Prefácio

Nenhum termo foi tão discutido como Ajax nos últimos doze meses. Também não é pra menos, há muito os desenvolvedores de aplicações Web vem se desenrolando para trazer ao usuário a mesma experiência que existe nos sistemas criados para desktop. Ajax veio pra ficar. Não porque é uma tecnologia, mas sim porque é um conjunto de tecnologias já praticadas na Web, onde há muito tempo enriquecem sites. Mas a atenção a essa maneira de desenvolver, ficou mais conhecida com o uso no GMAIL (desenvolvido pela Google).

O termo AJAX surgiu em fevereiro 2005, por Jesse James Garrett de Adaptive Path, LLC, onde publicou um artigo *on-line* intitulado, "Ajax: A New Approach to Web Applications".

AJAX é a sigla de **Asynchronous JavaScript and XML**, e como já foi dito, não é uma tecnologia e sim o uso de tecnologias incorporadas que tem as principais o JavaScript e o XML, onde juntos são capazes de tornar o navegador mais interativo, utilizando-se de solicitações assíncronas de informações.

Esse *e-book* é um capítulo extra e pode ser considerado como complemento ao livro *Dominando Ajax*.

Foi criado com dois objetivos: primeiro, para você ter uma idéia do assunto abordado no livro; segundo, como uma contribuição à comunidade.

Sumário

Extra

Desenvolvendo o Básico com Ajax

Este capítulo não tem a intenção de ensinar ao leitor tudo o que há por trás do Ajax, isso você aprende com o livro *Dominando Ajax*.

A intenção é complementar o assunto, abordar outras possibilidades que você pode ter usando Ajax.

Combos Dinâmicos

Mais que comuns na Internet, criar caixas de combinação é uma tarefa relativamente simples, mas em HTML. Já no caso de usar dados vindos do banco de dados, onde dois combos interagem entre si, por exemplo: Estados e Cidades, aí exige chamada ao servidor.

Esse foi um dos primeiros problemas que os desenvolvedores encontraram. No começo, começaram a usar iFrame; depois, com o navegador Internet Explorer, já era possível de se chamar remotamente o servidor pelo JavaScript, sem a necessidade de um iFrame. O que você verá aqui é exatamente isso.

O Lado Servidor

Para este exemplo, escolhi fazê-lo usando PHP 5. Uma porque o livro aborda tanto PHP 5 como Java para Web e outra porque o PHP é muito

popular entre as empresas de desenvolvimento. Minha empresa, Integrator Technology and Design, é uma dessas que emprega também o PHP para desenvolvimento de sites.

A Parte Lógica do Servidor

A seguir você tem a classe que montará os combos:

resultados_class.php

```
<?php

  class Resultados
  {
   private $conexao;

   // construtor
   function __construct()
   {
    // cria a conexão com o MySQL
    $this->conexao = mysqli_connect("localhost", "edson",
                 "integrator","livraria");

   }
   // destrutor, fecha a conexão com o banco de dados
   function __destruct()
   {
    mysqli_close($this->conexao);
   }
   // gera o XML dos livros do autor selecionado
   public function geraXML($id)
   {
    // cria a query que resultará nos dados
    $queryStr = "SELECT l.isbn, titulo, nome, editora_nome
         FROM livros l, autores a,
            editora e, publicacao p
         WHERE a.autor_id=p.autor_id
           AND l.isbn=p.isbn
           AND e.editora_id =p.editora_id
           AND a.autor_id=$id";
   // executa a query
   $result = mysqli_query($this->conexao,$queryStr);

   $XML='<?xml version="1.0" encoding="UTF-8"?>';
```

```
 $XML.="\n<resultados>\n";
if(@mysqli_num_rows($result)>0){
  while ( $row = mysqli_fetch_object($result) ) {
    $XML.="<livro>\n";
    $XML.="<isbn>" . $row->isbn . "</isbn>\n";
    $XML.="<titulo>" . stripslashes( $row->titulo ) . "</titulo>\n";
    $XML.="<editora>" . stripslashes( $row->editora_nome ) . "</editora>\n";
    $XML.="<autor>" . stripslashes( $row->nome ) . "</autor>\n";
    $XML.="</livro>\n";
  }
}
 $XML.="</resultados>";

return $XML; //retorna o XML gerado
}
// método que gera o XML para autores
public function autoresXML( )
{
 $queryStr = "SELECT * FROM autores";
// executa a query
$result = mysqli_query($this->conexao,$queryStr);

$XML='<?xml version="1.0" encoding="UTF-8"?>';
 $XML.="\n<autores>\n";
  while ( $row = mysqli_fetch_object($result) ) {
    $XML.="<autor>\n";
    $XML.="<codigo>" . $row->autor_id . "</codigo>\n";
    $XML.="<nome>" . stripslashes( $row->nome ) . "</nome>\n";
    $XML.="</autor>\n";
  }
 $XML.="</autores>";

return $XML; //retorna o XML gerado
}
// end class
}
?>
```

Essa classe tem dois métodos públicos que serão importantes para montar dois combos, um para montar um combo com os nomes de autores e outro para mostrar os livros escritos por este autor, depois de selecionado.

Caso você não conheça PHP 5, o **Apêndice A**, do livro Dominando Ajax, o ajudará nessa empreitada.

