André Milani
Softblue

Ajax

e-book

Sumário

Apresentação

A Softblue é uma empresa de cursos online na área de programação. Fundada em 2003 na cidade de Curitiba-PR, a empresa conta atualmente com um grande portfólio de cursos e dezenas de milhares de alunos espalhados em dezenas de países pelo mundo.

Este e-book foi criado por André Milani, sócio fundador da Softblue. André Milani é formado em Ciência da Computação pela PUC-PR, pós-graduado em Business Intelligence pela mesma instituição e possui diversas certificações na área de TI. É também autor de vários livros na área de informática, entre eles o *Programando para iPhone e iPad*, *MySQL - Guia do Programador* e *Construindo Aplicações Web com PHP & MySQL*, todos pela editora Novatec. Atua desde 2003 com desenvolvimento web e treinamentos de profissionais. Também é desenvolvedor de aplicativos para o ambiente iOS da Apple, possuindo aplicações que juntas somam mais de 50.000 downloads na AppStore.

Disclosure

1. Introdução

O caminho da evolução da jornada de um desenvolvedor web certamente passa pelo AJAX mais cedo ou mais tarde. No começo, era somente o HTML, que permitia a construção de páginas para a internet, diagramando conteúdo quando combinado com o CSS, mas tendo ainda pouca interação entre a página e seus visitantes.

Para dar um pouco mais de vida às páginas web compostas até então por HTML, o JavaScript apareceu com uma proposta interessante: fornecer meios de executar uma linguagem de programação do tipo script dentro do HTML.

Certamente o JavaScript inovou muito com isso, mas ainda havia uma lacuna aberta, que era a necessidade de realizar uma atualização de página e/ou endereço no navegador para que os dados do navegador e do servidor pudessem ser trocados, mas sem a necessidade de carregar a página novamente por completo. Certamente isso deixaria o site muito mais dinâmico e aumentaria as interações com o usuário, pois poderiam ser trocadas informações entre o lado cliente e o servidor de uma forma muito mais prática e dinâmica.

Foi então que o JavaScript incorporou um conjunto de novos recursos com a intenção de permitir a comunicação entre o lado cliente e o lado servidor de forma assíncrona, ou seja, sem a necessidade de atualizar totalmente a página web aberta no navegador. Este conjunto de recursos disponibilizados pelo JavaScript é chamado de AJAX, sigla de *Asynchronous JavaScript and XML* ou, em português, *JavaScript e XML Assíncrono*. Esse conjunto de recursos permite lançar uma requisição HTTP ao servidor e tratar seu retorno via script, sem atualizar a página já aberta no navegador do usuário.

Pode-se dizer que estes comandos são realizados em background, ou por "detrás dos panos", se preferir, pois são transparentes para o visitante da página. Ele só saberá que algo está ocorrendo se o seu código JavaScript tiver ações programadas para informá-lo ou atualizar a página via programação de alguma forma.

Este livro irá guiá-lo no aprendizado da tecnologia AJAX e, para isso, assume que você já conhece e domina a linguagem HTML e a linguagem JavaScript, pois ambas são utilizadas no decorrer do processo de uma transação feita em AJAX.

Ela pode parecer complicada no começo. No entanto, ela é bastante simples, pois basicamente é composta por comandos JavaScript que irão permitir lançar uma requisição HTTP e tratar o seu retorno.

Antes de começarmos a construir nossas transações AJAX, vamos conhecer um pouco a mais sobre os elementos que compõem esta tecnologia.

2. A classe XMLHttpRequest

A classe `XMLHttpRequest` é a responsável por permitir a comunicação assíncrona entre os lados cliente e servidor. Foi padronizada e incorporada oficialmente no JavaScript pelo W3C (órgão que padroniza as tecnologias da internet) a partir do ano de 2006. Por meio desta classe, é possível iniciar uma transação assíncrona, tratar o seu retorno, enviar e receber informações e ainda tratar diversas etapas pelas quais o ciclo da transação pode passar.

Existem algumas diferenças na criação de objetos desta classe, dependendo do navegador que você estiver utilizando. Como você já deve saber, a implementação do núcleo do JavaScript em cada navegador é de responsabilidade do fornecedor de cada navegador. Em alguns deles, a classe `XMLHttpRequest` é chamada de `XMLHTTP` e pode ser acessada por meio do `ActiveXObject` da Microsoft, da seguinte forma:

```
new ActiveXObject('Microsoft.XMLHTTP');
```

É cada vez menos comum ver a inicialização do AJAX conforme mencionado anteriormente, pois este processo está cada vez mais padronizado para ser realizado pelos principais navegadores da seguinte forma:

```
new XMLHttpRequest();
```

Mesmo o exemplo apresentado por último sendo o mais utilizado, ainda é importante tratar as duas formas nos códigos JavaScript. Pode ficar tranquilo, pois neste livro você aprenderá a construir um bloco de código que se adaptará ao navegador do usuário, utilizando o mecanismo mais apropriado para a construção do objeto em questão dependendo de cada caso.

Nos próximos tópicos deste livro você conhecerá os principais elementos envolvidos em uma transação AJAX, sendo boa parte deles propriedades e métodos da classe `XMLHttpRequest`. Não se preocupe neste momento em compreendê-los completamente. O objetivo inicial será apresentar os elementos e seus conceitos, para depois construirmos nossos primeiros exemplos de AJAX.

Após ter o objeto `XMLHttpRequest` criado e disponibilizado para utilização no JavaScript, é necessário informar qual modo a requisição assíncrona deve utilizar para trocar dados com o lado servidor, e também qual URL de destino será acessada para que a transação seja executada. Essas informações devem ser informadas diretamente por meio do método `open` desta classe, que recebe dois parâmetros, como é exibido a seguir:

```
ajax.open(<modo de abertura>, <url>);
```

O método `open()`, como foi apresentado, utiliza apenas dois parâmetros na chamada. O primeiro deles é o modo em que a requisição HTTP deve ser construída, que pode ser *GET* ou *POST*.

