Desenvolvimento Web II

Aula 05 - AJAX: Interagindo com o Servidor de Maneira Assícrona

Apresentação

Até agora, aprendemos a desenvolver aplicações Web em Java usando servlets, JSP e JSF, assim como a utilizar o JavaScript para manipular a o conteúdo de páginas HTML do lado do cliente (no navegador), melhorando assim a interatividade do usuário. Vamos agora aprender uma nova maneira de incrementar a interatividade e usabilidade de sua página Web de forma ainda mais dinâmica utilizando uma tecnologia chamada AJAX.

Bons estudos!

Vídeo 01 - Apresentação

Objetivos

- Reconhecer o modelo de programação usando AJAX;
- Aplicar o objeto XMLHttpRequest para interagir com o servidor;
- Entender e desenvolver a aplicação do lado do servidor que produz informações que são consumidas por requisições AJAX do lado do cliente;
- Entender o uso de AJAX através do desenvolvimento de exemplos mais complexos.

Introdução

Em meados de 1990, o cientista da computação Sir Tim Berners-Lee, juntamente com outros cientistas do CERN (http://public.web.cern.ch/public/), criaram a proposta do que viria a se tornar a web. A proposta inicial era bem mais simples do que o encontrado hoje em dia. Basicamente, **a ideia era disponibilizar informações estáticas através da internet, ou seja, basicamente textos, imagens e *links* sem qualquer interação entre o usuário e o servidor.** Divergindo bastante do que foi originalmente pensado pelo grupo de cientistas, a web hoje hospeda muito mais que simples páginas, hospeda sistemas e aplicativos cada vez mais ricos e complexos.

Modelo típico

Devido a uma série de facilidades, como a inexistência de instalação/atualização nos computadores dos seus respectivos usuários, os sistemas web têm se tornando cada vez mais uma tendência. Por não ter sido pensada desde o início com tal finalidade, a web precisou passar por um grande processo evolutivo. Até poucos anos atrás, a cada requisição feita pelo navegador, a resposta era uma página HTML completa a ser carregada no lugar da página que até então estava sendo exibida. Esse modelo, apresentado na Figura 1, estava longe do ideal pois, sabe-se que na grande maioria das interações apenas uma pequena parte da página em exibição precisaria ser modificada.

