Desmistificando o AJAX

Descrição : Explica o que é, para que serve e como implementar o AJAX
Nível: Iniciante
Responsável : Raphael Paiva
Contato: raphael@sena.com.br **MSN:** raphaelpaiva@msn.com
Data : 14/11/2005

1. O que é AJAX

Antes de explicar o que é AJAX, deve-se explicar o que ele não é; AJAX não é um framework, uma API nem uma tecnologia em si, é uma funcionalidade implementada por um conjunto de objetos de JavaScript, sendo o mais importante chamado XMLHttpRequest.

Este objeto, que trata uma requisição ou resposta de servidor com um documento XML DOM, contém uma série de métodos que possibilita que o browser possa realizar requisições e receber respostas do servidor sem que este tenha que atualizar(*refresh*) a tela.

2. A finalidade do AJAX

O principal problema resolvido com AJAX é a substituição da conhecida tela escondida ou "*hidden frame*", que era implementado como única solução para a realização de uma requisição sem *refresh* da página principal.

Com *hidden frame* tínhamos vários problemas:

Problemas com *Hidden Frame*
• Páginas com vários quadros • Quando ocorriam erros na página escondida, estes não eram rastreados com facilidade. • Dificuldade de manutenção. • O desenvolvedor, ao ver uma já implementada, tinha bastante receio em adicionar ou modificar alguma funcionalidade dela, pois poderiam ocorrer erros em outros locais. • Normalmente deveria ser implementado um novo jsp para cada método, assim, caso fosse necessário a utilização de várias funcionalidades, eram necessários mais páginas escondidas na mesma página. • Não se tinha um local que centralizasse e controlasse todas as requisições.

AJAX resolve este problema, possibilitando, através de um único método, realizar *request* e receber *responses* com ilimitadas respostas, ou seja, a comunicação cliente-servidor fica transparente, fazendo com que sem nenhuma dificuldade o desenvolvedor possa acessar métodos do servidor quase como se fosse um método JavaScript.

3. A Aplicabilidade do AJAX

Alguns usos mais comuns do AJAX podem ser listados:

1. **Validação em tempo real:** Validações que não possam ser feitas do lado do cliente, como, por exemplo, verificar se usuário já está cadastrado ou se a data informada é anterior à data atual.
2. ***Auto Completion*:** Possibilita que o ao mesmo tempo em que o usuário for digitando, possa aparecer uma lista de possíveis respostas.

   -Um bom exemplo é o Google Suggest (http://www.google.com/webhp?complete=1&hl=en)
3. **Visualização de detalhes de um item:** Ao invés de carregar todos os dados na tela ou então necessitar de *popups,* pode-se montar a lista de "itens-pai" e dependendo da escolha, montar os detalhes do item.

   -Para melhor exemplificação, um bom exemplo é a seção de notícias e de empregos do portal JavaFree(www.javafree.org)**.**
4. **Controles de interface de usuário sofisticados:** Controles dinâmicos como arvore de diretórios, menus, barras de progresso e interface ricas como aplicações RIA ou até mesmo jogos podem ser implementados sem necessidade de *refresh.*

   -Um exemplo de interface rica pode ser visualizada no site Flickr(http://flickr.com/), onde o usuário pode organizar uma coleção de fotos com diversos recursos, como por exemplo com utilização de *drag n´ drop.*

   -Um exemplo do famoso jogo Lemmings desenvolvido utilizando AJAX pode ser visto em (http://193.151.73.87/games/lemmings/)
5. **Atualização de dados na página:** Atualização de informações na página em tempo real sem a necessidade de *refresh* possibilita, por exemplo, o desenvolvimento de *chats*, acompanhamento de ações de bolsa, notícias ou aplicações semelhantes.

   -Um exemplo de chat existe no site QWAD(http://www.qwadchat.com).

   -Um exemplo de atualização de notícias segundo a segundo está em (http://digg.com/spy)

4. A arquitetura do AJAX

AJAX(Asynchronous Javascript And XML) modifica um pouco a arquitetura das aplicações atuais.