Relembrando um pouco dos conhecimentos abordados no e-book de HTML da Softblue, o método *GET* é aquele cujos valores e variáveis são informados junto com o endereço de destino da requisição (por exemplo: `minhaPagina.htm?nome=andre`). Já o método *POST* é aquele que empacota as variáveis separando-as do endereço de destino, sendo muito mais seguro quando utilizado em conjunto com o protocolo *HTTPS*.

O segundo parâmetro do método `open()` é a URL de destino da requisição que será lançada via AJAX. O método `open()` apenas define e configura a transação, mas não inicia a mesma ainda, pois, antes disso, é necessário informar qual função JavaScript tratará as mudanças de fases e conclusão da transação. Isto se faz necessário porque o AJAX não apenas busca uma informação qualquer, mas também informa a respeito de todas as etapas que são transcorridas no processo.

Para informar a função customizada que tratará tudo isso, é necessário configurar a propriedade `onreadystatechange` do objeto `XMLHttpRequest`. Essa configuração é bem simples de ser realizada e, por este motivo, veremos a mesma diretamente nos exemplos abordados logo mais adiante neste e-book.

Ao todo, são cinco diferentes etapas que uma transação AJAX passa, começando com a sua construção e passando pelo envio e carregamento dos dados de resposta. São elas:

- **0 (*UNSENT*)**: o objeto foi criado mas ainda não foi configurado (`open()` ainda precisa ser chamado).
- **1 (*OPENED*)**: o objeto já foi configurado, mas ainda não foi lançado (`send()` ainda precisa ser chamado).
- **2 (*HEADERS_RECEIVED*)**: a transação foi lançada e recebe o cabeçalho de resposta da requisição.
- **3 (*LOADING*)**: os dados (além do cabeçalho) da resposta começam a ser recebidos, mas ainda não estão completos.
- **4 (*DONE*)**: os dados da resposta são recebidos por completo.

Observe que na listagem das etapas do processo de uma transação AJAX é informado um código numérico para cada uma delas. Este valor é o código de cada fase, que poderá ser acessado no código JavaScript para podemos saber exatamente em qual etapa a transação se encontra a cada momento. A propriedade do objeto `XMLHttpRequest` que permite acessar o número da etapa corrente é a `readyState`, que também será abordada logo mais a frente, nos primeiros exemplos deste e-book.

Por meio das diferentes fases do AJAX, é possível exibir diferentes informações na tela do usuário, como por exemplo, mensagens de *solicitando dados* ou então *recebendo dados*. Contudo, o uso mais popular é apresentar um arquivo GIF animado para indicar que dados estão sendo trocados com o servidor e tratar somente a fase quatro (`4`) do processo, que é a que disponibiliza os dados para utilização pelo cliente.

Dos elementos principais do objeto `XMLHttpRequest`, restam ainda três para que possamos construir nosso primeiro exemplo. Um deles é a propriedade `status`, que permite acessar a situação da resposta da requisição por parte do servidor. Em outras palavras, quando uma transação AJAX chega a sua etapa final (`4`), ela pode apresentar diferentes situações respondidas pelo servidor, todas definidas pelo protocolo HTTP, sendo as principais delas:

- **200 (*OK*)**: indica que a transação obteve a resposta do servidor com sucesso.
- **400 (*BAD REQUEST*)**: indica que a construção da requisição é inválida.
- **401 (*UNAUTHORIZED*)**: indica falta de permissão para acessar a URL.
- **403 (*FORBIDDEN*)**: indica acesso proibido ao conteúdo no servidor.
- **408 (*REQUEST TIMEOUT*)**: indica que o servidor não respondeu em tempo hábil.

Como é possível notar na listagem das principais situações de retorno da resposta por parte do servidor, é o código `200` que desejamos encontrar para que possamos tratar os dados no JavaScript. Portanto, na função que irá avaliar as diferentes fases e situações do processo, é bastante comum concluir a transação AJAX somente quando a etapa for `4` e sua situação for `200`.

As duas últimas propriedades deste conjunto inicial de elementos são justamente as propriedades que permitem acessar os dados retornados pelo lado servidor. Caso a requisição tenha feito uma busca por conteúdo de texto, é a propriedade `responseText` que apresentará o resultado. Já para transações AJAX que troquem informações via XML, é a propriedade `responseXML` que você precisará utilizar.

Pronto! Neste momento posso afirmar que você já conhece os principais conceitos e elementos por trás das transações AJAX. A partir deste ponto, iremos colocar em prática estes conceitos, construindo exemplos práticos deste recurso, para que outras questões possam ser abordadas e esclarecidas. Portanto, pegue o seu cafezinho e vamos juntos para o próximo tópico!

3. Minha primeira transação AJAX

Nosso primeiro exemplo de transação AJAX será construído com a intenção de buscar um conteúdo qualquer no lado servidor e apresentar para o usuário, sem a necessidade de recarregar a página web, obviamente.

Uma consideração muito importante que deve ser feita neste momento é que o AJAX funciona exclusivamente por meio do uso do protocolo *HTTP* e/ou *HTTPS*. Ou seja, a transação AJAX não funcionará se você salvar os arquivos HTML em seu computador e acessá-los por meio do gerenciador de arquivos (*Windows Explorer*, por exemplo) e solicitar que sejam executados no navegador ou simplesmente abrir o navegador e solicitar a abertura do arquivo sem informar sua URL de acesso com o protocolo *HTTP*/*HTPS*. Quando um arquivo é aberto dessa forma, é o protocolo `file://` que assume o controle da operação, e este protocolo não funciona com AJAX. Portanto, é obrigatório o uso de um servidor web como o Apache, Tomcat ou outro para que os arquivos sejam acessados devidamente com o protocolo `http://` via navegador web.

Inicialmente, considere o arquivo de texto que ficará disponibilizado no servidor com o conteúdo que será capturado via AJAX posteriormente:

```
E-book de AJAX é na Softblue!
```

Código-Fonte 3-1. conteudoSimples.txt

O arquivo `conteudoSimples.txt` contém apenas uma frase, que servirá para a construção do nosso primeiro exemplo de transação AJAX. A única consideração mais importante neste momento é que o arquivo texto em questão seja salvo utilizando o formato *UTF-8* do seu editor de texto, para garantir o bom funcionamento dos caracteres especiais posteriormente. Em boa parte dos casos você pode utilizar o seu próprio editor HTML para salvar o arquivo, onde você pode definir o formato dos dados.