Figura 01 - Modelo tradicional

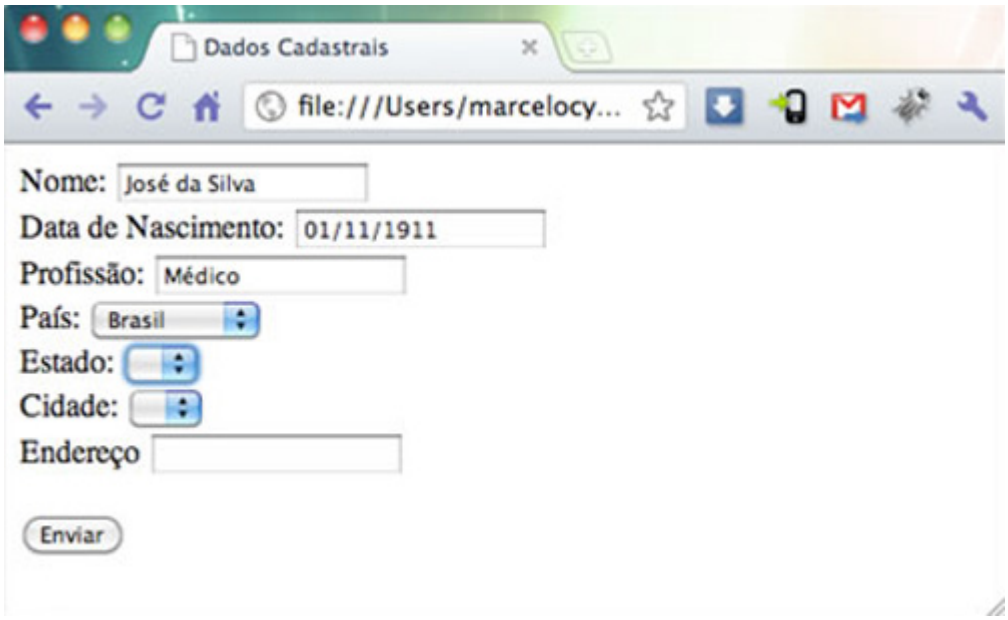
Fonte: Autoria Própria

Vídeo 02 - Ajax Comunicação

Por exemplo

No formulário cadastral da Figura 2, após o usuário selecionar o país, bastaria recuperar do servidor a lista de estados do país selecionado e em seguida popular o campo Estado em vez de carregar uma nova página inteira. O mesmo se repete para os campos Estado/Cidade.

Figura 02 - Após selecionar o país, bastaria atualizar o campo Estado



Principais problemas do modelo tradicional

1. Em grande parte das interações entre cliente e servidor, apenas uma pequena parte da interface precisa ser modificada (ao invés da página inteira).

2. Pelo fato de sempre receber como resposta uma página inteira, é gerado um tráfego de dados maior que o necessário.

3. Do momento que o navegador dispara uma requisição até o momento que ele recupere toda a resposta e carregue a nova página, o usuário fica impossibilitado de interagir com o navegador.

4. Pelos pontos apresentados acima, o modelo típico termina também por demandar mais recursos do servidor.

AJAX

Ao conjunto de tecnologias que combinadas têm o potencial de resolver as limitações apresentadas acima foi dado o nome "AJAX" (acrônimo para Asynchronous JavaScript And XML, ou seja, JavaScript assíncrono e XML). Apesar do nome sugestivo, AJAX não é limitado à comunicação assíncrona nem a dados em XML. Como veremos a seguir, tanto é possível utilizar comunicação síncrona como também outros formatos de dados além de XML. Com o amadurecimento das tecnologias que compõem o AJAX, podemos ver os sistemas web atingirem um grau de experiência oferecida ao usuário comparável ao das aplicações nativas.

Vídeo 03 - AJAX Introdução

Principais vantagens e diferenças do modelo AJAX

1. O navegador passa a não ser mais apenas um “terminal burro”, divide a responsabilidade com o servidor. Criando assim um cenário duplamente favorável: ao mesmo tempo em que o servidor é menos exigido, o cliente também ganha uma aplicação mais rica e interativa.

2. O usuário passa a não mais precisar ficar esperando entre cada interação com o servidor: através do AJAX, o navegador ganhou meios de se comunicar de forma assíncrona. Enquanto o navegador e o servidor

trocam dados em segundo plano (em background), o usuário pode continuar interagindo com a página. Por exemplo, na Figura 2, enquanto o navegador estiver buscando a lista de estados do país selecionado, o usuário já pode ir preenchendo o campo Endereço.

3. Existe separação entre o código da interface gráfica e os dados. Logo, existem requisições distintas para cada finalidade: nem toda resposta do servidor é uma página HTML. Como veremos no decorrer desse documento, a resposta pode conter apenas dados (ex: lista de estados de um país) em XML, JSON ou ainda texto puro.

4. Devido ao volume dos dados trocados em cada interação com o servidor ser bem menor no modelo AJAX, o novo modelo apresenta uma performance bastante superior ao modelo típico.

Exemplos

Antes de apresentar os detalhes da programação AJAX serão apresentados dois exemplos simples que visam tornar claro as diferenças e vantagens do novo modelo proposto. O primeiro exemplo utiliza os meios tradicionais: dados sendo submetidos para o servidor utilizando um formulário. O segundo é basicamente a reconstrução de parte do primeiro utilizando AJAX.

Exemplo 1

Como pode ser observado nas Figuras 3 e 4, esse exemplo é composto por uma página com apenas duas listas HTML. A primeira, contendo categorias pré-definidas de livros e a segunda contendo os respectivos livros da categoria selecionada.

Principais vantagens e diferenças do modelo AJAX

1. O navegador passa a não ser mais apenas um “terminal burro”, divide a responsabilidade com o servidor. Criando assim um cenário duplamente favorável: ao mesmo tempo em que o servidor é menos exigido, o cliente também ganha uma aplicação mais rica e interativa.