O Relacionamento

O livro propõe a criação de um banco de dados usando o MySQL. Esse banco de dados se chama livraria e tem as tabelas como mostrado na **Figura 1** a seguir:

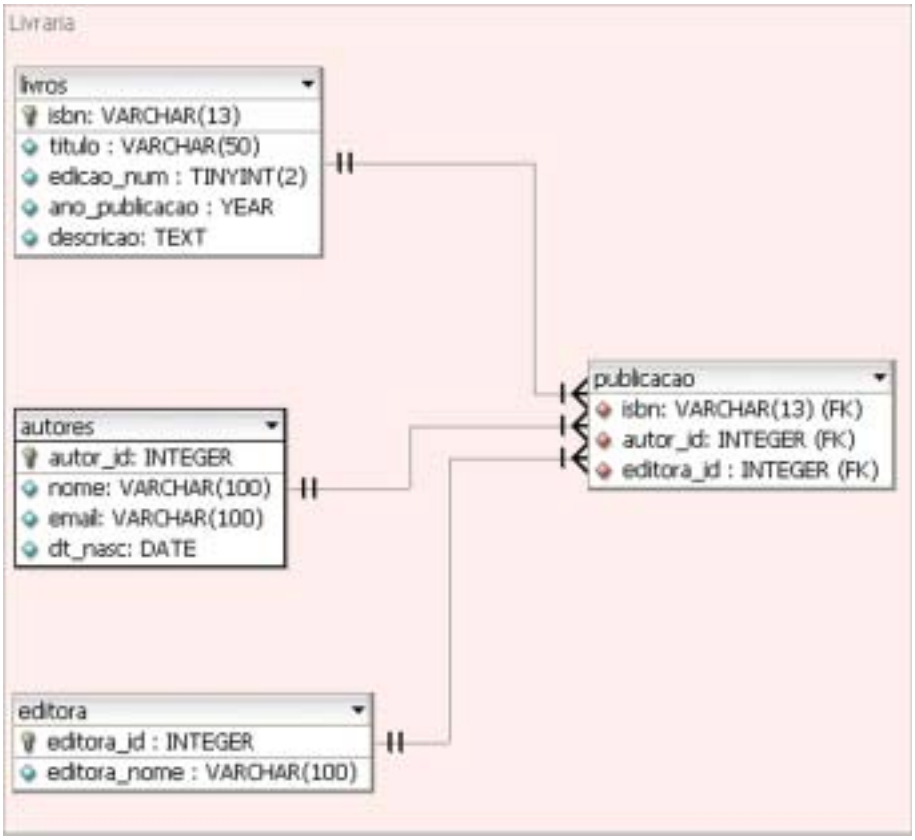

Figura 1

Os comandos para criar uma tabela e outros existentes para manipulação do banco de dados em questão, você encontra no **Apêndice E** do livro.

A Página que Irá Gerar os XML's

A página **resultados.php** é a que entrará em comunicação com o JavaScript da sua aplicação Ajax.

resultados.php

```
<?php
  require_once('resultados.class.php'); //inclui a classe Resultados
  // captura a ação
  $id = $_REQUEST['id'];

  // cria a instância de Resultados
  $resultados = new Resultados();

  // headers enviados para evitar cache no browser
  header('Expires: Fri, 25 Dec 1980 00:00:00 GMT');
  header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
  header('Cache-Control: no-cache, must-revalidate');
  header('Pragma: no-cache');
  // gera a saída no formato XML
  header('Content-type: text/xml; charset=UTF-8',true);
  if(!empty($id))
   echo $resultados->geraXML($id); // gera o XML de todos os dados
  else
   echo $resultados->autoresXML( ); // gera o XML de todos os dados
  ?>
```

Essa página recebe o ID do autor e, caso não esteja vazio, o método **geraXML( )** é chamado.

Dê uma boa olhada no livro, onde uma explicação minuciosa é feita a respeito da questão do desenvolvimento do formato XML.

A Lógica no Lado Cliente

O arquivo **resultados.js** é o responsável por fazer a comunicação com o servidor e receber dele os dados.

Assim que receber, o JavaScript terá que separar as informações e distribuí-las para seus respectivos lugares.

Esse arquivo não será explicado, recomendo o **Capítulo 4** do livro *Dominando Ajax* para que você entenda como ele funciona:

resultados.js

```
// JavaScript Document
  var xmlHttp=criaXMLHttpRequest( );
  var paginaResult = "resultados.php";
  var resultadosId = "livros";
  function init()
  {
     carregarAutores()
     return;
  }

  function criaXMLHttpRequest() {
    var xmlHttp;
    if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XmlHttp");
    }
    else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
    }
    return xmlHttp;
  }

  function carregarAutores()
  {
   // continua somente se o objeto XMLHttpRequest não der problema
   if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
   {
    var query = paginaResult;
    xmlHttp.open("GET", query, true);
    xmlHttp.onreadystatechange = mostraAutores;
    xmlHttp.send(null);
   }
  }

  function mostraAutores( )
  {
   // se readyState é 4, lê a resposta do servidor
   if (xmlHttp.readyState == 4)
   {
    // continua somente se o status for 200
    if (xmlHttp.status == 200)
    {
     // le a resposta
     resposta = xmlHttp.responseText;
     // erro no servidor?
     if (resposta.indexOf("ERRNO") >= 0
```

```
      || resposta.indexOf("error") >= 0
      || resposta.length == 0)
    {
     // mostra a mensagem de erro
     alert("Erro no servidor:\n"+  resposta);
     // sai da função
     return;
    }
    // o servidor respondeu em formato XML
    respostaXml = xmlHttp.responseXML;

    var autores   = respostaXml.getElementsByTagName("autor");

    //total de elementos contidos na tag cidade
    if(autores.length > 0) {
     //percorre o arquivo XML paara extrair os dados
     for(var i = 0 ; i < autores.length ; i++) {
      var autor = autores[i];
      //contéudo dos campos no arquivo XML
      var codigo   =
autor.getElementsByTagName("codigo")[0].firstChild.nodeValue;
      var nome =  autor.getElementsByTagName("nome")[0].firstChild.nodeValue;
      geraCombo(codigo,nome,"autores");
     }
    }
   }
   else
   {
    alert("Erro na leitura da resposta pelo servidor.")
   }
  }
}
function carregarLivros(id)
{
  // continua somente se o objeto XMLHttpRequest não der problema
  if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
  {
   var query = paginaResult+"?id="+id;
   xmlHttp.open("GET", query, true);
  xmlHttp.onreadystatechange = mostraLivros;
   xmlHttp.send(null);
  }
}
function mostraLivros()
{
  limpar("livros");
```

```
  // se readyState é 4, lê a resposta do servidor
  if (xmlHttp.readyState == 4)
  {
    // continua somente se o status for 200
    if (xmlHttp.status == 200)
    {

     // le a resposta
     resposta = xmlHttp.responseText;
     // erro no servidor?
     if (resposta.indexOf("ERRNO") >= 0
       || resposta.indexOf("error") >= 0
       || resposta.length == 0)
     {
      // mostra a mensagem de erro
      alert("Erro no servidor:\n"+  resposta);
      // sai da função
      return;
     }
     // o servidor respondeu em formato XML
     respostaXml = xmlHttp.responseXML;

     var livros   = respostaXml.getElementsByTagName("livro");

     //total de elementos contidos na tag cidade
     if(livros.length > 0) {
      //percorre o arquivo XML paara extrair os dados

      for(var i = 0 ; i < livros.length ; i++) {
       var livro = livros[i];
       //contéudo dos campos no arquivo XML
       var isbn    =  livro.getElementsByTagName("isbn")[0].firstChild.nodeValue;
       var titulo =  livro.getElementsByTagName("titulo")[0].firstChild.nodeValue;
       geraCombo(isbn,titulo,"livros");
      }
     }
    }
    else
    {
     alert("Erro na leitura da resposta pelo servidor.")
    }
  }
}

// método que gera os dados do combo
function geraCombo(valor,rotulo,elemento)
```

```
{
  //cria um novo option dinamicamente
  var campo = document.createElement("option");
  //atribui um valor ao elemento OPTION
  campo.value = valor;
  //atribui um rótulo ao elemento
  campo.text  = rotulo;
  //adiciona os valores ao select autores
  document.getElementById(elemento).options.add(campo);
}
// método que limpa os dados do combo
function limpar(obj) {
    document.getElementById(obj).length = 0;
}
```

A Página que o Cliente Irá Ver

A página que o cliente irá ver esses resultados em ação é a **resultados.html**, onde haverão dois combos, um para mostrar os nomes dos autores e outro para mostrar seus livros escritos:

resultados.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">
  <head>
  <title>Autores e Livros escritos</title>
  <script src="resultados.js"></script>
  </head>