Enquanto as aplicações clássicas tinham seqüência de troca de informações totalmente síncrona, AJAX, por *default* tem uma seqüência assíncrona, porém esta pode ser configurada para ser síncrona como será explicado depois.

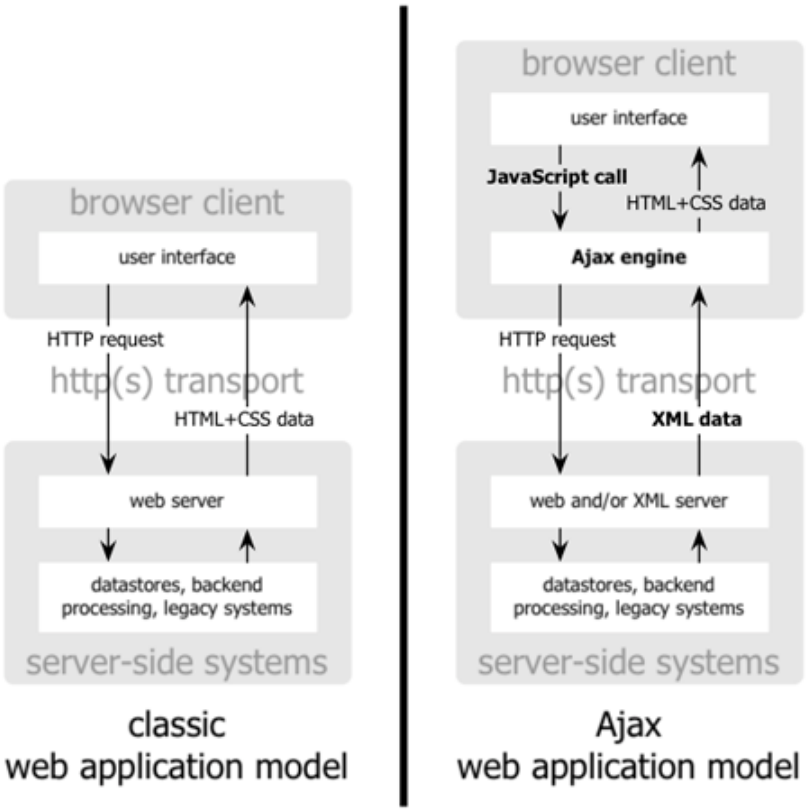


Figura 1 - Comparativo entre a arquitetura clássica e a AJAX

Agora com o AJAX é possível termos um centralizador/controlador de requisições e respostas, implementando assim o padrão MVC-2 e possibilitando uma melhor manutenibilidade do sistema. Mas a frente estaremos exemplificando.

4.1 Sincronismo X Assincronismo

A diferença entre requisições síncronas e assíncronas é quando se espera receber a resposta.

Num modelo síncrono das aplicações clássicas, temos chamadas e respostas seqüenciais, ou seja, é necessário recebermos a resposta da requisição anterior antes que passemos para a próxima requisição. Este tipo de modelo é o usado na arquitetura clássica, onde, caso precise chamar o servidor, é preciso realizar um *refresh* da página, e na página atualizada (ou na próxima página) receber a resposta.

A ilustração a seguir descreve a diferença dos modelos:

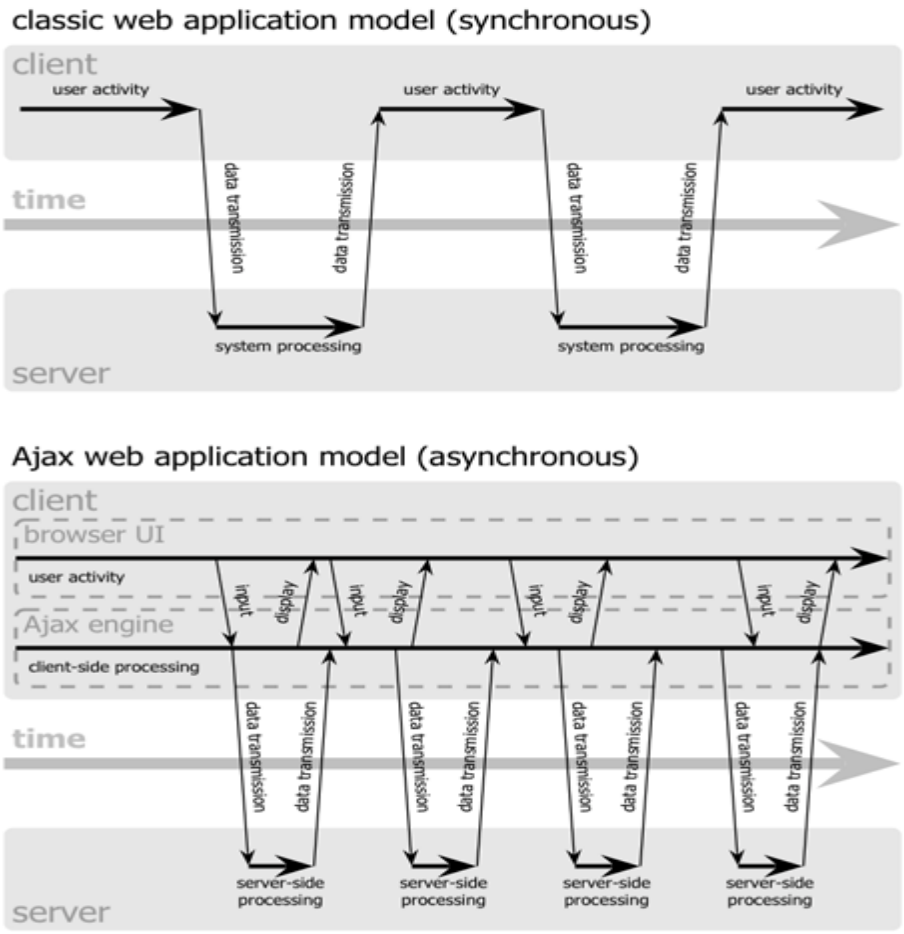


Figura 2 - Demonstração de modelo síncrono e assíncrono

No AJAX podemos utilizar ambos os modelos dependendo da necessidade, e em ambos os casos, não precisamos nem sequer realizar *refresh* da página.

Por exemplo, imagine que estamos num cadastro de usuário e, desejamos procurar pela cidade de acordo com o CEP informado; para este caso podemos utilizar o modelo **assíncrono**, pois não é de grande importância que essa informação chegue para que outras sejam executadas, assim, outras lógicas e métodos podem estar rodando em paralelo. Com o modelo **assíncrono**, no momento em que a resposta chegar ao cliente, será disparado um método que irá tratar e utilizar as informações retornadas do servidor.

Imagine agora, que na mesma tela de cadastro de usuário queremos verificar a idade deste usuário que apenas informou a data de nascimento; caso o usuário for maior de 18 anos, podemos habilitar algumas informações e campos na tela, caso menor, não deve mostrar; para este caso podemos utilizar o modelo **síncrono**; pois é crucial que o usuário de 18 anos preencha ou verifique tais dados antes de clicar em 'salvar'. Com o modelo **síncrono**, ao chamar o método que se deseja, a aplicação cliente fica aguardando receber a resposta do servidor, e só executa os demais métodos depois deste ter retornado.

Atenção: Deve-se utilizar o modelo síncrono apenas em um caso de real necessidade, pois caso o servidor nunca responda, a aplicação cliente ficará "travada" até que se dê um *time-out*.

5. Primeiros passos: Explicando um exemplo simples

Como dito no início, o AJAX na verdade se resume a um único objeto javascript chamado XMLHttpRequest, que possibilita chamadas ao servidor sem a necessidade de *refresh*.

Temos duas implementações deste objeto, uma seguida pelo Internet Explorer e outra seguida pelo Mozilla (a maioria dos outros browser do mercado seguem a implementação do Mozilla) e podemos instanciar da seguinte forma:

Criando um objeto XMLHttpRequest

```
1. function chamaAjax(){
2.     var req;
3.     var isIE;
4.     if (window.XMLHttpRequest) {
5.           req = new XMLHttpRequest();
6.     }else if (window.ActiveXObject) {
7.           isIE = true;
8.           req = new ActiveXObject("Microsoft.XMLHTTP");
9.     }
```

Com o código acima, teremos nossa variável 'req' instanciada com um objeto XMLHttpRequest para qualquer *browser*.

