AJAX - Tutorial

Anselmo Luiz Éden Battisti, Christiano Julio Pilger de Brito
Universidade Estadual do Oeste do Paraná
Reitoria - DRI
Cascavel - Paraná - Brasil
anselmobattisti@yahoo.com.br, chrisjulio@gmail.com

18 de agosto de 2006

1 AJAX - Sem Mistérios

AJAX (acrónimo em língua inglesa de *Asyncronous Javascript And XML*) é o uso sistemático de *Javascript* e XML (e derivados) para tornar o navegador mais interativo com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX não é somente um novo modelo, é também uma iniciativa na construção de aplicações *web* mais dinâmicas e criativas. AJAX não é uma tecnologia, são realmente várias tecnologias trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades. AJAX incorpora em seu modelo:

- Apresentação baseada em padrões, usando XHTML e CSS;
- Exposição e interação dinâmica usando o DOM;
- Intercâmbio e manipulação de dados usando XML e XSLT;
- Recuperação assíncrona de dados usando o objeto XMLHttpRequest;
- *Javascript* unindo todas elas em conjunto.

O modelo clássico de aplicação *web* trabalha assim: A maioria das ações do usuário na interface dispara uma solicitação HTTP para o servidor *web*. O servidor processa algo recuperando dados, mastigando números, conversando com vários sistemas legados e então retorna uma página HTML para o cliente. É um modelo adaptado do uso original da *web* como um agente de hipertexto, porém o que faz a *web* boa para hipertexto não necessariamente faz ela boa para aplicações de software.

Esta aproximação possui muito dos sentidos técnicos, mas não faz tudo que um usuário experiente poderia fazer. Enquanto o servidor está fazendo seu trabalho, o que o usuário estará fazendo? O que é certo, esperando. E a cada etapa em uma tarefa, o usuário aguarda mais uma vez.

Obviamente, se nós estivéssemos projetando a *web* do nada para aplicações, não faríamos com que os usuários esperassem em vão. Uma vez que a interface está carregada, por que a interação do usuário deveria parar a cada vez que a aplicação precisasse de algo do servidor? Na realidade, por que o usuário deveria ver a aplicação ir ao servidor toda vez?

A maior vantagem das aplicações AJAX é que elas rodam no próprio navegador *web*. Então, para estar hábil a executar aplicações AJAX, bastar possuir algum dos navegadores modernos, ou seja, lançados após 2001, são eles: Mozilla Firefox, Internet Explorer 5+, Opera, Konqueror e Safari.