  <body onload="init( );">
  <form id="form1" name="form1" method="post" action="">
   <p>Autores:
     <select name="autores" id="autores" onchange="carregarLivros(this.value);">
    <option id="autid">Selecione o autor</option>
     </select>
   </p>
   <p>Livros Escritos:
     <select name="livros" id="livros">
     </select>
   </p>
  </form>
  </body>
  </html>
```

Quando você rodar o exemplo, você terá uma tela inicial similar à **Figura 2** a seguir:

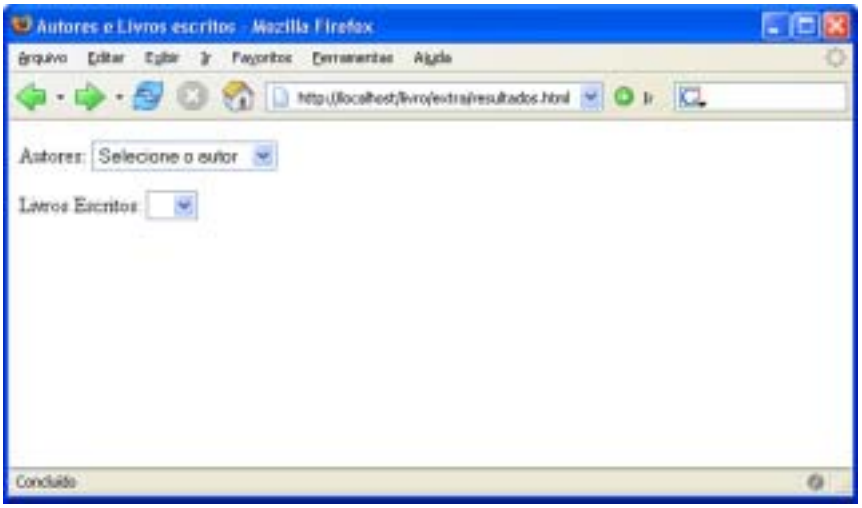

Figura 2

Ao selecionar um autor:

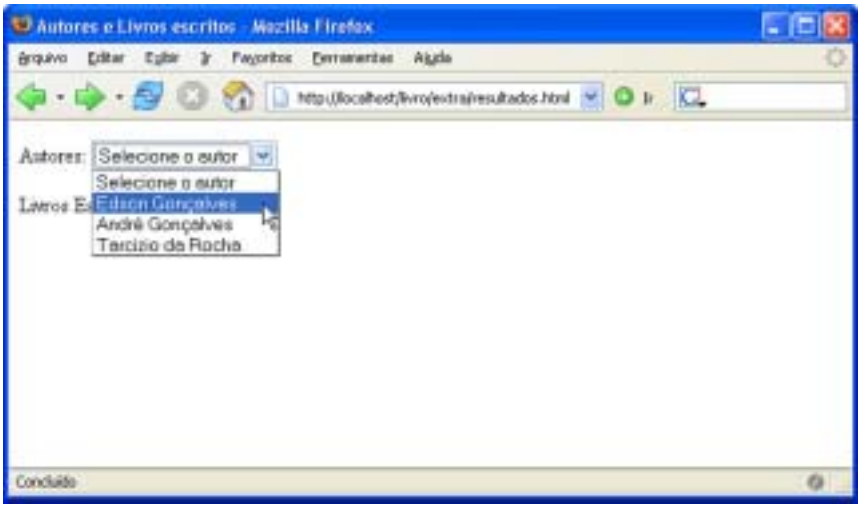

Figura 3

Você verá os livros deste autor na caixa de seleção abaixo, como mostra a **Figura 4** a seguir:

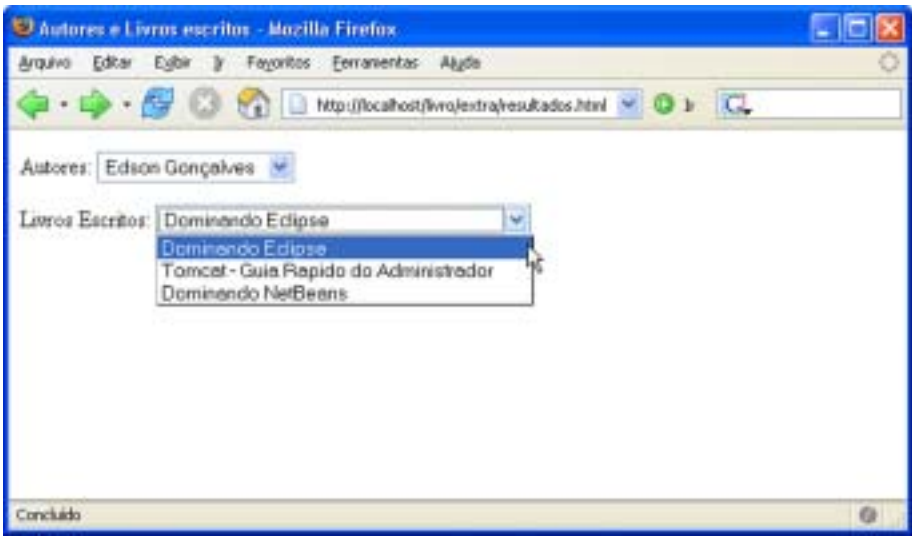

Figura 4

Validando um Formulário com Ajax

Digamos que você tenha que fazer um login e senha na sua aplicação Web para entrar em uma área restrita. Nesse caso você teria que preencher o formulário e submetê-lo.

Mas, e se antes de submeter o login e a senha, no preenchimento, você tivesse um feedback com o servidor e ele pesquisasse o login e o dissesse se existe ou não, antes mesmo de você submeter. Pois é, você construirá nesse exemplo, exatamente isso.

O Banco de Dados

Neste caso você fará um banco de dados chamado **admin** e uma tabela chamada de **tb_admin**, como mostra a **Figura 5** a seguir:

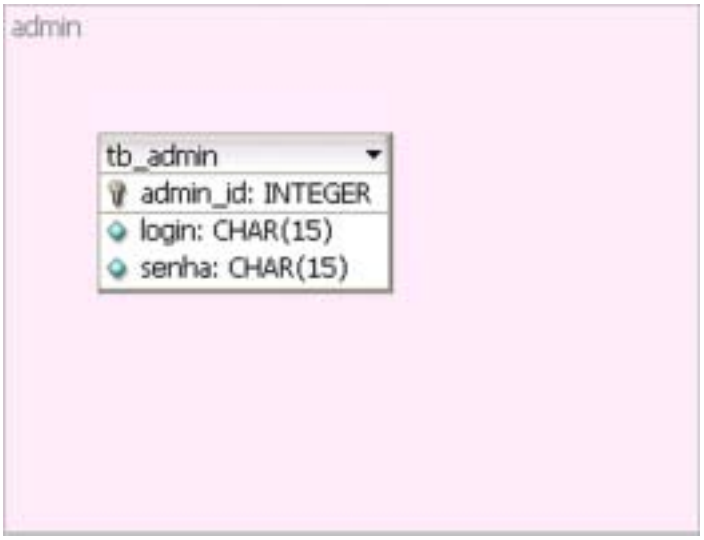

Figura 5

O Lado Servidor

Para este exemplo, você utilizará o Java. No **Apêndice E** do livro *Dominando Ajax* você tem mais informações sobre JSP.

A classe **Validar** faz a parte de conexão ao banco de dados e verifica a existência ou não de um usuário na tabela:

Validar.java

