas tecnologias que fazem uso do formato.

Com a possibilidade de definição de gramáticas através de esquemas, o XML alia uma grande liberdade de criação ao formalismo necessário para as tarefas de configuração de servidores e comunicação empresarial no modelo B2B. Diversas tecnologias foram criadas com base no XML, e o formato acabou sendo adotado como padrão de interoperabilidade, o que é de grande relevância para o ambiente extremamente heterogêneo da Web.

Justamente por isso, o XML acabou sendo uma das bases para criação de páginas dinâmicas, com carregamento assíncrono através do AJAX (Http Assíncrono, JavaScript e XML). Logo, é necessário compreender este formato e as diversas tecnologias associadas a ele.

Objetivos

- Explicar as regras de escrita XML e tecnologias associadas;
- Explicar a utilização de AJAX para carregamento dinâmico;
- Usar o objeto DOMParser para tratamento de XML.

Fonte: Markus Spiske / Unsplash

O que é XML?

É muito importante compreendermos o que é o XML (Extended Markup Language), bem como sua utilização dentro do contexto tecnológico atual.

Incialmente, XML não é uma linguagem, mas um conjunto de regras de escrita, ou de maneira formal, recomendações da W3C para gerar linguagens de marcação voltadas para necessidades específicas. Embora não seja uma linguagem, a sintaxe XML foi utilizada para a criação de diversas linguagens, como XHTML, MathML, SVG, SMIL, entre várias outras.

O XML foi definido a partir da SGML (Standard Generalized Markup Language); seu objetivo primordial é facilitar o compartilhamento de informações na Internet. Por se tratar de um padrão em forma de texto plano, não é bloqueado por firewalls, além de permitir a interpretação a partir de qualquer ambiente de programação, como Java, dotNet, PHP, ou JavaScript, o que viabiliza grande nível de interoperabilidade.

> **A interoperabilidade pode ser entendida como uma característica que se refere à capacidade de diversos sistemas e organizações trabalharem em conjunto (interoperar) de modo a garantir que pessoas, organizações e sistemas computacionais interajam para trocar informações de maneira eficaz e eficiente.**
>
> Governo Digita

Através dos documentos XML, somos capazes de organizar a informação de uma forma hierárquica, agrupando elementos constituintes de um determinado fragmento de dados e suas respectivas dependências recursivamente.

Elementos Básicos

Ao criarmos um documento XML, devemos seguir algumas regras:

- Sempre deve existir um e apenas um nó raiz
- Toda etiqueta precisa ter fechamento
- Os comandos, ou etiquetas, devem estar corretamente aninhados
- Os atributos de cada etiqueta não podem apresentar repetições
- Ocorre a diferenciação entre elementos em maiúsculo ou minúsculo (case-sensitive)

Podemos ver, no exemplo seguinte, um documento XML contendo alguns dos elementos básicos necessários para a constituição deste tipo de arquivo.

Exemplo

```
<?xml version="1.0" encoding="iso-8859-1"?>
<turma codigo="1001">
  <professor>Tom Sawyer</professor>
  <alunos>
    <aluno matricula="1968001">Gleedy Lee</aluno>
    <aluno matricula="1968002">Alex Lifeson</aluno>
    <aluno matricula="1968003">Neil Peart</aluno>
  </alunos>
</turma>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

1° linha: presença de uma diretiva do XML, que define qual página de código será utilizada para acentuação, sendo **iso-8859-1** uma das opções para textos brasileiros.