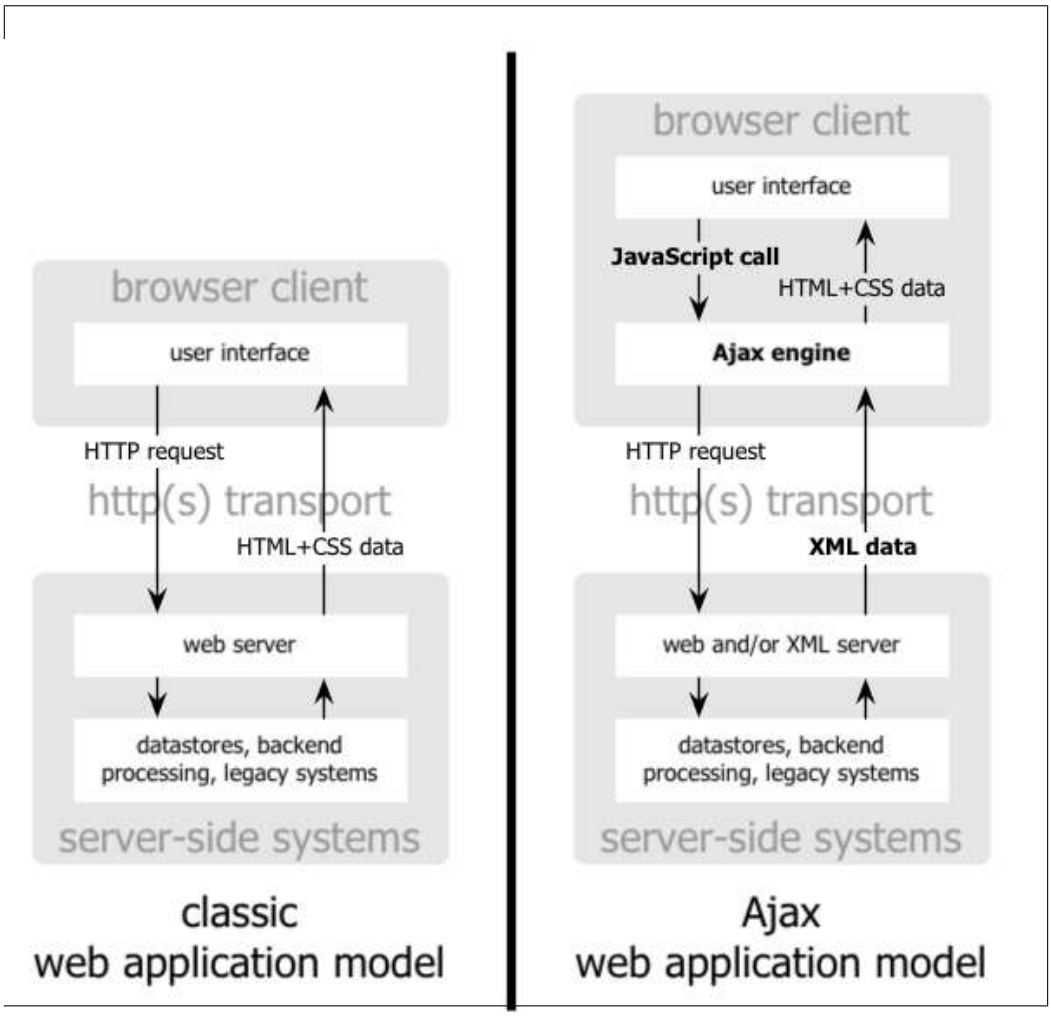

Figura 1: Diagrama de requisições ao servidor

2 A Classe Assíncrona

Para se fazer um pedido HTTP ao servidor usando *Javascript*, você precisa de um objeto que disponibiliza essa funcionalidade. Tal classe foi primeiro introduzida no Internet Ex-

plorer sob a forma de um objeto ActiveX chamado XMLHTTP, então, o Mozilla, o Safari e outros browsers seguiram-se, implementando uma classe de nome XMLHttpRequest que suportava os métodos e as propriedades do objeto ActiveX original da Microsoft.

O código abaixo mostra a forma genérica de instanciar um objeto que proverá os mecanismos para pedidos assíncronos.

Código 1: 'ajaxInit.js'

```
1  function ajaxInit() {
2
3    var xmlhttp;
4
5    try {
6       xmlhttp = new XMLHttpRequest();
7    } catch(ee) {
8       try {
9          xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
10      } catch(e) {
11         try {
12            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
13         } catch(E) {
14            xmlhttp = false;
15         }
16      }
17   }
18
19   return xmlhttp;
20 }
```

Os métodos e atributos padrões estão descritos abaixo.

2.1 Atributos

Segue abaixo alguns dos atributos do objeto de conexão assíncrona.

readyState : Código que diz o status da solicitação assíncrona, ele varia de 0 até 5

- 0 (uninitialized);
- 1 (a carregar);
- 2 (carregado);
- 3 (interativo);
- 4 (completo).

status : Status do retorno do HTML, são os códigos padrões do HTML 200 ok, 400 no found

responseText : Retorna a cadeia de caracteres que o servidor enviou.

responseXml : Retorna o XML o servidor enviou.

onreadystatechange : Define qual função será chamada para fazer a manipulação dos dados assim que houver um retorno.

http_request.onreadystatechange = nameOfTheFunction;

Vamos ver o que é que esta função deve fazer. Primeiro, a função precisa de verificar o estado do pedido, se o estado possui o valor 4, isso significa que a totalidade da resposta do servidor foi recebida e que pode continuar a processá-la à vontade, a próxima coisa a verificar é o código do estado da resposta HTTP do servidor. Para os nossos objetivos só estamos interessados na resposta 200 OK.

```
if (http_request.readyState == 4) {
   if (http_request.status == 200) {
      // deu certo
   } else {}
}
```

2.2 Métodos

Segue abaixo alguns dos métodos do objeto de conexão assíncrona.

open(mode, url, boolean)

- mode: Tipo da requisição, GET ou POST
- url: URL do objeto solicitado no modo assíncrono, pro questões de segurança. O Firefox não permite que a URL esteja em um servidor diferente da página que esta fazendo a solicitação.
- boolean: *true* (assíncrono) ou *false* (síncrono).

send() É o método SEND que ativa a conexão e faz a requisição de informações ao documento aberto pelo método OPEN. Este método possui somente um parâmetro que serve para enviarmos dados extras ao documento que estamos acessando. Usamos este parâmetro quando, por exemplo, no método OPEN, acessamos o documento com POST ao invés de GET, neste caso os dados do POST são passados neste parâmetro de SEND.