Considere também o arquivo HTML que conterá toda a marcação da página e também apresentará os recursos necessários do JavaScript para que a transação funcione adequadamente. A estrutura inicial deste arquivo deverá conter as marcações HTML necessárias para uma página, como mostra o exemplo a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>Softblue: E-book de AJAX</title>
06     <script>
07     </script>
```

```
08   </head>
09   <body>
10     <h1>Exemplo de captura de texto via AJAX</h1>
11
12     <input type="button" onClick="ajaxStart();"
13            value="Obter informação de texto via AJAX"><br><br>
14
15     <div id="meuDiv"></div><br>
16
17     <img src="imageEmpty.gif" id="imagemStatus">
18   </body>
19 </html>
```

Código-Fonte 3-2. simplesTexto.htm (versão parcial)

O arquivo `simplesTexto.htm` apresenta uma estrutura inicial de uma página HTML simples, com as marcações que você certamente já conhece. Mas vamos revisá-las rapidamente.

Na primeira linha temos a definição de um arquivo do tipo HTML, e nas linhas 2 e 19 a abertura e fechamento da marcação `<html>`, já configurada com o atributo `lang="pt-br"` informando o idioma utilizado pela página.

Entre as linhas 3 e 8, temos o cabeçalho do arquivo HTML, onde várias informações são preenchidas, tais como o título da página e a importante marcação `<meta>` com a propriedade `charset="utf8"`, informando que o conteúdo do arquivo utiliza este formato de codificação.

Vale a pena reforçar ainda que o arquivo em si deve ser salvo utilizando também o formato *UTF-8* no editor HTML. Caso contrário, os caracteres especiais (como acentuação) poderão aparecer com problemas no navegador.

Ainda no cabeçalho HTML, as linhas 7 e 8 abrem e fecham a marcação `<script>`, onde posteriormente os códigos JavaScript serão adicionados.

O corpo em si do conteúdo HTML deve ser definido entre a abertura e o fechamento da marcação `<body>`, feitos respectivamente nas linhas 9 e 18. Dentro do corpo temos um título da página na linha 10 feito por uma marcação `<h1>`; um botão construído por meio da marcação `<input>`, que inicia na linha 12 e termina na linha 13 apenas para organizar melhor suas propriedades; um `<div>` na linha 15; e uma imagem na linha 17.

Alguns detalhes devem ser observados com muita atenção no código HTML apresentado até o momento. O primeiro deles é que o botão inserido na linha 12 possui o atributo `onClick`, configurado para disparar uma função JavaScript chamada `ajaxStart()` quando for clicado pelo usuário. Essa função não foi construída ainda, mas ela será apresentada em breve.

O outro detalhe é que tanto o painel `<div>` apresentado na linha 15 quanto a imagem construída na linha 17 tiveram suas propriedades `id` definidas com os nomes `meuDiv` e `imagemStatus`, respectivamente. Esses atributos foram definidos porque ambos os painéis serão acessados via JavaScript neste exemplo.

A imagem `imageEmpty.gif` utilizada neste exemplo é uma imagem vazia. Isso mesmo, ela não apresenta nenhum conteúdo no momento. Ela será trocada por uma imagem GIF com animação, que representa carregamento de dados quando o AJAX estiver executando. Por este motivo, é importante que ela tenha um `id` definido, a fim de que possa ser acessada via JavaScript posteriormente. Já o painel `<div>` também precisa ter o `id` definido, pois é nele que escreveremos o conteúdo capturado via AJAX.

Agora que a parte HTML acabou de ser revisada, vamos ao principal conteúdo deste e-book: as funções JavaScript que gerenciam uma transação AJAX. São duas funções que precisam ser adicionadas no nosso exemplo: uma para configurar e lançar a transação, e outra para capturar o retorno e tratá-lo.

3.1. Lançando a requisição AJAX

Para fazer com que o botão apresentado no exemplo funcione com sucesso, é necessário criar a função que ele invoca, chamada de `startAjax()`. Por este motivo, dentro da marcação `<script>`, inclua o seguinte código-fonte:

```
01 function ajaxStart()
02 {
03   var imagem = document.getElementById("imagemStatus");
04   imagem.src = "imageLoading.gif";
05
06   var url = "conteudoSimplesTexto.txt";
07
08   if(window.XMLHttpRequest) // Maioria dos navegadores
09   {
10     req = new XMLHttpRequest();
11     req.onreadystatechange = ajaxProcessarRecebimento;
12     req.open("GET", url);
13     req.send(null);
14   }
15   else if(window.ActiveXObject) // IE ou outros Windows ActiveX
16  {
17    req = new ActiveXObject("Microsoft.XMLHTTP");
18    req.onreadystatechange = ajaxProcessarRecebimento;
19    req.open("GET", url);
20    req.send();
21   }
22 }
```

O corpo da função `startAjax()` é definido entre as linhas 2 e 22 do código-fonte apresentado. Inicialmente, esta função executará uma ação para informar visualmente o usuário que uma informação está sendo carregada, alterando a imagem existente no código HTML para apresentar a figura `imageLoading.gif`, por meio das operações

existentes nas linhas 2 e 3, que capturam a referência da imagem por meio do `id`. Em seguida, alteram a propriedade `src`. Estes temas foram abordados no e-book de JavaScript da Softblue. A imagem em questão (`imageLoading.gif`) pode ter outro nome e ser a imagem que você preferir. Neste exemplo é um GIF animado circular, que deve ser disponibilizado no mesmo endereço da página. Fique tranquilo, você pode baixar as imagens e os códigos-fonte deste e-book no site da Softblue.

Em seguida, na linha 6, temos a criação de uma variável chamada `url`, que contém o endereço do arquivo com o conteúdo que o AJAX deve acessar. Este endereço é relativo ao da página acessada, portanto o arquivo deve ser publicado na mesma pasta. É possível definir o caminho completo do arquivo se você preferir, algo como `http://localhost/conteudoSimplesText.txt`, ou o caminho que você estiver utilizando.

E, finalmente, é a partir da linha 8 que o AJAX realmente entra em cena. O comando apresentado nesta linha irá verificar se o navegador do usuário possui o objeto `XMLHttpRequest` para a construção da transação AJAX. Em caso positivo, são os códigos entre as linhas 10 e 13 que irão construir e configurar a transação, e nós já falaremos deles. Caso este objeto não exista, o navegador provavelmente possui outra implementação do JavaScript para esta funcionalidade, e neste caso a transação deve ser criada e configurada conforme o bloco de código apresentado entre as linhas 17 e 20.

Ambos os códigos apresentados entre as linhas 10-13 e 17-20 realizam as mesmas ações, mas de formas diferentes. Inicialmente, eles obtêm um objeto que permite a construção de uma transação AJAX e depois definem a propriedade `onreadystatechange`, que deve ser populada com o nome de uma função JavaScript customizada para ser chamada a cada troca de etapa na transação. Observe que a função em questão (`ajaxProcessarRecebimento`) ainda não foi construída. Ela será a última a ser abordada na montagem deste exemplo. Vale a pena reforçar ainda que o nome da função customizada deve ser informado sem os parenteses no final `()`, como mostram as linhas 11 e 18.

Nas últimas duas linhas de cada bloco de criação de transação AJAX, a mesma é criada por meio do método `open` e disparada por meio do método `send`. Em ambos os casos, o método escolhido neste exemplo foi o *GET* e a URL de destino foi o conteúdo da variável `url`, criada e preenchida anteriormente na linha 6.

Até aqui está tudo certo com a transação AJAX disparada. Mas o código está incompleto, sendo necessário construir ainda a função que irá fazer o tratamento das mudanças de etapas do processo. O nome desta função é `ajaxProcessarRecebimento`, conforme configurado na propriedade `onreadystatechange`. Para isso, vamos criar essa função e tratar o retorno da transação AJAX.

3.2. Tratando o retorno da transação AJAX

Para completar o código-fonte do primeiro exemplo de transação AJAX, inclua também dentro da marcação `<script>` do seu arquivo HTML o seguinte código-fonte:

```
01 function ajaxProcessarRecebimento()
02 {
03   if(req.readyState == 4)
04   {
05     if(req.status == 200)
06     {
07       document.getElementById("meuDiv").innerHTML = req.responseText;
08
09       var imagem = document.getElementById("imagemStatus");
10       imagem.src = "imageEmpty.gif";
11     }
12     else
13     {
14       alert(req.status + ": Possível erro é a falta do protocolo HTTP.");
15     }
16   }
17 }
```

DICA: Se o código-fonte digitado por você não funcionar, você pode baixar no site da Softblue os códigos-fonte corretos apresentados neste livro. Isso facilita bastante a procura por eventuais erros de digitação, causa mais comum de problemas que impedem a execução dos códigos.

A função `ajaxProcessarRecebimento()` é bastante simples. Ela será invocada pelo AJAX a cada mudança de etapa do processo, ou seja, ela provavelmente será executada várias vezes, nas mudanças da etapa `1` para `2`, da etapa `2` para `3` e `3` para `4`.

Como neste exemplo assume-se que não interessa ao visitante ser informado sobre a troca das etapas, somente a etapa `4` é que irá executar alguma ação. Portanto o comando `if` apresentado na linha 3 valida isso. Se a chamada em questão apresentar no objeto da requisição (`req`) a propriedade `readyState` com o valor 4, sabe-se que é a etapa final do AJAX, cujos dados provavelmente já foram baixados e estão prontos para serem utilizados.

Aqui uma dúvida é bastante comum entre os iniciantes em AJAX: *"o que é a variável req e de onde ela surgiu?"*. Essa variável é disponibilizada pelo JavaScript e representa a requisição AJAX que está em andamento. Assim como objetos do tipo `window` ou o próprio `document`, que são disponibilizados pelo JavaScript, a requisição neste caso poderá ser acessada por meio da variável `req`, que contém um objeto representando a transação.

Ainda, conforme dito anteriormente, se a etapa for `4`, provavelmente os dados estarão disponíveis, mas isso ainda não é garantido, pois é necessário verificar a

situação da etapa, a fim de saber se ela foi atendida com sucesso pelo servidor ou se algum problema ocorreu.

Para isso, é necessário validar o atributo `status` do objeto contido em `req` para saber se ele apresenta o valor `200`, já apresentado anteriormente neste livro, indicando o sucesso da operação. Qualquer valor diferente deste indica que um problema foi detectado e que os dados podem não estar disponíveis. Essa validação é realizada por meio do comando `if` apresentado na linha 5.

Se tudo der certo, é o bloco de código entre as linhas 6 e 11 que será executado. Este bloco acessa o conteúdo obtido via AJAX por meio da propriedade `responseText` do objeto contido em `req` e o atribui ao painel `meuDiv` na tela, logo abaixo do botão. Tudo isso é realizado na linha 7.

Ainda, as linhas 9 e 10 trocam novamente a imagem da página, para que a animação GIF de carregamento de dados saia de cena e retorne a imagem vazia (`imageEmpty.gif`), indicando o final da operação. O resultado final desta operação pode ser conferido na imagem a seguir.

Figura 3-1. Resultado de captura de conteúdo via AJAX

Contudo, se a etapa `4` não retornar o valor `200` para o status da operação, o bloco `else` existente entre as linhas 13 e 15 é executado, apresentando na tela por meio de um comando `alert` informações gerais para o usuário. Neste caso, optou-se por apresentar o código do status de retorno da requisição e a frase `Possível erro é a falta do protocolo HTTP`.

Vale a pena lembrar e reforçar: se você digitou o código ou baixou no site da Softblue e abriu o arquivo no navegador sem utilizar um servidor web real para publicá-lo e acessá-lo com um endereço válido de protocolo HTTP, é muito provável que o exemplo não funcione e apresente essa mensagem a você.

O exemplo executa muito rapidamente, ainda mais se você tiver feito a publicação em um servidor local (seu próprio computador). O processo é tão rápido que,

possivelmente, você não conseguiu nem mesmo ver o GIF animado que representa o carregamento dos dados, certo?

Caso você queira ver o ícone, remova ou comente a linha de código 10 da função `ajaxProcessarRecebimento()` e execute o exemplo novamente. Isso fará com que a imagem não volte ao estado inicial quando a transação AJAX encerrar, e você poderá ver o ícone por mais tempo. Mas não se esqueça de reativar esta linha depois.

O código completo deste exemplo pode ser visualizado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>Softblue: E-book de AJAX</title>
06     <script>
07       function ajaxStart()
08       {
09         var imagem = document.getElementById("imagemStatus");
10         imagem.src = "imageLoading.gif";
11
12         var url = "conteudoSimplesTexto.txt";
13
14         if(window.XMLHttpRequest) // Maioria dos navegadores
15         {
16           req = new XMLHttpRequest();
17           req.onreadystatechange = ajaxProcessarRecebimento;
18           req.open("GET", url);
19           req.send(null);
20         }
21         else if(window.ActiveXObject) // IE ou outros Windows ActiveX
22         {
23           req = new ActiveXObject("Microsoft.XMLHTTP");
24           req.onreadystatechange = ajaxProcessarRecebimento;
25           req.open("GET", url);
26           req.send();
27         }
28       }
29
30       function ajaxProcessarRecebimento()
31       {
32         if(req.readyState == 4)
33         {
34           if(req.status == 200)
35           {
36             document.getElementById("meuDiv").innerHTML = req.responseText;
37
38             var imagem = document.getElementById("imagemStatus");
39             imagem.src = "imageEmpty.gif";
40           }
41           else
42           {
43             alert(req.status + ": Possível erro é a falta do protocolo HTTP.");
44           }
45         }
46       }
47     </script>
48   </head>
```

```
49   <body>
50     <h1>Exemplo de captura de texto via AJAX</h1>
51
52     <input type="button" onClick="ajaxStart();"
53            value="Obter informação de texto via AJAX"><br><br>
54
55     <div id="meuDiv"></div><br>
56
57     <img src="imageEmpty.gif" id="imagemStatus">
58   </body>
59 </html>
```