```
<turma codigo="1001">
```

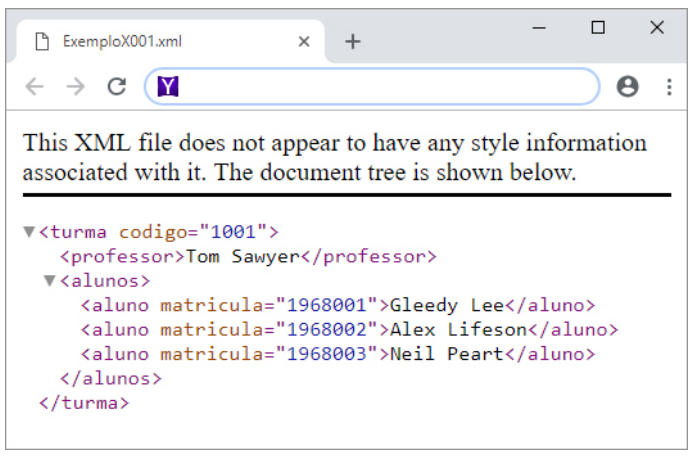
2° linha : nó inicial, que inclusive traz o atributo codigo. Note que mesmo com a acentuação correta, não é adequado utilizar acentos em nomes de elementos ou atributos.

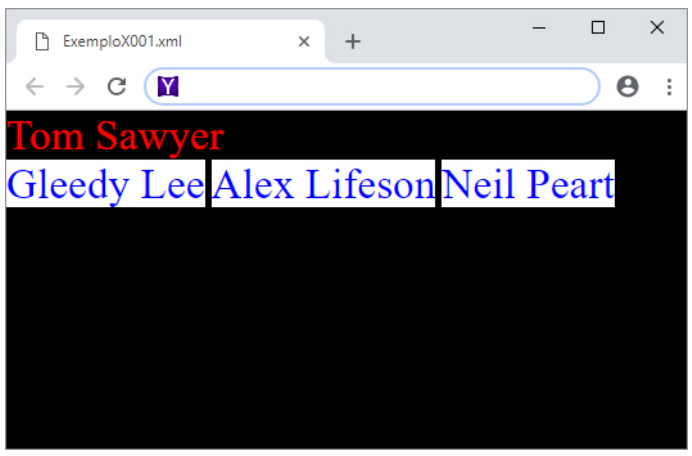
```
<alunos>
    <aluno matricula="1968001">Gleedy Lee</aluno>
```

Dentro de uma turma temos um professor e diversos alunos, cada aluno com um atributo de **matricula**.

Observe a mensagem que é apresentada pelo Chrome, onde diz que "o documento XML não parece ter nenhuma informação de estilo associada ao mesmo". Se tivermos uma folha de estilo associada ao XML, ele será apresentado de uma forma diferente da árvore padrão.

Todos estes dados acabam oferecendo uma estrutura hierárquica, onde a partir de um nó raiz temos a **turma**, constituída de **professor** e **alunos**, sendo este último elemento composto de diversas ocorrências do tipo **aluno**. Isso pode ser observado facilmente na saída proporcionada pelo **Chrome**, em formato de **árvore**.





Primeiro precisamos criar um arquivo CSS com nossas definições em termos de fontes, cores e formatações em geral. Como no HTML, aqui é possível utilizar as nossas tags diretamente.

Nós chamaremos este arquivo de "**turma.css**".

```
Turma    {background-color: #000000; width: 100%;}
professor {color: #FF0000; font-size: 20pt;}
alunos    {display: block; margin-bottom: 30pt; margin-left: 0;}
aluno    {background-color: #ffffff; border:1pt; color: #0000FF;
        font-size: 20pt;}
```

Note que são utilizadas propriedades comuns do CSS, como **color** e **background-color** para cores de fonte e fundo, bem como elementos de **margin** e **border** para configurar o distanciamento entre os elementos.

Depois de criado o arquivo com as configurações tipográficas, basta associar o XML ao mesmo, acrescentando a ligação através da diretiva **xml-stylesheet**.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/css" href="turma.css"?>
  <turma codigo="1001">
```

Podemos observar a versão final do XML a seguir.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/css" href="turma.css"?>
<turma codigo="1001">
  <professor>Tom Sawyer</professor>
  <alunos>
    <aluno matricula="1968001">Gleedy Lee</aluno>
    <aluno matricula="1968001">Gleedy Lee</aluno>
    <aluno matricula="1968001">Gleedy Lee</aluno>
  </alunos>
</turma>
```

Também é possível criarmos comentários no XML, de acordo com o mesmo modelo adotado pelo HTML, bem como implementarmos trechos livres de interpretação com o uso de CDATA [1].

O uso desses trechos CDATA é muito comum quando queremos acrescentar código em meio ao conteúdo de texto, como no exemplo seguinte.

Exemplo