3 Exemplo 1 Solicitando um arquivo de texto

No exemplo 1 iremos trabalhar com os elementos básicos vistos até agora

Código 2: 'Exemplo 1'

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
2  <html>
3     <head>
4        <title>Exemplo 1 - Carregar Arquivo do Servidor</title>
5        <script type="text/javascript" src="ajaxInit.js"></script>
6        <script type="text/javascript">
7        function carregar() {
8           ajax = ajaxInit();
9           document.getElementById("texto").innerHTML = "Carregando ...";
10          if (ajax) {
11             ajax.open('GET','http://200.201.81.38/anselmo/ciclo/texto.txt',true);
12             ajax.onreadystatechange = function() {
13                if (ajax.readyState == 4) {
```

```
14                      if(ajax.status == 200){
15                          document.getElementById("texto").innerHTML = ajax.responseText;
16                      }
17                  }
18              }
19          }
20          ajax.send(null);
21      }
22      </script>
23  </head>
24  <body>
25  <div id="texto"></div>
26  <input type="button" value="Carregar" onClick=" carregar() "/>
27  </body>
28  </html>
```

Na linha 8 ocorre a chamada para a função *ajaxInit()* que esta dentro do arquivo referenciado na linha 5. A linha 9 *getElementByid('texto')* [1] e *innerHTML*[2] inserem o texto Carregando dentro do DIV com o id igual a ẗexto:.

Nas linhas 11, 12 e 13 são verificados o *status* do pedido, e do retorno e efetiva inserção do conteúdo do arquivo texto.txt no DIV, respectivamente.

Neste primeiro exemplo não houve nenhum processamento por parte de uma linguagem de programação no servidor, porém no mundo real geralmente uma chamada no modo assíncrono se comunica com um *script* (PHP, JSP, ASP, Ruby, CGI, PERL ...), que faz um processamento sobre os dados a ele enviados e por fim retorna o resultado, nosso próximo exemplo mostrará exatamente isto.

4 Exemplo 2 Criando uma calculadora

Em nosso segundo exemplo iremos introduzir o uso de uma linguagem *server-side* realizando um processamento sobre os dados enviados no modo assíncrono, isto será feito criando uma calculadora.

Nossa calculadora será composta de 2 caixas de texto para a entrada dos valores e uma label para exibição dos resultados, a única operação que ela realiza é a soma.

Código 3: 'Calculadora HTML'

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
2  <html>
3      <head>
4          <title>Exemplo 2 - Calculadora </title>
5          <script type="text/javascript" src="ajaxInit.js"></script>
6          <script type="text/javascript">
7          function calcular() {
8              ajax = ajaxInit();
9              document.getElementById("resultado").innerHTML = "Calculando  ...";
10             if (ajax) {
11                 ajax.onreadystatechange = resultado;
12                 v1 = document.getElementById("v1").value;
13                 v2 = document.getElementById("v2").value;
14                 url = 'http://200.201.81.38/anselmo/ciclo/calcular.php?v1='+v1+'&v2='+v2;
```

[1] Comando em *Javascript* para selecionar o elemento cujo id é igual a texto.

[2] Comando em *Javascript* para inserir uma *string* em um elemento de modo que esta seja interpretada como código HTML.

```
15              ajax.open('GET',url,true);
16              ajax.send(null);
17          }
18      }
19
20      function resultado() {
21          if (ajax.readyState == 4) {
22              if(ajax.status == 200){
23                  document.getElementById("resultado").innerHTML = ajax.responseText;
24               }
25          }
26      }
27      </script>
28  </head>
29  <body>
30      <form name="calculadora">
31          <input type="text" name="v1" id="v1" size="3"/> + <input type="text" name="v2"
                id="v2" size="3"/>  =
32          <label id="resultado"></label> <br/>
33          <input type="button" value="Carregar" onClick="calcular()"/>
34      </form>
35  </body>
36 </html>
```

Código 4: 'Calculadora PHP'

```
1 <?
2     echo $_GET['v1'] + $_GET['v2'];
3 ?>
```

Na linha 11 o método *onreadystatechange* assume uma função externa a função que cria o objeto ajax.