Código-Fonte 3-3. simplesTexto.htm

Uma pergunta bastante recorrente neste momento é: *"mas o conteúdo a ser capturado é sempre o mesmo?"*. Não necessariamente. Pode ser que o seu site, sua aplicação, atualize de tempos em tempos o arquivo texto com conteúdos mais relevantes, ou ainda você pode fazer o seu código AJAX apontar diretamente para um arquivo PHP ou um arquivo JAVA no servidor, para que ele processe as informações e retorne um conteúdo customizado para seu usuário. A partir deste ponto, a sua criatividade é o limite.

Tranquilo, não é? Como você pôde perceber é bastante simples utilizar AJAX em seus projetos. Os próximos exemplos deste livro serão muito parecidos com este. A diferença estará basicamente na função que trata o retorno da operação, onde iremos aprender a tratar arquivos XML simples e também listas de elementos.

4. Transação AJAX com XML

O exemplo anteriormente abordado neste livro capturava, via AJAX, um conteúdo em formato de texto simples. Esse tipo de utilização é bastante comum e popular, mas existe outro que vale a pena ser citado, que é o conteúdo em formato XML.

DICA: Se você não está familiarizado com XML, vale a pena conhecer um pouco sobre este tema acessando este endereço: https://pt.wikipedia.org/wiki/XML

Os procedimentos da transação AJAX não mudam muito para deixar de capturar um conteúdo em texto simples e capturar um conteúdo em XML. Basicamente, é a função de tratamento de retorno que precisa ser adaptada para navegar adequadamente entre os elementos do arquivo XML.

Vamos ver neste tópico um exemplo de transação AJAX para capturar um conteúdo em formato XML. Para isso, considere o arquivo XML apresentado a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOFTBLUE>
  <TITLE codigoLivro="28">AJAX com André Milani</TITLE>
  <COUNTRY>Brasil</COUNTRY>
  <COMPANY>Softblue</COMPANY>
</SOFTBLUE>
```

Código-Fonte 4-1. conteudoSimplesXml.xml

O arquivo `conteudoSimplesXml.xml` apresenta um arquivo XML simples, com algumas informações fictícias, como por exemplo, um elemento raiz chamado `SOFTBLUE` e três elementos internos: `TITLE`, `COUNTRY` e `COMPANY`, apresentando informações de exemplo. Observe ainda que o elemento `TITLE` possui um atributo chamado `codigoLivro`, além do seu conteúdo apresentado posteriormente.

Assim como em outros momentos neste livro, vale a pena lembrar e reforçar que o arquivo XML em questão deve apresentar o atributo `encoding="UTF-8"` na primeira linha, e também deve ser salvo pelo editor de conteúdo utilizando o formato *UTF-8*, a fim de que os caracteres especiais funcionem adequadamente.

As mudanças que precisam ser realizadas no arquivo HTML são basicamente informar a nova URL para a transação AJAX acessar o novo arquivo e a adaptação da função de tratamento de retorno para manipular o conteúdo XML. Para isso, vamos primeiro visualizar o código-fonte completo e logo em seguida realizar os esclarecimentos necessários:

```
01 <!DOCTYPE html>
```

```
<html lang="pt-br">
  <head>
    <meta charset="utf-8">
    <title>Softblue: E-book de AJAX</title>
    <script>
      function ajaxStart()
      {
        var imagem = document.getElementById("imagemStatus");
        imagem.src = "imageLoading.gif";

        var url = "conteudoSimplesXml.xml";

        if(window.XMLHttpRequest) // Maioria dos navegadores
        {
          req = new XMLHttpRequest();
          req.onreadystatechange = ajaxProcessarRecebimento;
          req.open("GET", url);
          req.send(null);
        }
        else if(window.ActiveXObject) // Navegadores ActiveX
        {
          req = new ActiveXObject("Microsoft.XMLHTTP");
          req.onreadystatechange = ajaxProcessarRecebimento;
          req.open("GET", url);
          req.send();
        }
      }

      function ajaxProcessarRecebimento()
      {
        if(req.readyState == 4)
        {
          if(req.status == 200)
          {
            var xmlDoc = req.responseXML;

            var eSoftblue = xmlDoc.getElementsByTagName("SOFTBLUE")[0];
            var eTitle = eSoftblue.getElementsByTagName("TITLE")[0];
            var conteudoTitle = eTitle.firstChild.nodeValue;
            var buffer = "Título do livro: " + conteudoTitle + " ";

            var eAtributo = eTitle.getAttribute('codigoLivro');
            buffer += "Código do livro: " + eAtributo + " ";

            var eCountry = eSoftblue.getElementsByTagName("COUNTRY")[0];
            var conteudoCountry = eCountry.firstChild.nodeValue;
            buffer += "País: " + conteudoCountry + " ";

            var eCompany = eSoftblue.getElementsByTagName("COMPANY")[0];
            var conteudoCompany = eCompany.firstChild.nodeValue;
            buffer += "Empresa: " + conteudoCompany + " ";

            alert(buffer);

            var imagem = document.getElementById("imagemStatus");
            imagem.src = "imageEmpty.gif";
          }
          else
          {
            alert(req.status + ": Possível erro é a falta do protocolo HTTP.");
          }
        }
```