```
<?xml version="1.0" encoding="utf8"?>
<comando>
  <!-- Estrutura de Decisão -->
  <nome>if</nome>
  <exemplo>
  <![CDATA[
  if(a<b)
    maior = b;
  else
    maior = a;
  ]]>
</exemplo>
</comando>
```

Sem o uso de CDATA nesse exemplo, ocorreria um erro, pois o parser entenderia que existe uma tag iniciada com b e que não foi fechada, isso devido à presença do if com a condição a<b.

Outro conceito muito importante para a criação de arquivos XML é o namespace.

Muitas vezes precisamos distinguir o significado de uma mesma palavra em contextos diferentes como, por exemplo, a palavra “ponto”, que pode significar um elemento em termos de coordenadas cartesianas ou o local de parada de ônibus, entre outros.

O uso de namespaces permitirá esta diferenciação, como podemos observar no exemplo seguinte.

Exemplo

```xml
<?xml version="1.0" encoding="utf8"?>
<colaborador xmlns:empresa="01.001.001/0001-01">
  <nome>Ana Maria</nome>
  <matricula uf="RJ">20180065-A</matricula>
  <empresa:nome>Padaria do João</empresa:nome>
</colaborador>
```

Nesse exemplo podemos observar a ocorrência da tag **<nome>**, referente ao nome do funcionário, e a tag **<empresa:nome>**, esta última se referindo à identificação da empresa.

A definição do namespace "empresa" ocorre logo no início do XML, com a associação do mesmo a uma URL de referência, a qual não tem grande significado neste caso, mas poderia ser um endereço real, com regras de escrita.

```xml
<colaborador xmlns:empresa="01.001.001/0001-01">
```

Esquemas

Muitas vezes escrevemos um XML sintaticamente correto, mas que não é válido para ser aplicado a determinado contexto.

Ao criarmos um aplicativo, esperamos receber tipos específicos de dados para tratamento; mas, com a liberdade de criação oferecida pelo XML, ocorre a possibilidade de o programa receber dados com as tags organizadas das mais diversas formas e com os mais diversos tipos de dados.

Exemplo

Por exemplo, alguém pode enviar a data como um campo texto único, enquanto outro usuário estaria fornecendo o mesmo dado como três campos numéricos, indicando o dia, o mês e o ano.

É necessário, portanto, definir um **esquema**, ou seja, uma gramática específica para o aplicativo em questão, envolvendo uma organização hierárquica dos dados e definição de tipos a serem utilizados.

Os dados podem ser verificados através de rotinas de programação, mas além de ser um processo bastante sujeito a falhas, não traz para o usuário do aplicativo qualquer informação acerca do formato que ele deverá utilizar para o envio da informação.

Logo, é mais interessante utilizarmos uma metodologia formal, com a validação através de **parsers**, os quais irão verificar de forma automática o XML frente a algum esquema.

Em termos de XML, os esquemas podem ser definidos com o uso de dois tipos de tecnologia:

Document Type Definition (DTD) [2]

W3C XML Schema [3]

Exemplo de DTD

```
<!DOCTYPE colaborador[
<!ELEMENT colaborador(nome, matricula)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT matricula (#PCDATA)>
<!ATTLIST matricula uf CDATA #REQUIRED>
]>
```

Note que são definidos elementos com o uso de **ELEMENT** e atributos com o uso de **ATTLIST**.

EXEMPLO de arquivo XSD

```
<xsd:schema xmlns:xsd="//www.w3.org/2001/XMLSchema">
<xsd:element name="colaborador">
 <xsd:complexType name="tipoColaborador">
  <xsd:sequence>
  <xsd:element name="nome" type="xsd:string"/>
  <xsd:element name="matricula" type="xsd:string">
    <xsd:complexType>
    <xsd:attribute name="uf" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Com o uso de XSD, os elementos são criados com tipos específicos, como **xsd:string**, existindo também outras opções, como tipos numéricos, por exemplo, e podemos definir nossos próprios tipos e nomeá-los.

```
<xsd:complexType name="tipoColaborador">
```

Note o uso de namespace na criação do arquivo XSD. Quem for utilizá-lo como definição de regra gramatical, também utilizará a chamada como namespace.

Comentário

Lembrando que é possível a combinação de diversos namespaces para um mesmo XML, o que permitirá a junção de diversas sintaxes dentro de um mesmo contexto.

Fonte: Sai Kiran Anagani / Unsplash

Transformação com XSLT

Hoje em dia é comum escutarmos o termo **“linguagem de transformação”**, mas o que isso realmente significa?

Como podemos observar, os documentos XML permitem a organização da informação, mas não estão preocupados com o aspecto visual desta informação, pois é papel do HTML, junto ao CSS, exibir os dados para o usuário.