A linha 12 e 13 cria duas variáveis que armazenam os valores dos campos que estão no formulário, este dado são pegos através do *value* [3].

A variável **url** da linha 14 irá conter o endereço e as variáveis que serão usadas pelo PHP para calcular a soma dos resultados.

O arquivo PHP que realiza o cálculo da soma é realmente simples, ele pega as duas variáveis **v1** e **v2** do *array $_ GET* do PHP, soma seus valores e escreve o resultado.

Até o momento o resultado de nossas chamadas assíncronas foram escritos de forma direta através do método *innerHTML*, o uso deste método é desaconselhado pelo W3C pelas seguinte razões.

- É uma propriedade desenvolvida pela *Microsoft*, por esta razão proprietária, porém a maior parte dos navegadores incorporou-a.
- Por não ser padrão esta sendo abolida das novas versões destes navegadores
- O resultado é um *string* ao passo que o DOM é uma estrutura hierárquica em XHTML

Os próximos dois exemplo são idênticos, diferenciando apena a forma como os dados são passados do servidor para o cliente, o primeiro usando texto normal e o outro uma *string* no formato *JSON*, usando então o DOM para exibir o resultado da requisição.

[3] Propiedade de alguns objetos em *Javascript* que retorna o conteúdo do mesmo

5 Exemplo 3 Preenchimento um *SELECT* a partir de outro *SELECT*

Um exemplo clássico do preenchimento de um *SELECT* a partir de outro é a escolha das cidades de um determinado estado. Este recurso é uma das maiores 'pedras no sapato' dos programadores de sistema *web*, pois quando o usuário seleciona o estado toda a página deve ser recarregada, inclusive os valores dos demais campos já preenchidos, para que o *SELECT* com as cidades seja exibido.

Com AJAX esta tarefa fica bem mais fácil pois basta que seja feita uma consulta das cidades do estado que esta no primeiro *SELECT* e inserir as cidades no segundo *SELECT*.

5.1 Usando *InnerHTML*

Código 5: 'Cidades HTML'

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-br"
lang="pt-br">
   <head>
      <title>Exemplo 3 - Escolha a Cidade</title>
      <script type="text/javascript" src="ajaxInit.js"></script>
      <script type="text/javascript">
      function selecionaCidade(estado) {
         ajax = ajaxInit();
         if (ajax) {
            ajax.onreadystatechange = escreveCidades;
            url = 'http://200.201.81.38/anselmo/ciclo/cidades.php?estado='+estado;
            ajax.open('GET',url,true);
            ajax.send(null);
         }
      }

      function escreveCidades() {
         if (ajax.readyState == 4) {
            if(ajax.status == 200){
               document.getElementById("cidade").innerHTML = ajax.responseText;
             }
         }
      }
      </script>
   </head>
   <body>

      <form name="selecioneCidade">
         <label for="estado" accesskey="e"><u>E</u>stado :</label>
         <select name="estado" id="estado" onChange="if (this.options[this.selectedIndex
             ].value){selecionaCidade(this.options[this.selectedIndex].value)}">
            <option value="pr">PR</option>
            <option value="sp">SP</option>
         </select><br/>

         <label for="cidade" accesskey="c"><u>C</u>idade :</label>

         <select name="cidade" id="cidade">
         </select>
      </form>
   </body>
```

```
43 </html>
```

Código 6: 'Cidades PHP'