```
64       }
65     </script>
66   </head>
67   <body>
68     <h1>Exemplo de captura de XML via AJAX</h1>
69
70     <input type="button" onClick="ajaxStart();"
71            value="Obter informação de XML via AJAX"><br>
72
74     <img src="imageEmpty.gif" id="imagemStatus">
75   </body>
76 </html>
```

Código-Fonte 4-2. simplesXml.htm

A primeira diferença entre o arquivo `simplesXml.htm` e o exemplo anterior abordado neste e-book ocorre na linha 12, onde a URL do arquivo que será acessado deve ser atualizada para `conteudoSimplesXml.xml`. A segunda diferença é que, neste exemplo, não será criado um painel `<div>` no HTML, pois exibiremos o resultado por meio de um comando `alert` do JavaScript. Todas as demais alterações ocorrem diretamente na função de tratamento de retorno, ou seja, na função `ajaxProcessarRecebimento()`, entre as linhas 31 e 64.

O tratamento do retorno de arquivo XML inicia com as mesmas validações de etapa e situação da resposta nas linhas 32 e 34, validando se `req.readyState` apresenta o valor `4` (representando que a transação encerrou), e se `req.status` apresenta o valor `200` (que representa o sucesso da captura dos dados). A partir daí o conteúdo XML pode ser acessado por meio da propriedade `responseXML` da requisição, como ocorre na linha 36, onde o conteúdo é atribuído a uma variável JavaScript chamada `xmlDoc`, para ser processado. A partir daí, ocorre a navegação nos elementos do XML.

4.1. Capturando o elemento raiz

O primeiro passo ao analisar um conteúdo XML é capturar o seu elemento raiz, que contém todas as informações do documento. O objeto acessado via `responseXML` apresenta métodos para essa operação, entre eles o método `getElementsByTagName()`, que retorna um array com todos os elementos encontrados para o nome especificado no seu parâmetro, como ocorre na linha 38.

Observe ainda na linha 38 que, para capturar o elemento raiz (`SOFTBLUE`), a variável `xmlDoc` é utilizada para acessar o objeto que contém o conteúdo XML, e, por meio do método `getElementsByTagName("SOFTBLUE")`, todos os elementos de nome `SOFTBLUE` serão retornados e organizados em um array. Como sabemos que há somente um elemento com este nome em nosso arquivo de exemplo, podemos diretamente no final da chamada deste método incluir o índice `[0]`, que acessará a primeira posição do array retornado, vinculando o elemento `SOFTBLUE` em questão à variável `eSoftblue`.

4.2. Capturando um elemento interno

É dentro do elemento `SOFTBLUE` que os demais elementos foram criados. Portanto, é na referência deste objeto (`eSoftblue`) que iremos realizar a próxima busca, procurando pelo elemento `TITLE` dentro deste elemento, como mostra a linha 39. Observe que ela é bastante similar à linha 38. A partir do objeto contido em `eSoftblue`, o método `getElementsByTagName("TITLE")` realiza uma busca por elementos chamados `TITLE` e novamente retorna apenas a primeira posição ao fazer o uso do `[0]` no final da instrução. Este elemento é vinculado à variável `eTitle`, para que possamos manipulá-lo mais facilmente nas próximas linhas.

Agora temos um elemento (`TITLE`) em mãos que apresenta um conteúdo (`Ajax com André Milani`) no arquivo XML. Como proceder então para acessar este conteúdo? É exatamente isso que a linha 40 faz. Basta acessar a propriedade `firstChild` deste elemento (`eTitle`) e, nela, a propriedade `nodeValue`, a fim de acessar o conteúdo vinculado a este elemento. Observe que a linha 40 faz isso e vincula o texto capturado à variável `conteudoTitle`, que será utilizada posteriormente para apresentar essa e outras informações para o usuário.

Na linha 41 do exemplo apresentado, é criada uma variável chamada `buffer`, que irá armazenar o texto capturado do elemento `TITLE` e armazenará também os valores que serão capturados no decorrer do exemplo, para somente na linha 54 apresentá-los ao usuário por meio de um comando `alert` do JavaScript.

4.3. Capturando um atributo de um elemento

O elemento `TITLE` do exemplo apresentado no arquivo `conteudoSimplesXml.xml` possui um atributo chamado `codigoLivro`. Para acessar via JavaScript um atributo em um elemento XML, é necessário utilizar o método `getAttribute`, como ocorre na linha 43.

Observe que o acesso é realizado a partir da variável `eTitle`, que armazena o elemento `TITLE` capturado anteriormente na linha 39, e que agora realiza uma busca pelo atributo chamado `codigoLivro`, atribuindo o retorno desta operação à variável `eAtributo`, que tem seu conteúdo adicionado à variável `buffer` para apresentar também o seu valor ao usuário, na linha 54.

4.4. Finalizando o exemplo

Para finalizar o exemplo, basicamente é necessário capturar os textos dos outros dois elementos contidos no elemento principal, que são chamados `COUNTRY` e `COMPANY`. Esses elementos devem ser acessados exatamente como o elemento `TITLE` foi acessado anteriormente, ou seja, realizando a busca pelos mesmos (`getElementsByTagName`) a partir

do elemento contido em `eSoftblue`, que é o elemento que os contém, e sempre acessando o índice `[0]` do array retornado por esta instrução, como fazem as linhas 46 a 52.

Agora é só executar o arquivo no servidor web e clicar no botão `Obter informação de XML via AJAX` para ver o resultado na tela. O resultado pode ser visto na figura a seguir.

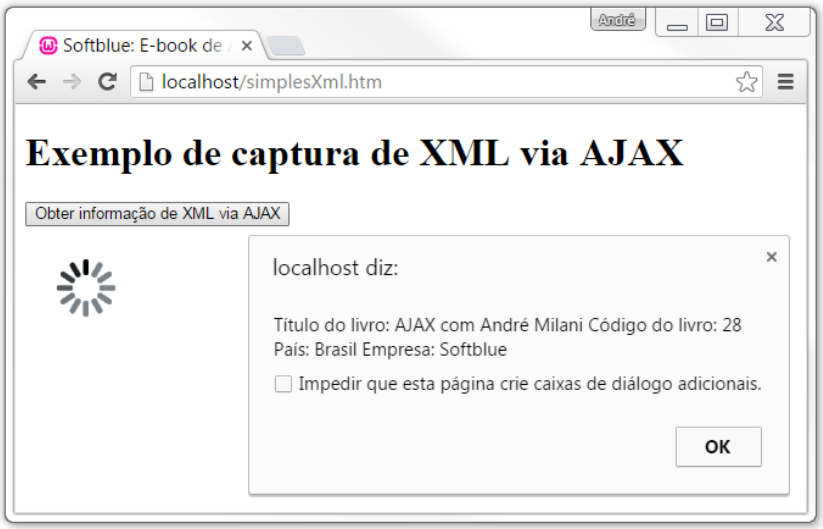

Figura 4-1. Resultado de captura de conteúdo XML via AJAX

Talvez uma dúvida esteja na sua cabeça neste momento: *"e se meu elemento apresentar dois ou mais sub-elementos com mesmo nome, como devo fazer para acessá-los?"*. Essa é uma ótima pergunta! Arquivos XML também podem apresentar listas de elementos, ou seja, elementos com o mesmo nome. Veremos como lidar com listas XML no próximo tópico deste e-book.

5. Transação AJAX com lista de valores em XML

Analisar listas de elementos XML em JavaScript não é muito diferente do que você já viu. Basicamente, é o mesmo método `getElementsByTagId()` que retorna um array com os elementos de mesmo nome. A partir daí, é só navegar neste array por meio de algum laço de repetição.

Para ilustrarmos como realizar a navegação em elementos XML via JavaScript e AJAX, considere o seguinte arquivo de dados XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOFTBLUE>
  <COURSE>
    <TITLE>Fundamentos de Java</TITLE>
    <INSTRUCTOR>Carlos Tosin</INSTRUCTOR>
    <TYPE>Regular</TYPE>
  </COURSE>
  <COURSE>
    <TITLE>PHP para Facebook</TITLE>
    <INSTRUCTOR>André Milani</INSTRUCTOR>
    <TYPE>Express</TYPE>
  </COURSE>
  <COURSE>
    <TITLE>Integração de Tecnologias</TITLE>
    <INSTRUCTOR>André Milani, Carlos Tosin</INSTRUCTOR>
    <TYPE>Webinar</TYPE>
  </COURSE>
</SOFTBLUE>
```