Ocorre, portanto, a necessidade de um processo intermediário para a transformação dos dados XML para um formato HTML, passível de visualização em navegadores. É neste ponto que surgem linguagens específicas para este fim, as chamadas “linguagens de transformação”.

A linguagem de transformação mais utilizada para o ambiente que envolve XML é denominada eXtensible Stylesheet Language (XSLT). Esta é uma linguagem com sintaxe XML para a transformação de arquivos XML em outros formatos, como SVG, VRML, HTML, ou até outros arquivos XML.

Podemos observar, a seguir, um arquivo que será nomeado “**contatos.xsl**”.

Neste código podemos observar várias das características do XSLT, a começar pela utilização do namespace correto para o reconhecimento dos comandos da linguagem.

```
<xsl:stylesheet xmlns:xsl="//www.w3.org/1999/XSL/Transform" version="1.0">
```

A partir deste ponto podemos definir **templates** aplicáveis a determinadas tags do XML, ou à raiz do documento.

```
<xsl:template match="/">
```

Neste caso, ao encontrar a raiz, será iniciada a construção do HTML contendo uma tabela no corpo do documento, sendo criada uma linha para cada contato encontrado no documento XML com o uso de **xsl:for-each**.

```
<xsl:for-each select="contatos/contato">
```

Os valores a serem utilizados na saída são obtidos com o comando **xsl:value-of**. Este comando aceita apresentar o valor de texto de um elemento interno diretamente, de atributos com uso de **arroba**, ou o texto do elemento corrente com uso de **ponto**.

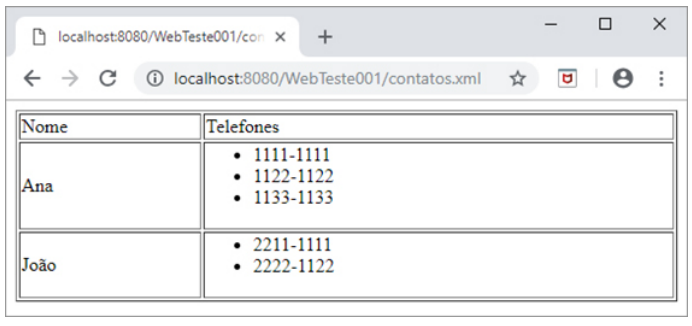
```
<xsl:value-of select="@nome"/>
```

```
<?xml version="1.0" encoding="utf8" ?>
<xsl:stylesheet xmlns:xsl="//www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<html><body>
<table border="1" width="100%">
<tr><td>Nome</td><td>Telefones</td></tr>
<xsl:for-each select="contatos/contato">
<tr><td><xsl:value-of select="@nome"/></td>
<td><ul>
<xsl:for-each select="telefone">
<li><xsl:value-of select="."/></li>
</xsl:for-each>
</ul></td></tr>
</xsl:for-each>
</table>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

Os elementos de navegação utilizados para **select** ou **match** fazem parte de um padrão de nomenclatura denominado **XPath**.

Vamos observar, a seguir, um exemplo de arquivo XML que utilizará o XSL anterior, contendo uma agenda de contatos e sua visualização no navegador.

```
<?xml version="1.0" encoding="utf8"?>
<?xml-stylesheet type="text/xsl" href="contatos.xsl"?>
<contatos>
  <contato nome="Ana">
    <telefone>1111-1111</telefone>
    <telefone>1122-1122</telefone>
    <telefone>1133-1133</telefone>
  </contato>
  <contato nome="João">
    <telefone>2211-1111</telefone>
    <telefone>2222-1122</telefone>
</contato>
</contatos>
```



Note que o arquivo XML referencia o XSL como uma folha de estilo.

```
<?xml-stylesheet type="text/xsl" href="contatos.xsl"?>
```

Este é um exemplo de formatação fixa, mas existe a possibilidade de utilizar o XSL sem efetuar a referência a partir do XML, desde que apoiado por bibliotecas de linguagens específicas, como Java e C#.

Como podemos observar, o arquivo XSL permite uma transformação muito grande da saída padrão do arquivo XML. Assim como é possível transformar para uma saída HTML, é possível transformar para qualquer formato texto, segundo o mesmo processo.

Atenção

Por questões de segurança, navegadores como o Chrome não permitem o acesso a arquivos locais a partir de outros arquivos locais; para abrir este exemplo deverá ser utilizada a opção de linha de comando **--allow-file-access-from-files**, ou criar um aplicativo Web e chamar o XML a partir de **localhost**.

Leitura com DOM

Com o uso de JavaScript é muito simples tratar os arquivos XML, pois o uso do **parser** segue os mesmos princípios de acesso aos elementos do HTML.

Vamos observar um pequeno exemplo em uma página HTML.

Exemplo