```
1  <?
2  header("Content-Type: text/html; charset=iso-8859-1");
3
4  $estado = $_GET['estado'];
5  $cidades = "";
6  if ($estado=="pr"){
7     $cidades .= "<option value=\"1\">Tupãssi</option>";
8     $cidades .= "<option value=\"2\">Toledo</option>";
9     $cidades .= "<option value=\"3\">Cascavel</option>";
10    $cidades .= "<option value=\"4\">Pato Branco</option>";
11 }
12
13 if ($estado=="sp"){
14    $cidades .= "<option value=\"5\">Mogi</option>";
15    $cidades .= "<option value=\"6\">Palmeiras</option>";
16    $cidades .= "<option value=\"7\">Santos</option>";
17 }
18
19 echo $cidades;
20 ?>
```

O arquivo Cidades.php gera os elementos *options* que serão inseridos dentro do *SELECT* das cidades. Esta abordagem não é muito inteligente pois a quantidade de informação transmitidas do servidor para o cliente em sua maioria foi desnecessária. Da cidade apenas o código e o nome deveriam ser transmitidos, as demais informações são repetidas e podem ser geradas pelo navegador.

Na linha 31 do código fonte 5 é feita a verificação se um dos elementos do *SELECT* estado foi selecionado, se sim então envie o valor para a função **selecionaCidade**.

O código fonte 6 a variável estado é recebida e de acordo com ela é passado ao navegador as opções do *SELECT* cidade.

5.2 Usando *JSON*

JSON (com a mesma pronuncia do nome "Jason"em inglês), um acrônimo para *Javascript Object Notation*, é um formato leve para intercâmbio de dados computacionais. JSON é um subconjunto da notação de objeto de Javascript, mas seu uso não requer Javascript exclusivamente.

A simplicidade de JSON tem resultado em seu uso difundido, especialmente como uma alternativa para XML em AJAX. Uma das vantagens reivindicadas de JSON sobre XML como um formato para intercâmbio de dados neste contexto, é o fato de ser muito mais fácil escrever um analisador JSON. Em Javascript mesmo, JSON pode ser analisado trivialmente usando a função eval(). Isto foi importante para a aceitação de JSON dentro da comunidade AJAX devido a presença deste recurso de Javascript em todos os navegadores *web* atuais.

Na prática, os argumentos a respeito da facilidade de desenvolvimento e performance do analisador são raramente relevados devido aos interesses de segurança no uso de eval() e a crescente integração de processamento XML nos navegadores *web* modernos. Por esta

razão JSON é tipicamente usado em ambientes onde o tamanho do fluxo de dados entre o cliente e o servidor é de supra importância (daí seu uso por Google, Yahoo, etc., os quais servem milhões de usuários), onde a fonte dos dados pode ser explicitamente confiável, e onde a perda dos recursos de processamento XSLT no lado cliente para manipulação de dados ou geração da interface, não é uma consideração.

Enquanto JSON é freqüentemente posicionado "em confronto"com XML, não é incomum ver tanto JSON como XML sendo usados na mesma aplicação. Por exemplo, uma aplicação no lado cliente a qual integra dados do Google Maps com dados atmosféricos através de SOAP, requer suporte para ambos formatos de dados.

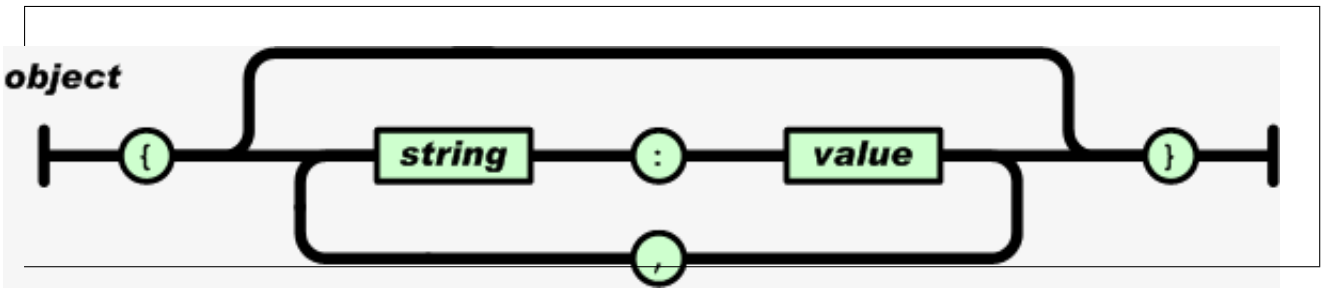

Figura 2: Automato que gera a linguagem JSON

A figura 2 ilustra em forma de um autômato finito a gramática da linguagem *JSON*. A *string* abaixo mostra um retorno das cidades do estado do Paraná.