OBS: em algumas versões antigas do IE, deve ser instanciado o objeto "Msxml2.XMLHTTP" ao invés de "Microsoft.XMLHTTP". Um exemplo de código mais detalhado para todos os casos será demonstrado no decorrer do tutorial.

Utilizando o XMLHttpRequest para transportar dados para o servidor

```
10.     var url = "ajax?nome=raphael&tecnologia=java;
12.     req.onreadystatechange = processRequest;
13.     req.open("GET", url, true);
14.     req.send(null);
15. }
```

O código acima faz a tarefa de passar ao servidor, através do método http GET, os parâmetros para processamento. Após a linha 14 ser executada, a requisição é transmitida.

OBS: Neste código já aparecem 3 propriedades do objeto XMLHttpRequest, estas e outras mais serão explicadas uma a uma posteriormente.

Declarando a Servlet a ser usada para o Ajax no web.xml

```
1. <servlet>
2.     <servlet-name>ServletAjax</servlet-name>
3.     <servlet-class>ServletAjax</servlet-class>
4. </servlet>
5. <servlet-mapping>
6.     <servlet-name>ServletAjax</servlet-name>
7.     <url-pattern>/ajax</url-pattern>
8. </servlet-mapping>
```

Acima demonstra como mapear no web.xml da aplicação o servlet que ficará responsável por receber as requisições. Qualquer requisição feita com o comando "ajax" será redirecionada para a Servlet descrita a seguir.

Implementando a Servlet que receberá as requisições das páginas

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. public class ServletAjax extends HttpServlet {
4.     protected void service(HttpServletRequest request, HttpServletResponse response)
5.                                                 throws ServletException, IOException {
6.         String nome = request.getParameter("nome ").trim();
7.         String tecnologia = request.getParameter("tecnologia ").trim();
8.         //LÓGICAS NECESSÁRIAS
9.         response.getWriter().write(nome + "gosta de " + tecnologia);
10.     }
11.}
```

Acima é exemplificado o Servlet que lerá toda requisição feita pelo Ajax neste nosso exemplo. Nela, é possível pegar todos os parâmetros passados na requisição, realizar as lógicas necessárias, como acesso a banco, validação de valores, etc.

Através do método write da classe PrintWriter, pode-se retornar qualquer valor que seja *String*, *int* ou *array* de *chars* (para maiores detalhes, o JavaDoc [http://java.sun.com/j2se/1.4.2/docs/api/index.html] é nosso amigo).

Pronto! Já temos nosso Servlet que recebe, trata e responde às requisições ao cliente! Tudo isso, lembrando, sem *refresh*; agora precisamos escrever o método no cliente para que este receba a resposta e mostre-a ao usuário.

Implementando método de recepção de respostas do servidor