```
<!DOCTYPE html>
<html>
 <body>
  <p id="demo"></p>
  <script>
  var text, parser, xmlDoc;
  text = "<contato><nome>Ana</nome>"+
    "<telefone>1111-1111</telefone></contato>";
  parser = new DOMParser();
  xmlDoc = parser.parseFromString(text,"text/xml");
  var noInfo = xmlDoc.getElementsByTagName("nome")[0];
  document.getElementById("demo").innerHTML =
    noInfo.childNodes[0].nodeValue;
  </script>
 </body>
</html>
```

Neste exemplo temos o conteúdo XML em uma variável denominada **text**, contendo dados de contato como nome e telefone.

Em seguida instanciamos um objeto do tipo **DOMParser**. Este objeto será responsável pela criação da árvore DOM a partir do conteúdo XML.

```
parser = new DOMParser();
```

A árvore é montada em memória e colocada na variável **xmlDoc** com o uso do **parser**.

```
<xsl:for-each select="contatos/contato">
```

A partir daí é possível acessar os elementos do XML como qualquer árvore DOM, e com o uso de **getElementsByTagName** recebemos todas as tags com o nome especificado, sendo recebidas em um vetor com índice inicial **zero**.

```
var noInfo = xmlDoc.getElementsByTagName("nome")[0];
```

Porém, este nó será do tipo elemento; para acessar o texto, precisamos acessar o **primeiro filho** dele na árvore, correspondendo a um nó do tipo texto, com a propriedade **nodeValue** informando o valor deste texto.

```
noInfo.childNodes[0].nodeValue;
```

Após a obtenção dos dados, podemos exibi-los em qualquer camada ou parágrafo com o uso de **innerHTML**.

A única novidade real neste processo é o uso do parser, pois as alterações dinâmicas e o acesso aos componentes via DOM não diferem em nada de processos anteriores no ambiente HTML.

Neste exemplo o conteúdo XML está dentro do trecho em JavaScript, como variável de texto, mas normalmente este conteúdo será recebido via HTTP, o que nos levará ao estudo do **AJAX**.

Fonte: Luca Bravo / Unsplash

Tecnologia AJAX

O padrão de design de páginas atuais envolve a modificação dinâmica de partes do conteúdo, particularmente com a adoção de camadas e a possibilidade de alteração do conteúdo interno com o uso de **innerHTML**. Com estas funcionalidades é possível criarmos as diversas partes da página de forma dinâmica.

Aliado a isto, e seguindo o modelo inicialmente utilizado em portais, as informações podem ser modularizadas, sendo fornecidas por diferentes origens, o que nos permite dizer que as diversas camadas poderiam organizar a visualização do conteúdo, e cada camada pode estar sendo "alimentada" por uma fonte de dados distinta, normalmente em formato XML ou JSON.

Com isso já teríamos a possibilidade de criar uma página constituída de fragmentos de informação combinados, mas ainda podemos ir além...

Como as camadas tratam de informações de fontes independentes, podemos aproveitar o modelo de processamento viabilizado pela máquina para que as camadas sejam preenchidas em paralelo, diminuindo muito o tempo total de carga.

Este paralelismo é viabilizado através de um modelo **assíncrono** de comunicação, ou seja, um modelo no qual o servidor é requisitado, mas, ao contrário do padrão normalmente utilizado, em que o cliente interrompe o processamento e aguarda a resposta, neste, o cliente continua o seu processamento e deixa uma função **callback** esperando a resposta em paralelo.

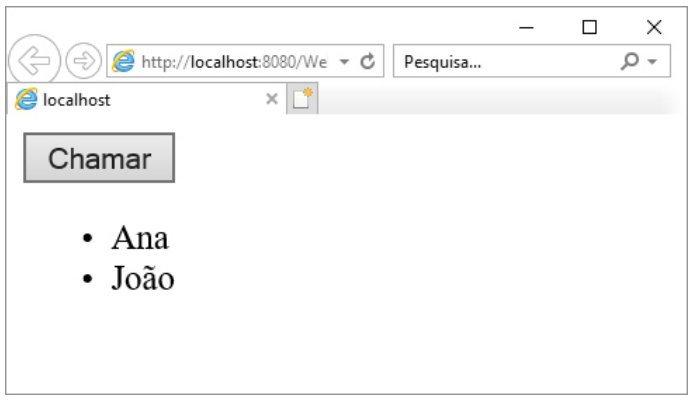
É justamente a partir destes princípios que surge o nome de uma "nova" tecnologia, na verdade a junção de tecnologias já existentes, o AJAX (Http Assíncrono, JavaScript e XML). Embora o XML faça parte da descrição, qualquer formato texto pode ser transmitido, como o JSON, por exemplo.

Podemos observar estes detalhes no exemplo seguinte, com a leitura assíncrona do arquivo XML de contatos criado anteriormente.

Exemplo