```
{
'cidades':[
{'cdg':'1','nm':'Tupãssi'},
{'cdg':'2','nm':'Toledo'},
{'cdg':'3','nm':'Cascavel'},
{'cdg':'4','nm':'Pato Branco'}
   ]
}
```

Usando a função *eval*[4] do *Javascript* podemos transformar esta *string* em uma estrutura de objetos facilmente manipuláveis.

Se usarmos a função *eval* no retorno das cidades poderiamos por exemplo acessar o código da cidade de Pato Branco através do seguinte comando:

```
json = eval("("+ajax.responseText+")");
json.cidades[3].cdg;
```

Na primeira linha é aplicada a função *eval* e seu resultado é armazenado na variável **Json**, na linha 2 é acessado o atributo cidades, que é um vetor, na posição 3 no atributo cdg.

Para um exemplo completo veja os códigos abaixo.

[4]Introduzida na especificação 1.0 do *Javascript* esta função interpreta uma string como uma expressão, eval("fred=999; wilma=777; document.write(fred + wilma);") retorna 1776

Código 7: 'Cidades HTML'

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-br"
lang="pt-br">
   <head>
      <title>Exemplo 3 - Escolha a Cidade</title>
      <script type="text/javascript" src="ajaxInit.js"></script>
      <script type="text/javascript">

      function selecionaCidadeJson(estado) {
         ajax = ajaxInit();
         if (ajax) {
            ajax.onreadystatechange = escreveCidadesJson;
            url = 'http://200.201.81.38/anselmo/ciclo/cidadesJson.php?estado='+estado;
            ajax.open('GET',url,true);
            ajax.send(null);
         }
      }

      function escreveCidadesJson() {
         if (ajax.readyState == 4) {
            if(ajax.status == 200){
               var x = 0;
               var json = eval("("+ajax.responseText+")");

               limparDestino("cidade2");

               for(x=0;x<=json.cidades.length;x++){

                  option = document.createElement("option");

                  option.setAttribute("value",json.cidades[x].cdg);

                  option.appendChild(document.createTextNode(json.cidades[x].nm));

                  document.getElementById(document.createTextNode(json.cidades[x].nm));

                  document.getElementById("cidade2").appendChild(option);
               }
            }
         }
      }

      /*
         Remove todos os elementos filhos de um elemento
      */
      function limparDestino(destino) {
         obj = document.getElementById(destino);
         while(obj.firstChild)
                obj.removeChild(obj.firstChild);
      }

      </script>
   </head>
   <body>
      <form name="selecioneCidade">
         <label for="estado" accesskey="e"><u>E</u>stado :</label>
         <select name="estado" id="estado" onChange="if (this.options[this.selectedIndex
             ].value){selecionaCidadeJson(this.options[this.selectedIndex].value)}">
            <option value="-">---</option>
            <option value="pr">PR</option>
            <option value="sp">SP</option>
         </select><br/>

```

```
64          <label for="cidade" accesskey="c"><u>C</u>idade :</label>
65          <select name="cidade" id="cidade2">
66          </select>
67         </form>
68    </body>
69 </html>
```

Código 8: 'Cidades PHP'

```
1  <?
2  header("Content-Type: text/html; charset=iso-8859-1");
3
4  $estado = $_GET['estado'];
5  $cidades = "{'cidades':[";
6  if ($estado=="pr"){
7     $cidades .= "{'cdg':'1','nm':'Tupãssi'},
8                  {'cdg':'2','nm':'Toledo'},
9                  {'cdg':'3','nm':'Cascavel'},
10                 {'cdg':'4','nm':'Pato Branco'}]}";
11 }
12
13 if ($estado=="sp"){
14    $cidades .= "{'cdg':'5','nm':'Mogi'},
15                 {'cdg':'6','nm':'Palmeiras'},
16                 {'cdg':'7','nm':'Santos'}]}";
17 }
18
19 echo $cidades;
20 ?>
```

Revisando o Código 8 podemos ver que a linha 2 chama a função *header*, nela é passado o comando *charset=iso88591*, isto é muito importante pois diz a codificação do texto que esta sendo retornado, evitando assim que o navegador do usuário exiba de forma incorreta os acentos e outros caracteres especiais.

Com os exemplos acima pudemos ter uma boa noção de como utilizar o ajax em nossas aplicações *web*, mas, queremos coisas mais dinâmicas e com efeitos por isso vamos fazer um pequeno *Google Sugest*

6 Sugerindo um Conteúdo

Nosso próximo exemplo consistirá em um campo de texto que a cada tecla pressionada irá ao servidor e trará uma lista de palavras que comecem com o que já foi escrito. Isto será exibido em uma DIV. Quando clicarmos em uma das palavras que vieram do servidor a DIV irá sumir e o texto irá para dentro da caixa de texto.

Este sistema é utilizado por exemplo no serviço de e-mail do **Yahoo**, nele, quando digita-se o nome do contato ele já vai filtrando os que começam com as letras já digitadas.