```
1. function processRequest() {
2.     if (req.readyState == 4) {
3.         if (req.status == 200) {
4.             var texto = req.responseText;
5.             alert("resposta = "+texto);
6.         }
7.     }
8. }
```

O método javascript acima é responsável por, simplesmente recuperar o texto enviado pelo servidor e mostrá-lo ao usuário.

6. Utilizando no dia-a-dia: Um exemplo real

No exemplo anterior, tínhamos a tarefa apenas de enviar um dado ao servidor e retorná-lo ao cliente e conseguimos executá-la com sucesso, ela já serve para algumas implementações simples de pedido de retorno de um dado simples para o usuário, porém, AJAX pode fazer bem mais! Não retornando apenas um dado, mas uma lista destes, podendo inclusive trabalhar com folhas-de-estilo(CSS) para melhor visualização do usuário.

Já existem uma série de frameworks que trabalham com AJAX, e estes podem ser incorporados a **qualquer** aplicação web, com a utilização de **qualquer** framework MVC, ou seja.. não ache que você está preso ao framework ou ao AJAX, você pode adicioná-lo e retirá-lo de onde quiser e quando quiser.

Alguns exemplos de framework AJAX do mercado:

Framework e APIs de AJAX mais utilizadas na atualidade:
• DWR - http://getahead.ltd.uk/dwr/ • AjaxAnywhere - http://ajaxanywhere.sourceforge.net/ • SaJax - http://www.modernmethod.com/sajax/ • Rico - http://openrico.org/rico/home.page • BackBase - http://www.backbase.com • AjaxTags - http://ajaxtags.sourceforge.net/

Todos estes frameworks acima são bastante utilizados no mercado mundial, têm grande suporte e uma série de funcionalidades visuais, como drag´n drop de componentes, preenchimento de grids em tempo real, sugestão de escrita(como no google suggest), etc.

O exemplo que iremos dar neste tutorial não utilizará de nenhum desses frameworks, iremos implementar o AJAX de forma pura, dessa vez retornando do servidor um XML com todos os valores (ao invés de apenas um texto, como no exemplo anterior) e iremos preencher um uma lista de valores.

6.1 Requisitos do exemplo

O que iremos implementar para este exemplo é preenchimento de um *combobox* com uma pequena lista de cidades de acordo com o estado selecionado em outra combo, e após a seleção da cidade, nos será mostrado a quantidade de fãs do Java.

Abaixo, a lista de cidades de acordo com o estado:

Estado	Cidade	Nº de Javaneses
CE	-Caucaia -Fortaleza -Sobral	60 400 1
PB	-João Pessoa	200
PE	-Caruaru -Recife	40 550

Iremos dessa vez, ao invés de retornamos um texto para o servidor, um XML com os dados limpos de formatação HTML, no caso, um arquivo com a seguinte característica:

Arquivo xml de retorno para o cliente