```
<html>
 <button onclick="atualizar()">Chamar</button>
 <div id="resposta">Resposta Aqui</div>
 <script>
 function atualizar(){
 var url = "contatos.xml";
 xhttp = new XMLHttpRequest();
 xhttp.open("GET", url, true);
 xhttp.onreadystatechange = AJAX_Callback;
 xhttp.send();
 }
 function AJAX_Callback(){
  if (xhttp.readyState == 4 && xhttp.status == 200) {
  var parser = new DOMParser();
  var xmlDoc =
  parser.parseFromString(xhttp.responseText,"text/xml");
  var nomes = xmlDoc.getElementsByTagName("contato");
  var saida = "<ul>";
  for(i=0;i<nomes.length;i++){
  saida += "<li>"+nomes[i].attributes[0].value+"</li>";
 }
  saida += "</ul>"
  document.getElementById("resposta").innerHTML =saida;
  }
  }
 </script>
  </body>
</html>
```

Neste exemplo, ao clicar sobre o botão, teremos a carga do nome dos contatos na camada, e todo o processo é iniciado ao chamar o método "**atualizar**".



O primeiro passo é a configuração da chamada através de um objeto do tipo **XmlHttpRequest**, que viabiliza a conexão HTTP de forma assíncrona (padrão) ou síncrona.

```
function atualizar(){
  var url = "contatos.xml";
  xhttp = new XMLHttpRequest();
  xhttp.open("GET", url, true);
  xhttp.onreadystatechange = AJAX_Callback;
  xhttp.send();
}
```

Segundo a configuração utilizada, temos uma chamada para o arquivo XML em modo GET, e a função de tratamento será "**AJAX_Callback**".

Comentário

Lembrando que teremos de executar uma chamada via HTTP para que nosso exemplo funcione, o que exigirá que as páginas estejam hospedadas, mesmo que em um servidor local.

Quanto à função de tratamento, ela deve inicialmente verificar se a carga dos dados já está completa, pois é chamada a cada mudança de estado da conexão. Lembrando sempre que esta função executa em paralelo, sem interferir na funcionalidade da página.

```
if (xhttp.readyState == 4 && xhttp.status == 200)
```

Os dados obtidos deverão ser carregados em um objeto do tipo **DOMParser**, permitindo o acesso aos elementos e atributos com o uso dos mesmos elementos de navegação DOM utilizados no HTML.