Código 9: 'Auto Completar HTML'

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
2  <html>
3     <head>
4        <title>Auto Completar</title>
5        <script type="text/javascript" src="ajaxInit.js"></script>
6        <script type="text/javascript">
7           function sugerir(s) {
```

```
8               ajax = ajaxInit();
9               document.getElementById("resultado").innerHTML = "Calculando  ...";
10                  if (ajax) {
11                      ajax.onreadystatechange = resultado;
12                      url = 'http://200.201.81.38/anselmo/ciclo/autoCompletar.php?s='+s;
13                      ajax.open('GET',url,true);
14                      ajax.send(null);
15                  }
16          }
17
18          function resultado() {
19              if (ajax.readyState == 4) {
20                  if(ajax.status == 200){
21                      json = eval("("+ajax.responseText+")");
22                      var x = 0;
23                      limparDestino("resultado");
24                      document.getElementById('resultado').style.visibility='visible';
25                      for (x=0; x < json.pal.length;x++) {
26                          P = document.createElement("p");
27                          P.appendChild(document.createTextNode(json.pal[x]));
28                          P.setAttribute("onClick","document.getElementById('pal').value='"+
                                json.pal[x]+"';                                    document.
                                getElementById('resultado').style.visibility='hidden'");
29                          document.getElementById("resultado").appendChild(P);
30                      }
31                  }
32              }
33          }
34
35          /*
36              Remove todos os elementos filhos de um elemento
37          */
38          function limparDestino(destino) {
39              obj = document.getElementById(destino);
40                  while(obj.firstChild)
41                      obj.removeChild(obj.firstChild);
42          }
43      </script>
44      <style type="text/css">
45          #resultado {
46              background-color : #e5e5e5;
47              border           : 1px solid #000;
48              width            : 150px;
49              padding          : 0px;
50              margin           : 0px;
51              position         : relative;
52              margin-left      : 45px;
53              margin-top       : 0px;
54          }
55          p {
56              line-height      : 15px;
57              margin           : 0px;
58              font             : 18px "Arial",sans-serif;
59              padding          : 5px;
60          }
61          p:hover {
62              background-color : #cdcdcd;
63              cursor           : pointer;
64          }
65      </style>
66  </head>
67  <body>
68      <form name="auto">
69          <label for="pal" accesskey="pal">Palavra</label>
70          <input type="text" id="pal" name="pal" onKeyUp="sugerir(document.getElementById
```

```
                ('pal').value)"/>
71            <div id="resultado" style="visibility : hidden"></div>
72        </form>
73    </body>
74 </html>
```

Código 10: 'Auto Completar PHP'

```
1  <?
2      $p[0]     = "abacate";
3      $p[1]     = "abobora";
4      $p[2]     = "banana";
5      $p[3]     = "beco";
6      $p[4]     = "cabo";
7      $p[5]     = "cebo";
8      $p[6]     = "dedo";
9      $p[7]     = "dado";
10     $p[8]     = "elefante";
11     $p[9]     = "faca";
12     $p[10]    = "gato";
13     $p[11]    = "porta";
14
15     $search = $_GET['s'];
16     $x = 0;
17     $res = "{'pal':[";
18     for ($x = 0;$x<11;$x++){
19          if($search==substr($p[$x],0,strlen($search))){
20             $res .= "'".$p[$x]."',";
21          }
22     }
23     $res = substr($res,0,(strlen($res)-1));
24     $res .= "]}";
25     echo $res;
26 ?>
```

Vamos analisar o Código 9, na linha 24 é chamada a função *style*[5].*visibility*[6] define que o resultado esta oculto.

Na linha 28 duas funções são adicionadas ao parágrafo com o texto que veio do servidor, a primeira função coloca a palavra dentro da caixa de texto e a segunda função esconde o DIV resultado.

Agora vamos ver o Código 10 em PHP, existe um vetor chamado $p com 12 palavras, a linha 17 e 20 montam a *string* no formato *json*, a linha 23 remove a virgula da última palavra.

6.1 Ligando a um Banco de Dados

A integração de AJAX com bancos de dados não difere em nada do que foi feito até agora, pois ao fim do uso do banco de dados pelo cliente o servidor envia uma *string* de volta.

Nosso exemplo usará o mesmo arquivo HTML do código anterior, apenas modificaremos o arquivo .php que será chamado por ele.

Código 11: 'Conexão com o Banco'

[5] Acessa os estilos em CSS em um objeto

[6] Atributo que define se o objeto esta visível 'visible' ou oculto 'hidden'