```
<?xml version='1.0' ?>
<root>
   <estado>
      <cidade nome='Caucaia'>
          <javaneses>15</javaneses>
       </cidade>
       <cidade nome='Fortaleza'>
          <javaneses>400</javaneses>
       </cidade>
       <cidade nome='Sobral'>
           <javaneses>1</javaneses>
       </cidade>
    </estado>
</root>
```

Para a utilização de xml como passagem de valores, deveremos ter algumas modificações tanto no Servlet, quanto no javascript que tratará do retorno. O web.xml continuará o mesmo.

Nosso Servlet, que estamos chamando de ServletAjax sofrerá duas pequenas mudanças (além da lógica de negócio).

Serão adicionadas duas linhas a serem passadas como parâmetro para o HTML, uma é para informarmos que nossa resposta será um xml e estará codificado no padrão “UTF-8”; a segunda é apenas uma informação para que o browser não utilize seu cache.

6.2 Mãos na massa

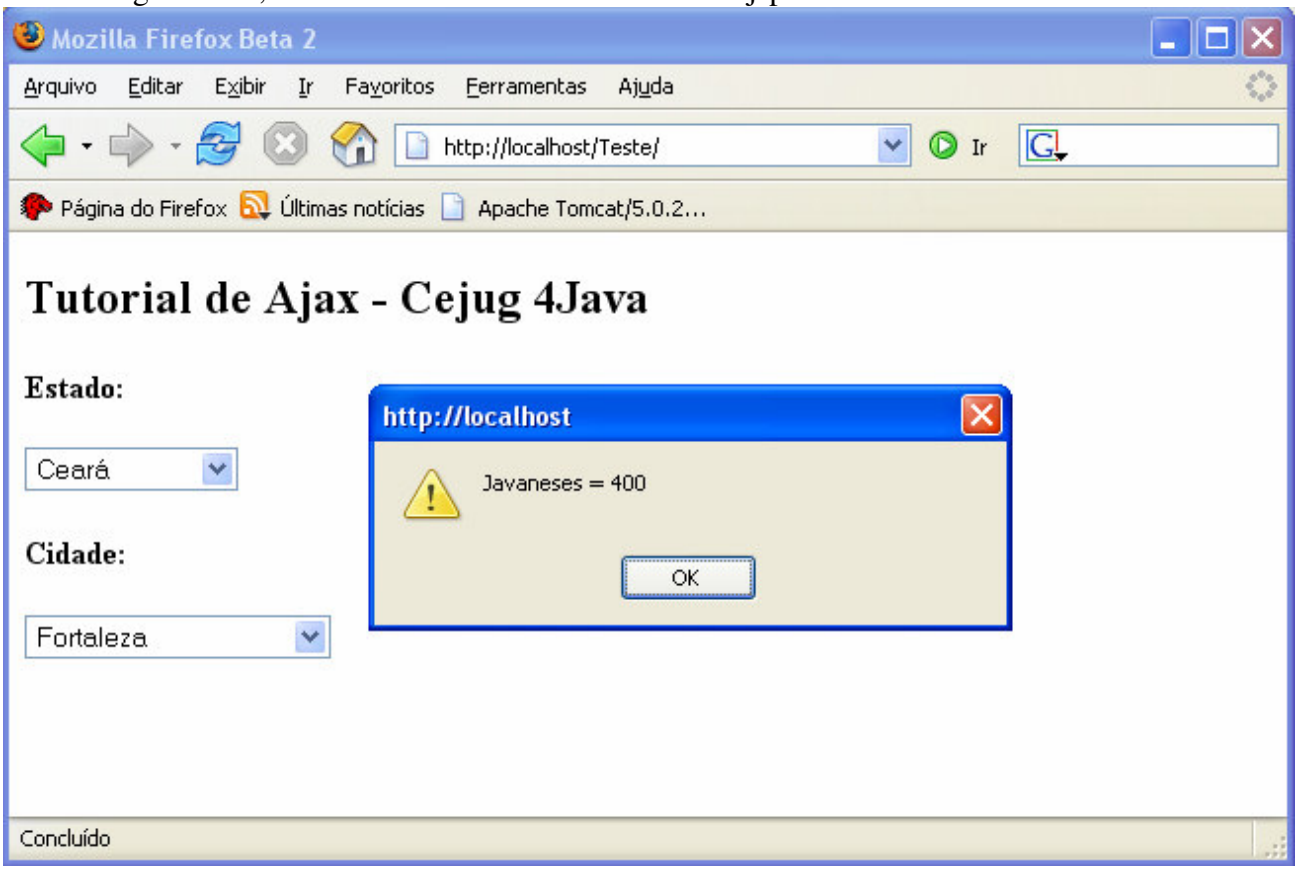
ServletAjax que receberá as requisições das páginas e retornará um XML como resposta

```
1.import javax.servlet.*;
2.import javax.servlet.http.*;
3.public class ServletAjax extends HttpServlet {
4. protected void service(HttpServletRequest request, HttpServletResponse response)
5.                                                    throws ServletException, IOException {
6.              String uf = request.getParameter("uf");
7.              String retorno = "<?xml version='1.0' ?><root>";
8.              if(uf.equalsIgnoreCase("CE")){
9.                      retorno +=  "<estado>"+
10.                                    "<cidade nome='Caucaia'>"+
11.                                     "<javaneses>15</javaneses>"+
12.                                   "</cidade>"+
13.                                   "<cidade nome='Fortaleza>"+
14.                                     "<javaneses>400</javaneses>"+
15.                                   "</cidade>"+
16.                                   "<cidade nome='Sobral'>"+