```
if (xhttp.readyState == 4 && xhttp.status == 200)
```

Da mesma forma que chamamos um arquivo XML neste exemplo, podemos efetuar a chamada de um programa qualquer no servidor que retorne os dados neste formato, ou outro formato texto, podendo utilizar tanto GET quanto POST.

(Fonte: Shutterstock)

Uso de XML na Web

O formato XML é amplamente utilizado na Web em comunicações entre empresas (B2B), principalmente devido ao formalismo proporcionado pelos esquemas **XSD** e **DTD**, o que garante a recepção de dados com formatos e tipos previamente estipulados.

Exemplo

Podemos observar diversas áreas no Brasil que utilizam este tipo de arquivo, como as notas fiscais eletrônicas, transações bancárias do tipo DOC e TED, entre diversos outros exemplos, sendo também um formato fundamental na integração de aplicativos, dentro de uma arquitetura orientada a serviços, com o uso de Web Services do tipo **SOAP**.

Isso justifica o nome dado à tecnologia **AJAX**, pois o padrão inicial de transmissão de dados era o formato XML, e embora seja utilizado JSON em diversas situações atuais, o uso de XML é muito mais adequado ao tratamento com uso de DOM.

Recentemente as áreas de comunicação com o cliente (B2C) começaram a utilizar amplamente os Web Services do tipo REST, com dados transitados em formato JSON, principalmente como efeito da grande expansão do mercado de dispositivos móveis. Porém, como o JSON não apresenta um formalismo adequado, os Web Services SOAP continuarão a ser o principal meio de integração na comunicação B2B.

Exemplo

Outro exemplo do uso de XML na Web, muito comum para todos nós, são os feeds de notícias com uso do formato **RSS**.

Além desta aplicação direta na transmissão de dados, existem diversas sintaxes baseadas em XML com ampla utilização na Web, como o **MathML** para expressar equações matemáticas, o formato **XMI** para representar diagramas UML e o **SVG** para desenho de gráficos vetoriais.

Vale lembrar também que a configuração de servidores de rede e de bancos de dados, em sua grande maioria, utiliza arquivos de configuração no formato XML.

Atividade

1. Em determinados momentos esperamos que uma aplicação receba dados XML com uma ordem e tipos específicos, necessitando a definição de uma gramática através de esquemas. Quais são as tecnologias utilizadas para a definição destes esquemas?

a) XSLT e SVG
b) DTD e XSD
c) MathML e RSS
d) SOAP e DOM
e) XMI e AJAX

2. Foi solicitado que você utilize a tecnologia AJAX para o carregamento dinâmico de dados em uma camada a partir de uma chamada a um Web Service SOAP. Como sabe que os dados deste tipo de componente são oferecidos no formato XML, quais serão os componentes necessários para a conexão com o servidor e a interpretação dos dados recebidos?

3. Considerando o XML apresentado a seguir, e um objeto DOMParser denominado parser apontando para o conteúdo, como seria o comando para obter a marca do primeiro carro?

```
<oficina>
  <carro>
    <marca>Volvo</marca>
    <placa>XPT2001</placa>
  </carro>
  <carro>
    <marca>Volvo</marca>
    <placa>XPT2001</placa>
  </carro>
</oficina>
```

a) parser.getElementById("marca")[0].value
b) parser.getElementsByTagName("carro")[0].childNodes[0].childNodes[0].nodeValue
c) parser.getElementById("marca")[0].nodeValue
d) parser.getElementsByTagName("carro")[0].childNodes[0].nodeValue
e) parser.getElementsByTagName("carro")[0].getElementById("marca").value

Notas

CDATA[1]

O uso desses trechos CDATA é muito comum quando queremos acrescentar código em meio ao conteúdo de texto, como no exemplo seguinte.

Document Type Definition (DTD)[2]

O modelo de gramática proporcionado pelo **DTD** é bastante simples, mas não utiliza XML em sua definição e não permite o uso de namespaces, o que significa dizer que não podemos ter gramáticas múltiplas com uso de DTD. Além dessas características, os esquemas do tipo DTD são muito limitados com relação a tipos de dados.

Na verdade, nós já utilizamos uma DTD ao colocar **<!DOCTYPE html>** no início de nossas páginas HTML.

W3C XML Schema[3]

Normalmente utiliza arquivos do tipo **XSD** (XML Schema Definition), e conta com sintaxe XML, aceitando namespaces e definição de tipos de dados.

Referências

DEITEL, P; DEITEL, H. **Ajax, rich internet applications e desenvolvimento web para programadores**. São Paulo: Pearson Education, 2009.

PLOTZE, R. **Tecnologias web.** Rio de Janeiro: Estácio, 2016.

SANTOS, F. **Tecnologias para internet II.** Vol. 1. Rio de Janeiro: Estácio, 2017.

Próxima aula

- O ambiente servidor Java;
- A tecnologia de Servlets;
- A tecnologia JSP.

Explore mais

Para conhecer mais sobre utilização de XSLT, acesse os seguintes tutoriais:

Para conhecer mais sobre utilização de XSLT, acesse os seguintes tutoriais:

- Tutorial XSLT <//www.dicas-l.com.br/arquivo/tutorial_xslt.php>
- XSLT – Transformação <https://www.w3schools.com/xml/xsl_transformation.asp>
- Transformando XML com XSLT <https://ciclosw.wordpress.com/2014/09/06/transformando-o-xml-com-xslt/>
- Analisador XML <https://www.w3schools.com/xml/xml_parser.asp>
- AJAX <https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest/send>