```
1  <?php
2  # FileName="Connection_php_mysql.htm"
3  # Type="MYSQL"
4  # HTTP="true"
5  $hostname_conexao = "localhost";
6  $database_conexao = "test";
7  $username_conexao = "aluno";
8  $password_conexao = "aluno";
9  $conexao = mysql_pconnect($hostname_conexao, $username_conexao, $password_conexao) or die
       (mysql_error());
10 mysql_select_db($database_conexao, $conexao);
11 ?>
```

Código 12: 'Auto Completar Usando Banco de Dados'

```
1  <?
2      include("conexao.php");
3
4      $search = $_GET['s'];
5
6      $sql= "SELECT * FROM municipio WHERE mnc_descricao like '".$search."%'";
7
8      $consulta = mysql_query($sql,$conexao);
9      $linhas  = mysql_fetch_array($consulta);
10
11     $res = "{'pal':[";
12     do {
13           $res .= "'".str_replace("'"," ",$linhas['mnc_descricao'])."',";
14     } while ($linhas = mysql_fetch_array($consulta));
15
16     $res = substr($res,0,(strlen($res)-1));
17     $res .= "]}";
18     echo $res;
19 ?>
```

O código 11 cria uma conexão com o banco de dados, a linha 9 conecta com o servidor usando o usuário e senha e a linha 10 inicia o uso do banco de dados **text**.

O código 12 faz a pesquisa dos municípios no banco de dados, na linha 2 é incluída o arquivo que faz a conexão, na linha 4 a variável $search recebe a palavra que foi digitada na caixa de texto, na linha 13 o comando *str_ replace*[7] foi utilizado para evitar problemas com palavras que por ventura contenham aspas simples como por exemplo d'agua; se este tratamento não for feito na hora de chamar a função *eval* ocorreria um erro.

7 Vantagens no uso do Ajax

Alguns benefícios que temos ao usar AJAX são:

- Torna possível utilizar uma aplicação baseada na *web* com recurso e funcionalidades parecidos com os que possuímos hoje em nossos sistemas *Desktop*. Isto torna o uso dos aplicativos *web* realmente usáveis;
- Como a modificação das informações da tela são parciais uma grande quantidade de informações deixa de trafegar inutimente pela rede;

[7]Substitui todas as ocorrências da *string* procurada na *string* de retorno

- O servidor que roda a aplicação fica menor carregado pois existe uma divisão de tarefas com o cliente;
- AJAX não é uma tecnologia por isto não é necessário pagar para a utilizar.

Porém como já dizia o poeta "nem tudo são flores ", temos que:

- AJAX não é a solução milagrosa para todos os males e se usado de forma inadequada pode piorar uma situação que já era feia;
- Os Navegadores usam diferentes métodos por isso temos que estar atentos quanto a eles para não impossibilitar o uso de nossos sistemas por tais navegadores;
- Como existe uma divisão de processamente entre o cliente e o servidor, temos que minimizar este processamento pois os Navegador em geral não suporta uma carga muito pesada de *scripts*;
- Os botão voltar, avança e histórico não funcionam muito bem com AJAX.

Referências

[1] Erko Bridee de Almeida Cabrera. *AJAX Visão Conceitual*. Portal Java, 2005.

[2] Steve Ganz. *Alternatives to innerHTML*. SlayerOffice, 2006.

[3] Jesse James Garrett. Ajax: A new approach to web applications, http://adaptivepath.com/publications/essays/archives/000385.php. 2005.

7.1 Para Saber Mais

- Artigo muito bom sobre AJAX
  http://adaptivepath.com/publications/essays/archives/000385.php

- Usando POST com AJAX
  http://www.dotpix.com.br/ wendel/projetos/ajax/postXMLHttpRequest/sumGet.html

- Não use innerHTML
  http://slayeroffice.com/articles/innerHTML_alternatives/#intro

- Página oficial do JSON
  http://www.json.org/

- DHTML
  http://www.quirksmode.org/js/cross_dhtml.html

- Exemplos em Ajax
  http://www.japs.etc.br/ajax/

- Material sobre Css
  http://www.maujor.com

- DOM XHTML e HTML
  http://www.amtechs.com/w3c/introduction.html