17.                                     "<javaneses>1</javaneses>"+
18.                                   "</cidade>"+
19.                                 "</estado>";
20.             }else if(uf.equalsIgnoreCase("PB")){
21.                     retorno +=  "<estado>"+
22.                                   "<cidade nome='João Pessoa'>"+
23.                                     "<javaneses>200</javaneses>"+
24.                                   "</cidade>"+
25.                                 "</estado>";
26.             }else if(uf.equalsIgnoreCase("PE")){
27.                     retorno +=  "<estado>"+
28.                                   "<cidade nome='Recife'>"+
29.                                     "<javaneses>550</javaneses>"+
30.                                   "</cidade>"+
31.                                   "<cidade nome='Olinda'>"+
32.                                     "<javaneses>30</javaneses>"+
33.                                   "</cidade>"+
34.                                 "</estado>";
35.             }
36.             retorno += "</root>";
37.             response.setContentType("text/xml;charset=UTF-8");
38.             response.setHeader("Cache-Control", "no-cache");
39.             response.getWriter().write(retorno);
40.     }
41.}
```

O código do SerletAjax acima espera receber como parâmetro (como visto na linha 6) a UF desejada pelo usuário, através dela faz-se uma pequena lógica para retornar o XML resultante.

Perceba que a linha 37 é MUITO importante, sem ela, frases com caracteres com acento, por exemplo, não seriam reconhecidas; no nosso caso, João Pessoa iria para o browser como Jo?o Pessoa.

Logo abaixo, veremos como ficará a tela do nosso jsp:



E adiante, o fonte completo da jsp. Note que o javascript, a chamada ao AJAX e o HTML estão na mesma página apenas para fins didáticos:

JSP da página de escolha de cidades