```
package dominando.ajax;

  import java.sql.*;

  public class Validar{

     private Connection getConexao(){
        Connection con=null;
        try{
            Class.forName("org.gjt.mm.mysql.Driver").newInstance();
            con =  DriverManager.getConnection(
                  "jdbc:mysql://localhost/admin", "edson", "integrator");

        } catch(Exception ex){ex.printStackTrace();}
```

```
        return con;
    }
    private String gerarConsulta(String campos,
          String tabela,
          String condicao) {

        String query = "SELECT "+campos+" FROM "+tabela+" "+condicao;

        return query;

    }
    public String pesquisaLogin(String login) {
        // cria a query que resultará nos dados
        String query = gerarConsulta("count(*) as geral",
              "tb_admin",
              "WHERE login='"+login+"'");
        String dados="";
        try{
            Connection con = getConexao();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            rs.next();
            if(rs.getInt("geral")>0){
                dados="Usuário existente";
            }
            else{
                dados="Usuário não encontrado";
            }
            rs.close();
            stmt.close();
            con.close();
        } catch(Exception ex){ex.printStackTrace();}

        return dados; //retorna o texto gerado
    }

}
```

A classe **Validar** contém o método público **pesquisaLogin( )** que se responsabiliza em verificar e retornar se existe ou não um usuário com o parâmetro recebido.

A página **valida.jsp** chama a classe e imprime o resultado encontrado, ou seja, se há ou não um usuário existente.

valida.jsp

```
<%@ page language="java" import="java.io.*" %>
  <%@ page import="dominando.ajax.Validar"%>
  <jsp:useBean id="ds" scope="page" class="dominando.ajax.Validar" />
  <%
    response.setHeader("Cache-Control","no-cache"); //HTTP 1.1
    response.setHeader("Pragma","no-cache"); //HTTP 1.0
    response.setDateHeader ("Expires", -1);
    //recupera o login digitado
    String login =  request.getParameter("login");

    String dados="";

    try{
      dados = ds.pesquisaLogin(login);
    }catch(Exception ex){}

    out.print(dados);
  %>
```

Essa página tem que recuperar o **login**, passado pelo método GET que resultará na chamada ao método **pesquisaLogin( )** da classe **Validar**. Assim que o resultado é dado, o método **print( )** responde para o navegador.

O Lado Cliente

O arquivo **validar.js** é o responsável por fazer a comunicação com o servidor e receber dele a informação resultante. Dessa vez o resultado não será em XML, portanto o JavaScript é mais simples. Caso você queira mais detalhes, o Capítulo 4 aborda profundamente o uso de XML e também de texto simples.

validar.js

```
// JavaScript Document
  var xmlHttp=criaXMLHttpRequest();
  var paginaResult = "valida.jsp";
  var resultadosId = "status";

  function criaXMLHttpRequest( ) {
```

```
var xmlHttp;
  if (window.ActiveXObject) {
    xmlHttp = new ActiveXObject("Microsoft.XmlHttp");
  }
  else if (window.XMLHttpRequest) {
    xmlHttp = new XMLHttpRequest();
  }
  return xmlHttp;
}
// método que pega o login e o envia para o servidor
function dados(login){

  if(xmlHttp) {
    var query = paginaResult+"?login="+login

     xmlHttp.open("GET", query, true);
    xmlHttp.onreadystatechange = function( ) {
      if (xmlHttp.readyState == 4) {
        if (xmlHttp.status == 200) {

          // chama o elemento status
          var status = document.getElementById('status');
          // a torna visível
          status.style.visibility='visible';
          // caso recupere, imprime a resposta na <div /> status
          status.innerHTML=xmlHttp.responseText;
        } else {
          alert('Houve um problema ao carregar o resultado.');
        }
      }
     }
  }
  xmlHttp.send(null);
}
```

A Página que o Usuário Irá Visualizar

A página que o usuário irá visualizar é simples, chamando o arquivo JavaScript que fará toda a conversa com o servidor, e chamando o método **dados( )** que recebe um parâmetro, o login que você deseja transmitir.

```
<!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">
  <head>
  <title>Área Administrativa</title>
  <script src="validar.js"></script>
  <style type="text/css">
  <!—
  #status {
    position:absolute;
    left:232px;
    top:12px;
    width:200px;
    height:18px;
    z-index:1;
    visibility: hidden;
    background:#FFCC66;
    font-weight:bold;
        padding:2px
  }
  —>
  </style>
  </head>