Código-Fonte 5-1. conteudoListaXml.xml

O arquivo `conteudoListaXml.xml` consiste em uma lista com três cursos da Softblue, cada um deles com seus respectivos elementos internos apresentando informações do curso. Portanto temos três elementos `COURSE` neste arquivo.

Neste exemplo, voltaremos a fazer uso de um painel `<div>` para apresentarmos o resultado, que montará uma tabela HTML e apresentará os cursos nesta tabela. Como é basicamente a função de tratamento de retorno que sofre consequências, primeiro será apresentado o código completo do exemplo e, em seguida, as explicações necessárias:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>Softblue: E-book de AJAX</title>
06     <script>
07       function ajaxStart()
08       {
09         var imagem = document.getElementById("imagemStatus");
10         imagem.src = "imageLoading.gif";
11
```

```
12          var url = "conteudoListaXml.xml";
13
14          if(window.XMLHttpRequest) // Maioria dos navegadores
15          {
16            req = new XMLHttpRequest();
17            req.onreadystatechange = ajaxProcessarRecebimento;
18            req.open("GET", url);
19            req.send(null);
20          }
21          else if(window.ActiveXObject) // Navegadores ActiveX
22          {
23            req = new ActiveXObject("Microsoft.XMLHTTP");
24            req.onreadystatechange = ajaxProcessarRecebimento;
25            req.open("GET", url);
26            req.send();
27          }
28        }
29
30        function ajaxProcessarRecebimento()
31        {
32          if(req.readyState == 4)
33          {
34            if(req.status == 200)
35            {
36              var xmlDoc = req.responseXML;
37
38              var buffer = "";
39              buffer += "<table>";
40              buffer += "  <tr>";
41              buffer += "    <th>Curso</th>";
42              buffer += "    <th>Instrutor</th>";
43              buffer += "    <th>Tipo</th>";
44              buffer += "  </tr>";
45
46              var eSoftblue = xmlDoc.getElementsByTagName("SOFTBLUE")[0];
47
48              var courses = eSoftblue.getElementsByTagName("COURSE");
49              for(i=0; i < courses.length; i++)
50              {
51                buffer += "<tr>";
52
53                buffer += "  <td>" + courses[i]
54                              .getElementsByTagName("TITLE")[0]
55                              .firstChild
56                              .nodeValue + "</td>";
57
58                buffer += "  <td>" + courses[i]
59                              .getElementsByTagName("INSTRUCTOR")[0]
60                              .firstChild
61                              .nodeValue + "</td>";
62
63                buffer += "  <td>" + courses[i]
64                              .getElementsByTagName("TYPE")[0]
65                              .firstChild
66                              .nodeValue + "</td>";
67
68                buffer += "</tr>";
69              }
70
71              buffer += "</table>";
72
73              document.getElementById("meuDiv").innerHTML = buffer;
```