```
1.<html>
2.<head>
3.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4.<script type="text/JavaScript">
5.var req;
6.var isIE;
7.function initRequest(url) {
8.      if (window.XMLHttpRequest) {
9.         req = new XMLHttpRequest();
10.     } else if (window.ActiveXObject) {
11.         isIE = true;
12.         req = new ActiveXObject("Microsoft.XMLHTTP");
13.     }
14.}
15.function atualizaCidades(uf){
16.     if(uf != ''){
17.         var url = "ajax?uf="+escape(uf);
18.         initRequest(url);
19.         req.onreadystatechange = processRequest;
20.         req.open("GET", url, true);
21.             req.send(null);
22.     }else{
23.         var html = "<SELECT NAME='Cidades'><OPTION value=''>----</OPTION></SELECT>";
24.         document.getElementById("DivCidades").innerHTML = html;
25.     }
26.}
27.function processRequest() {
28.     if (req.readyState == 4) {
29.         if (req.status == 200) {
30.             var xml = req.responseXML;
31.             var html = "<SELECT NAME='Cidades' onChange='if(this.value!=\"\"){alert(\"Javaneses =
32.\"+this.value);}'><OPTION value=''>--Selecione Cidade--</OPTION>";
33.             var documento = xml.childNodes[0].nodeName;
34.             var estados = xml.getElementsByTagName("estado");
35.             for(i = 0; i < estados.length; i++){
36.                     var cidades = estados[i].getElementsByTagName("cidade");
37.                     for(j = 0; j < cidades.length; j++){
38.                             var nomeCidade = cidades[j].getAttribute("nome");
39.     var javaneses = cidades[j].getElementsByTagName("javaneses").item(0).firstChild.data;
40.                             html += "<OPTION value='"+javaneses+"'>"+nomeCidade+"</OPTION>";
41.                     }
42.             }
43.             html += "</SELECT>";
44.             document.getElementById("DivCidades").innerHTML = html;
45.         }
46.     }
47.}
48.</script>
49.</head>
50.<body>
51.<form method="post" name="frm">
52.<h2>Tutorial de Ajax - Cejug 4Java</h2>
53.<h4>Estado:</h4>
54.<SELECT NAME='Estados' onchange="atualizaCidades(this.value);">
55.     <OPTION value=''>--Selecione--</OPTION>
56.     <OPTION value='CE'>Ceará</OPTION>
57.     <OPTION value='PB'>Paraíba</OPTION>
58.     <OPTION value='PE'>Pernambuco</OPTION>
59.</SELECT>
60.<h4>Cidade:</h4>
61.<div id="DivCidades" style="position: static;">
62.<SELECT NAME='Cidades'><OPTION value=''>----</OPTION></SELECT>
63.</div>
64.</form>
65.</body>
66.</html>
```

7. Conclusão

No nosso caso, demonstramos um exemplo bem simples, porém capaz de apresentar a 'ponta do iceberg' do desenvolvimento Web com AJAX.

A popularização do AJAX ajudou e muito no nascimento de novos termos, como a Web 2.0 (http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html), uma quase renascença da Internet, desta vez, liderada pelo Google; e foi justamente o Google onde tivemos a primeira aplicação com alta popularidade que utilizou o AJAX, o GMail.

O mais interessante é que AJAX não é novo! Já era utilizado por inúmeros desenvolvedores mundo afora, porém, com a criação do acrônimo para referenciá-lo a Javascript, XML e chamadas assíncronas, popularizou-se em poucas semanas; o artigo que "batizou" a "velha" tecnologia não muito difundida foi escrito por Jesse James Garret no site da Adaptive Path (http://www.adaptivepath.com/publications/essays/archives/000385.php), onde me serviu de primeira base de estudos (e onde "colei" as duas figuras sobre o modelo assíncrono e a arquitetura do AJAX).

Após a leitura do artigo, me interessei e fui atrás de novas fontes, principalmente com exemplos, e onde eu fiz meu primeiro *test-drive* foi no catálogo de *BluePrints* do Java.net (https://bpcatalog.dev.java.net/nonav/ajax/). A partir daí, e até hoje, estou pesquisando mais sobre o assunto aos poucos. A maior e melhor fonte que posso indicar é a wikipedia (http://en.wikipedia.org/wiki/AJAX) com inúmeros artigos, *frameworks*, exemplos, scripts e histórico sobre o tema.

Pessoalmente ainda não utilizo nenhum *framework* que implemente AJAX, pois normalmente estes são voltados para a camada visual da aplicação, e ainda não tive oportunidade (ou necessidade) de utilizar algo semelhante, porém, acho que um dia eu chegue a utilizar algum; caso se interesse por algum *framework*, listei alguns que li no início do capítulo 6, vale a pena escolher algum para estudar e usar. Pretendo que meu próximo artigo/tutorial sobre AJAX seja um comparativo entre estes *frameworks* ou algo semelhante a isto (qualquer idéia, pode entrar em contato comigo ☺).

Durante o tutorial não cheguei a comentar em detalhes os métodos do objeto XMLHttpRequest para que ficasse algo mais direto à prática, já que falei razoavelmente sobre a teoria por detrás; assim, no apêndice a seguir, organizei os principais métodos e suas funções.

Apêndice

CRIANDO UM OBJETO	
FORMA	**BROWSER**
`var req = new XMLHttpRequest();`	Mozilla e outros
`var req = new ActiveXObject("Microsoft.XMLHTTP");`	Internet Explorer
`var req = new ActiveXObject("Msxml2.XMLHTTP");`	Internet Explorer

MÉTODOS DO OBJETO	
MÉTODO	**FUNÇÃO**
abort()	Cancela a atual requisição
getAllResponseHeaders()	Retorna a coleção de cabeçalhos do http como uma String
getResponseHeader("campoCabecalho")	Retorna o valor do campo do cabeçalho
open("metodo","URL",assincrono)	Especifica o método, a URL e outros atributos opcionais de uma requisição. O parâmetro "método" pode ser GET, POST ou PUT. O parâmetro URL pode ser uma url relativa ou completa. O parâmetro assíncrono informa se a requisição será assíncrona ou síncrona: true significa que o script continuará após a execução do método send; false significa que o script ficará esperando a resposta do servidor no método send.
send(conteudo)	Envia a requisição
setRequestHeader("campo","valor")	Adiciona um par (campo/valor) ao cabeçalho http a ser enviado

PROPRIEDADES DO OBJETO	
PROPRIEDADE	**DESCRIÇÃO**
onreadystatechange	Indica o evento a ser chamado cada vez que o estado do objeto muda.
readyState	Informa o estado do objeto da requisição: 0 = objeto não inicializado 1 = carregando 2 = carregado 3 = interagindo 4 = completado
responseText	Retorna a resposta como uma string
responseXML	Retorna a resposta como um XML (que pode ser tratada como um W3C DOM)
status	Retorna o estado da requisição como um número (ex: 404 para "não encontrado" e 200 para "OK")
statusText	Retorna o estado da requisição como uma string (ex: "Não encontrado" ou "OK")