  <body>
  <div id="status"></div>
  <form id="form1" name="form1" method="post" action="">
    <table width="200" border="0">
      <tr>
        <td width="42" align="right">Login:</td>
        <td width="148"><input name="login" type="text" id="login"
  onblur="dados(this.value)" /></td>
      </tr>
      <tr>
        <td align="right">Senha:</td>
        <td><input name="senha" type="text" id="senha" /></td>
      </tr>
      <tr>
        <td> </td>
        <td><input type="submit" name="Submit" value="Logar" /></td>
      </tr>
    </table>
  </form>
  </body>
  </html>
```

Como resultado você terá uma tela similar à **Figura 6** caso não encontre o usuário:

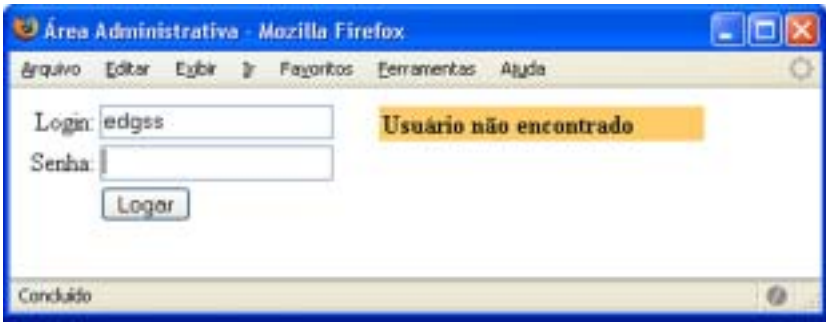

Figura 6

Ou terá uma tela igual à **Figura 7** caso o servidor encontre o usuário:

Figura 7

O interessante desse método é que você pode criar um live form, para criar atualizações de dados. Assim, você altera o que quer e ele vai salvando após perder o foco.

Criando um Auto Completar Simples

O exemplo colocado agora é um simples sistema de sugestão de informações vindas do banco de dados. Você começa digitando o título e ele

vai completando com o restante do primeiro título encontrado. A **Figura 8** ilustra melhor essa idéia:

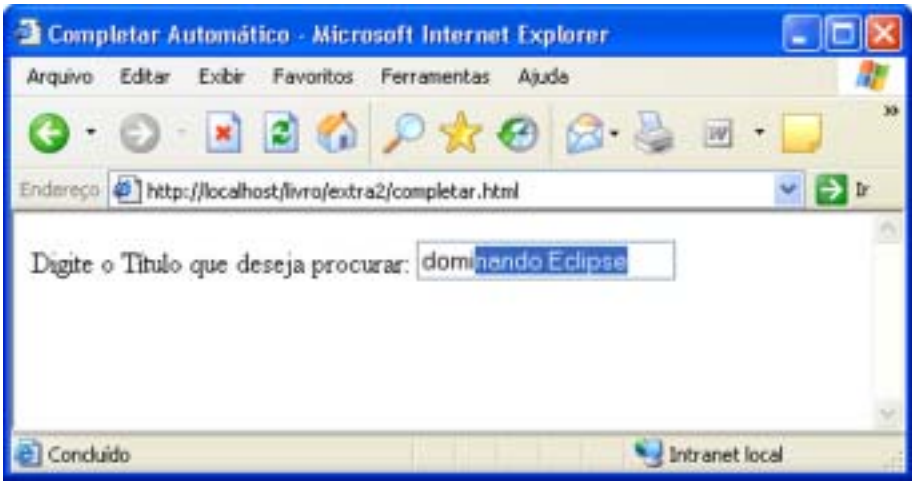

Figura 8

O Lado Servidor

Para este exemplo, você utilizará novamente PHP. A classe **Completar** faz a parte de conexão ao banco de dados e verifica a existência ou não de parte do título do livro que deseja procurar na tabela:

completar.class.php

```
<?php

  class Completar
  {
    private $conexao;

    // construtor
    function __construct( )
    {
      // cria a conexão com o MySQL
      $this->conexao = mysqli_connect("localhost", "edson",
                  "integrator","livraria");
```

```
  }
  // destrutor, fecha a conexão com o banco de dados
  function __destruct( )
  {
    mysqli_close($this->conexao);
  }
  // gera o resultado dos títulos dos livros
  public function gerar($titulo)
  {
    // cria a query que resultará nos dados
    $queryStr = "SELECT titulo
          FROM livros
          WHERE
           titulo like '$titulo%'
          ORDER BY titulo";
   // executa a query
   $result = mysqli_query($this->conexao,$queryStr);

   if(@mysqli_num_rows($result)>0){
     $row = mysqli_fetch_object($result);
       $resultado=stripslashes( $row->titulo );

   }


   return $resultado; //retorna o XML gerado
  }

// end class
}
?>
```

A query do banco de dados dessa vez traz os dados resultantes do primeiro título encontrado, em ordem alfanumérica, de acordo com a inicial, ou iniciais, passadas pelo parâmetro no método **gerar( )**. O resultado é um texto simples.

A página **completar.php** será a usada para chamar os resultados e enviá-los ao JavaScript do lado cliente:

completar.php

```
<?php
  require_once('completar.class.php'); //inclui a classe Resultados
  // captura a ação

  $titulo = $_REQUEST['titulo'];

  // cria a instância de Completar
  $completar = new Completar();

  // headers enviados para evitar cache no browser
  header('Expires: Fri, 25 Dec 1980 00:00:00 GMT');
  header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
  header('Cache-Control: no-cache, must-revalidate');
  header('Pragma: no-cache');

  if(!empty($titulo))
   echo $completar->gerar($titulo);
  ?>
```

Essa página receberá uma query string chamada **titulo** contendo as letras digitadas e, ao chamar o método **gerar( )**, esse valor é passado e recebido com o resultado do valor encontrado.

O Lado Cliente

O arquivo **autocompletar.js** é o responsável por fazer a comunicação com o servidor e receber dele a informação resultante. Como o resultado não será em XML, esse também terá um código mais simples. Caso você queira mais detalhes sobre um sistema mais complexo de autocompletar, o **Capítulo 9** aborda esse assunto.

autocompletar.js

```
function criaXMLHttpRequest() {
    var xmlHttp;
    if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XmlHttp");
    }
    else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
    }
```