```
74
75              var imagem = document.getElementById("imagemStatus");
76              imagem.src = "imageEmpty.gif";
77            }
78            else
79            {
80              alert(req.status + ": Possível erro é a falta do protocolo HTTP.");
81            }
82          }
83        }
84    </script>
85    </head>
86    <body>
87      <h1>Exemplo de captura de lista XML via AJAX</h1>
88
89      <input type="button" onClick="ajaxStart();"
90             value="Obter informação de lista XML via AJAX"><br>
91
92      <img src="imageEmpty.gif" id="imagemStatus"><br>
93
94      <div id="meuDiv"></div>
95    </body>
96  </html>
```

Código-Fonte 5-2. listaXml.xml

O arquivo `listaXml.xml` tem uma estrutura similar aos demais exemplos já apresentados neste e-book.

A primeira alteração é justamente na URL de destino da transação AJAX, definida pela variável `url` na linha 12. A segunda alteração é que o painel `<div id="meuDiv">`, apresentado na linha 92, será utilizado novamente para apresentar o resultado na tela. As demais alterações estão relacionadas ao processamento do retorno da transação, ou seja, todas dentro da função `ajaxProcessarRecebimento()`.

A função `ajaxProcessarRecebimento()` inicia da mesma forma que nos demais exemplos, validando a etapa (`4`) e a situação do retorno da transação AJAX (`200`). O conteúdo capturado é acessado via `responseXML` na linha 36 e atribuído à variável `xmlDoc`, para ser mais fácil de manuseá-lo no decorrer do código.

Para organizar os registros dos elementos `COURSE` encontrados no arquivo XML, foi criada uma variável chamada `buffer` na linha 38. Esta mesma variável é populada com alguns valores entre as linhas 39 e 44, montando o início de uma tabela HTML com três colunas. A ideia neste caso é justamente essa: montaremos via JavaScript uma tabela HTML para ser apresentada na tela no final do processo.

Em seguida, na linha 46, é capturado o elemento `SOFTBLUE` e atribuído à variável `eSoftblue`. Como este elemento é único e é o elemento raiz, o uso de `[0]` no final do método `getElementsByTagName` foi utilizado, para acessar diretamente a primeira (e única) posição do array de retorno.

É na linha 48 que as coisas mudam um pouco quando comparado ao exemplo abordado anteriormente neste e-book.

A captura dos elementos `COURSE` é realizada por meio da chamada `eSoftblue.getElementsByTagName("COURSE")`, pois é no elemento `eSoftblue` que os elementos `COURSE` estão contidos. Ainda, observe que neste caso (linha 48) não foi utilizado o índice `[0]` no final. Isso ocorre porque, desta vez, será retornada uma lista de elementos com o nome `COURSE`, já que existem vários elementos deste tipo no arquivo XML. Neste caso, como queremos acessar todos os elementos, o array retornado é atribuído integralmente para a variável `courses`, e não apenas a primeira posição, para que a navegação pelos elementos seja realizada na sequência.

O grande segredo da navegação na lista de elementos XML via JavaScript é o laço de repetição criado na linha 49, que define uma variável auxiliar chamada `i` para navegar da posição `0` até a posição final do array, que teve seu tamanho acessado via `courses.lentgh` na condição do laço de repetição `for`. A partir daí, basta acessar cada valor e continuar a montagem da tabela HTML, como é feito entre as linhas 51 e 68.

Observe que os acessos aos textos de cada elemento foram feitos em uma única instrução. As linhas 53, 54, 55 e 56, por exemplo, representam uma única instrução JavaScript. Apenas foram escritas dessa forma para ficarem organizadas no arquivo, mas basicamente o que ocorre nessas linhas é o acesso ao elemento contido na posição `i` do array `courses` na linha 53; a chamada ao método `getElementsByTagName`, buscando pelo elemento `TITLE` nesta posição do array e acessando sua única posição de retorno `[0]` na linha 54; o acesso à propriedade `firstChild` na linha 55; e `nodeValue` na linha 56. Junto com tudo isso, as marcações `<tr>` e `</tr>` foram concatenadas com o resultado, continuando a construir a tabela que está sendo armazenada na variável `buffer`.

Após o laço de repetição `for` encerrar na linha 69 a marcação `</table>` é fechada na linha 71 e, finalmente, na linha 73 o código HTML da tabela construída é renderizado no painel `div` existente, cujo código `id` tem valor `meuDiv`.

Ufa, hein! O resultado da operação abordada neste exemplo pode ser visualizado na figura apresentada a seguir.

Figura 5-1. Resultado de captura de lista de elementos XML via AJAX

Vale a pena lembrar que os arquivos XML também podem ser dinâmicos, ou seja, você pode fazer sua transação AJAX apontar para um arquivo PHP ou JAVA, que renderize em tempo real o conteúdo que você deseja acessar, e retorná-lo em formato XML para a sua página processar.

Do ponto de vista do JavaScript e HTML não muda absolutamente nada. Basta você aprender a gerar arquivos XML nas tecnologias web com as quais você trabalha.

6. It's party time!

É isso aí! Chegamos ao final de mais um e-book aqui da Softblue! O assunto AJAX é relativamente curto, pois os conhecimentos para utilizá-lo já foram, em boa parte, abordados nos e-books de HTML e de JavaScript.

Estou certo de que muitos conhecimentos foram ganhos por você no decorrer dos estudos deste e dos demais e-books da Softblue. E digo, sem sombra de dúvidas, que fico muito orgulhoso de você e de todos os meus alunos por se dedicarem tanto a esta profissão, que considero uma das mais complexas do mundo, com uma taxa de desistência altíssima já nas primeiras semanas de aprendizado. Certamente se você chegou até este ponto, você já cruzou a fronteira dos que desistem da área, e foi para o lado dos que irão fazer muito sucesso!

Foram três e-books devorados, portanto a dica agora é comemorar! *It's party time!* Parabéns, comemore, divirta-se e depois descanse um pouco e se prepare: a jornada não acabou por aqui, pois há muito a ser aprendido pela frente. Portanto, um passo de cada vez, sempre! O importante é não parar.

Deixo aqui um grande abraço para você, e nos vemos em breve nos materiais e eventos da Softblue! Novamente: parabéns pela conquista, e até breve =D

Lista de Figuras

Lista de Códigos-Fonte

Índice Remissivo