```
  return xmlHttp;
}


function autocompletar (enviar, ev) {

   //processa somente teclas alfanuméricas
   if (( ev.keyCode >= 48 && ev.keyCode <= 57 )
         ||  ( ev.keyCode >= 65 && ev.keyCode <= 90 )) {

      var xmlHttp=criaXMLHttpRequest();
      var pagina = "completar.php?titulo=" + enviar.value;
      xmlHttp.open("GET", pagina, true);


      xmlHttp.onreadystatechange = function ( ) {
         if (xmlHttp.readyState == 4) {
            var sugestao = xmlHttp.responseText;
            var txtAuto = document.getElementById ('txtAuto');

            if (sugestao) {
               //Firefox ou Opera ...
               if (document.getSelection) {
                  var largura = txtAuto.value.length;
                  txtAuto.value +=
                  sugestao.substring(txtAuto.value.length, sugestao.length);
                  txtAuto.selectionStart = largura;
                  txtAuto.selectionEnd = txtAuto.value.length;

               }
               //Internet Explorer
               else if (document.selection) {
             var largura = txtAuto.value.length;
                  var sel = document.selection.createRange ( );
                   sel.text = sugestao.substring(largura, sugestao.length);
                   sel.move ("character", -(sugestao.length));
                   sel.findText (sugestao.substring(largura, sugestao.length));
                   sel.select ( );


               }
            }
         }
      }
      xmlHttp.send (null);
   }
}
```

Como você já deve ter imaginado, o método **autocompletar( )** é o responsável pela lógica de completar o restante do texto digitado por você, na caixa de texto.

Esse método recebe dois parâmetros, o primeiro indica a caixa de texto e o segundo a tecla que está sendo usada.

A idéia é não chamar o servidor caso você não use teclas alfanuméricas, mesmo porque, você não vai enviar um espaço, mas pode enviar uma palavra, mais um espaço, e depois outra palavra.

Não significa que não seja possível enviar caracteres não alfanuméricos, longe disso, significa apenas que você não chamará o servidor desnecessariamente.

A página **completar.html** termina com a caixa de texto chamando o método **autocompletar( )** que você usará chamando o JavaScript que ligará o cliente ao servidor.

completar.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">
  <head>
  <title>Completar Automático</title>
  <script src="autocompletar.js"></script>
  </head>

  <body>
  <form id="form1" name="form1" method="post" action="">
  Digite o Título que deseja procurar:
    <input type="text"
       id="txtAuto" name="txtAuto"
       onkeyup="autocompletar (this,event);"
    />
  </form>
  </body>
  </html>
```

Display Dinâmico Usando Ajax

Como último exemplo, você aprenderá a fazer um display dinâmico, capaz de trazer informações para a página.

Esse exemplo é muito simples, mas se você combinar a poderosa maneira de desenvolvimento com XML e XSLT, obrigatório no Ajax e mais do que demonstrado no livro *Dominando Ajax*, todo o conteúdo visto a seguir poderá ser dinâmico.

O Lado Servidor

Para esse exemplo, você utilizará novamente PHP. A classe **Display** seleciona os dados necessários do banco e traz pra você o pedido feito pelo ISBN do livro:

display.class.php

```
<?php

  class Display
  {
   private $conexao;
   // construtor
   function __construct()
   {
    // cria a conexão com o MySQL
    $this->conexao = mysqli_connect("localhost", "edson",
                                    "integrator","livraria");

   }
   // destrutor, fecha a conexão com o banco de dados
   function __destruct()
   {
    mysqli_close($this->conexao);
   }
   // gera o resultado do isbn do livro
   public function gerar($isbn)
   {
    // cria a query que resultará nos dados
    $queryStr = "SELECT descricao
          FROM livros
          WHERE
           isbn='$isbn'";
   // executa a query
   $result = mysqli_query($this->conexao,$queryStr);

   if(@mysqli_num_rows($result)>0){
```

```
      $row = mysqli_fetch_object($result);
        $resultado=stripslashes( $row->descricao );

    }
    return $resultado; //retorna o XML gerado
   }

  // end class
  }
  ?>
```

Muito similar aos exemplos já vistos, você tem a página que será chamada pelo JavaScript:

display.php

```
<?php
  require_once('display.class.php'); //inclui a classe Resultados
  // captura a ação

  $isbn = $_REQUEST['isbn'];

  // cria a instância de Display
  $display = new Display();

  // headers enviados para evitar cache no browser
  header('Expires: Fri, 25 Dec 1980 00:00:00 GMT');
  header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
  header('Cache-Control: no-cache, must-revalidate');
  header('Pragma: no-cache');

  if(!empty($isbn))
   echo $display->gerar($isbn);
  ?>
```

A página **display.php** receberá uma query string chamada **isbn** com o valor transmitido pelo JavaScript e retornará o valor encontrado pelo método **gerar( )** da classe **Display( )**.

O Lado Cliente

O arquivo **display.html** é o responsável por fazer a comunicação com o servidor e trazer as informações para a página:

display.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">
  <head>
  <title>Display de Descrições de Livros</title>
  <style>
  .retrair{
    visibility:hidden;
    display:none;
  }
  .fundo{
    background-color:#663399;
    color:#FFFFFF;
    font-weight:bold;
  }

  </style>
  <script>
  function criaXMLHttpRequest( ) {
    var xmlHttp;
    if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XmlHttp");
    }
    else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
    }
    return xmlHttp;
  }
  function enviar (elemento, enviar) {

        //Prepare a server request
        var xmlHttp=criaXMLHttpRequest();
        var pagina = "display.php?isbn=" + enviar;
        xmlHttp.open("GET", pagina, true);

        //Response function
        xmlHttp.onreadystatechange = function ( ) {
           if (xmlHttp.readyState == 4)
        {
           if (xmlHttp.status == 200)
        {

              var resultado = xmlHttp.responseText;
              toggle (elemento, resultado);
              }
           }
      }
        xmlHttp.send (null);
```

```
}
function toggle (tag_div, resultado) {
   var elemento = document.getElementById (tag_div);
   var visivel = (elemento.style.visibility == "visible") ? true : false;
  elemento.innerHTML="<em>"+resultado+"</em>";
   elemento.style.visibility = visivel ? "hidden" : "visible";
   elemento.style.display = visivel ? "none" : "inline";
}
</script>
</head>

<body>
<table border=0 width="35%" cellpadding="3">
<tr>
   <td class="fundo" onclick="enviar('eclipse','85-7393-486-7');">
      Dominando Eclipse
   </td>
</tr>
<tr>
   <td bgcolor="#E0E0E0">
      <div id="eclipse" class="retrair">
      </div>
   </td>
</tr>
<tr>
   <td class="fundo" onclick="enviar('netbeans','85-7393-519-7');">
      Dominando NetBeans
   </td>
</tr>
<tr>
   <td bgcolor="#E0E0E0">
      <div id="netbeans" class="retrair">
      </div>
   </td>
</tr>
<tr>
   <td class="fundo" onclick="enviar('calc','85-7393-504-9');">
      OpenOffice.org 2.0 Calc
   </td>
</tr>
<tr>
   <td bgcolor="#E0E0E0">
      <div id="calc" class="retrair">
      </div>
   </td>
</tr>
</table>
</body>
</html>
```

O método **enviar( )** recebe dois parâmetros, o primeiro o ID que deverá mostrar os dados recebidos e o segundo o ISBN do livro para a consulta ao servidor.

Assim que recebido, o método **toggle ( )** é chamado e abre a tag e escreve o texto pego pelo servidor.

O resultado é como mostrado na **Figura 9** a seguir:

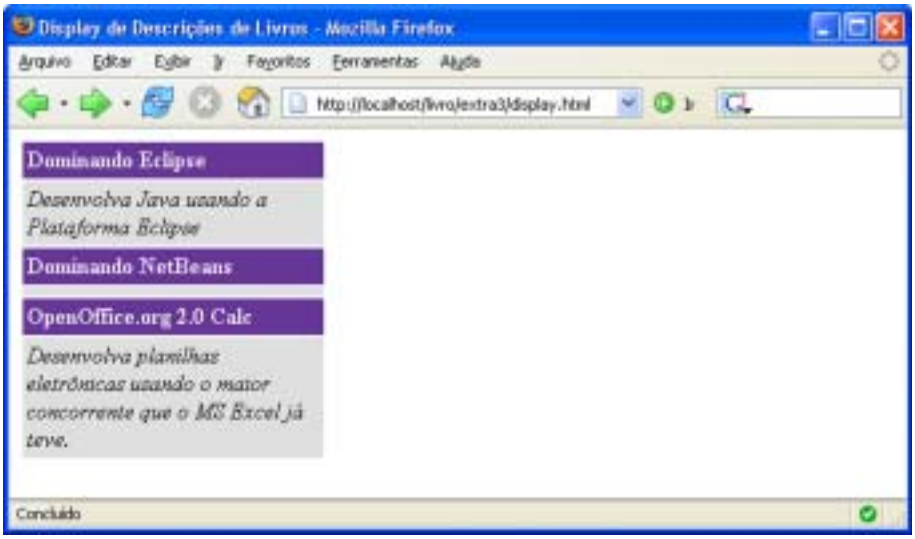

Figura 9

Nas Melhores Livrarias

Desenvolvido a partir de uma larga experiência com o JavaScript e tecnologias de servidor, o autor mostra diversas possibilidades que você pode ter usando o essa linguagem de scripts combinada com CSS, XML e XSLT.

Abordando as melhores práticas usando PHP 5 e Java, você será levado a um universo de infinitas possibilidades, passando pelo básico até os mais complexos exemplos, todos feitos para funcionar tanto no Mozilla Firefox como no Microsoft Internet Explorer.

Entenda como funciona o JavaScript, começando pelo básico e chegando a tão falada orientação a objetos.

Melhore a aparência dos seus documentos usando as famosas CSS's, modelando assim um padrão visual para suas aplicações.

Utilize os resultados vindos do servidor, trabalhando com XML, combinado com XSLT e CSS, possibilitando uma aparência mais agradável e dinâmica.

Entenda como funcionam os envios POST e GET para o servidor usando o Ajax e aprenda como tratar as respostas do servidor, seja em texto simples ou em XML.

No livro você aprenderá a criar um DataGrid simples, com exclusão de múltiplos dados; e um completo, contendo inserção de dados, atualizações e exclusões.

Aprenda a fazer um auto-completar com diversas sugestões, entendendo como ter um time entre o seu aplicativo e o servidor, evitando assim grandes demoras na atualização das suas requisições.

Aprenda a paginar resultados, usando XML combinado com XSLT e um JavaScript totalmente orientado a objetos.

O livro traz ainda nos seus apêndices como instalar e a utilizar, tanto em Linux como em Windows, o banco de dados MySQL, usado nos exemplos com acesso a banco ao longo do livro, o Tomcat 5.5 para suas aplicações Web com Java e o Apache 2 configurado com o PHP 5. Abrange também como integrar o Tomcat ao Apache 2.