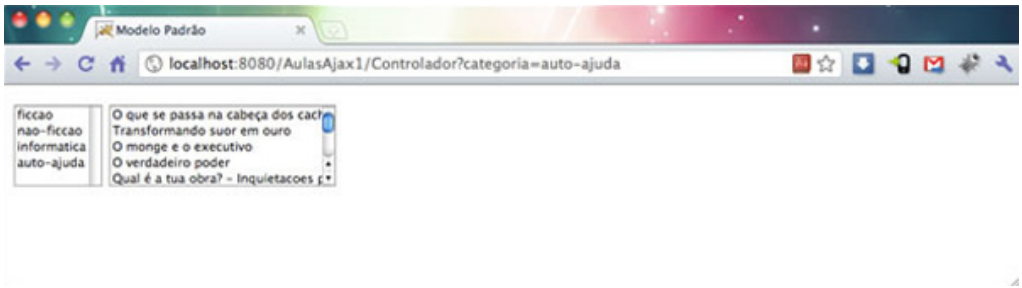
2. O usuário passa a não mais precisar ficar esperando entre cada interação com o servidor: através do AJAX, o navegador ganhou meios de se comunicar de forma assíncrona. Enquanto o navegador e o servidor trocam dados em segundo plano (em background), o usuário pode continuar interagindo com a página. Por exemplo, na Figura 2, enquanto o navegador estiver buscando a lista de estados do país selecionado, o usuário já pode ir preenchendo o campo Endereço.

3. Existe separação entre o código da interface gráfica e os dados. Logo, existem requisições distintas para cada finalidade: nem toda resposta do servidor é uma página HTML. Como veremos no decorrer desse documento, a resposta pode conter apenas dados (ex: lista de estados de um país) em XML, JSON ou ainda texto puro.

4. Devido ao volume dos dados trocados em cada interação com o servidor ser bem menor no modelo AJAX, o novo modelo apresenta uma performance bastante superior ao modelo típico.

Figura 03 - Antes de selecionar uma categoria.

Figura 04 - Após selecionar uma categoria.



A seguir veremos um exemplo de requisição AJAX envolvendo Livros e Categorias, onde são trocados dados entre o navegador e o servidor desde a primeira requisição para obter a página da Figura 3 até o segundo momento após o usuário clicar em uma categoria e disparar uma nova requisição para recuperar os respectivos livros (Figura 4).

Requisição (para obter a página da **Figura 3**):

A seguinte requisição é feita no servidor (ex: digitando-se a URL no navegador) para acessar a página inicial:

```
http://localhost:8080/AulasAjax1/Controlador
```

A resposta retornada é a página HTML descrita na Listagem 1.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>Modelo Padrão</title>
</head>
<body>
   <form action="/AulasAjax1/Controlador">
     <p>
     <select name="categoria" size="5" onchange="submit()">
       <option value="ficcao">ficcao</option>
       <option value="nao-ficcao">nao-ficcao</option>
       <option value="informatica">informatica</option>
       <option value="auto-ajuda">auto-ajuda</option>
     </select>
     <select name="livro" size="5" style="width: 200px;">
     </select>
     </p>
   </form>
</body>
</html>
```

Listagem 1 - Primeira página HTML retornada do exemplo 1.

Requisição (para obter a página da **Figura 4**):

Ao clicar em uma das categorias listadas, a seguinte requisição é feita no servidor:

```
http://localhost:8080/AulasAjax1/Controlador?categoria=auto-ajuda
```

Resposta:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>Modelo Padrão</title>
</head>
<body>
<form action="/AulasAjax1/Controlador">
   <p>
   <select name="categoria" size="5" onchange="submit()">
      <option value="ficcao">ficcao</option>
      <option value="nao-ficcao">nao-ficcao</option>
      <option value="informatica">informatica</option>
      <option value="auto-ajuda">auto-ajuda</option>
   </select>
   <select name="livro" size="5" style="width: 200px;">
      <option value="O que se passa na cabeÁa dos cachorros">O que
      se passa na cabeÁa dos cachorros</option>
      <option value="Transformando suor em ouro">Transformando suor
      em ouro</option>
      <option value="O monge e o executivo">O monge e o executivo</option>
      <option value="O verdadeiro poder">O verdadeiro poder</option>
      <option value="Qual È a tua obra? - Inquietacoes propositivas">Qual
      È a tua obra? - Inquietacoes propositivas</option>
      <option value="Fora de serie- Outliers">Fora de serie-
      Outliers</option>
     <option value="O futuro hoje">O futuro hoje</option>
      <option value="PMBOK - Guia do conjunto de conhecimentos">PMBOK
      - Guia do conjunto de conhecimentos</option>
      <option value="Os 7 h·bitos das pessoas altamente eficazes">Os
      7 h·bitos das pessoas altamente eficazes</option>
      <option value="Faça como Steve Jobs">Faça como Steve Jobs</option>
   </select>
   </p>
</form>
</body>
</html>
```

Listagem 2 - Segunda página HTML retornada do exemplo 1.

Notas

1. Observe no exemplo acima que, mesmo já possuindo praticamente todo código da interface gráfica (faltava apenas o que seria preenchido na segunda lista), a resposta do servidor foi uma nova página completa, repetindo quase toda informação da página inicial. Excetuando-se o trecho das linhas 17 a 34 (<select> preenchido), a primeira e a segunda resposta são idênticas. Tal fato ocorre, pois, neste modelo não existe separação entre o código da interface e os dados.
2. Além de gerar um tráfego de dados maior que o necessário e uma espera maior para o usuário voltar a poder interagir com a página, o fato de o navegador carregar uma nova página causa um efeito visual ruim de recarregamento de tela (esse efeito é ainda mais perceptível em páginas mais complexas).

Atividade 01

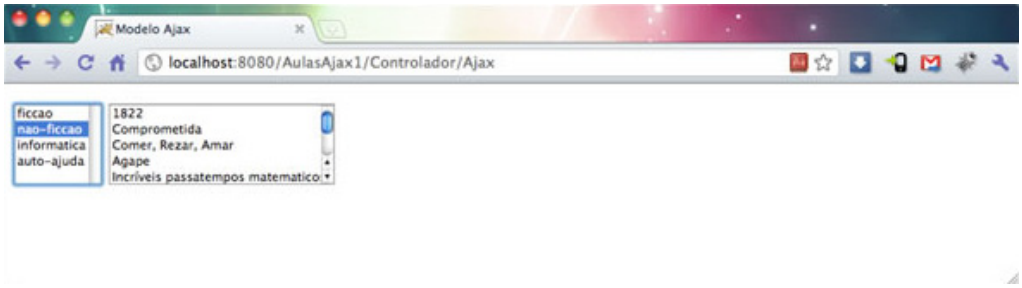
1. Desenvolva um Servlet de nome Controlador que implemente o comportamento do exemplo mostrado.

Exemplo 2

Este exemplo é uma reconstrução do primeiro exemplo utilizando AJAX para recuperar os livros da categoria selecionada. O objetivo deste exemplo é apenas dar uma visão geral de como o AJAX funciona e quais as diferenças em relação ao modelo tradicional de programação Web. Cada um dos elementos expostos será explicado posteriormente, à medida que o modelo de programação AJAX for apresentado ao longo desta aula.

Figura 05 - Antes de selecionar uma categoria.

Figura 06 - Após selecionar uma categoria.



Observando as Figuras 5 e 6, podemos ter a falsa impressão que não existem diferenças visuais entre os dois modelos. No entanto, ao visualizar os exemplos em ação (rodando no browser), notaremos as diferenças: no exemplo que utiliza Ajax, a página não é recarregada. Apenas a lista de livros é atualizada, como acontece em aplicações que rodam no desktop. Como consequência da página não ser recarregada, podemos ver na segunda figura deste exemplo que, mesmo sem nenhum tratamento, o estado da página foi mantido (ex.: a categoria selecionada pelo usuário continua selecionada mesmo após carregar os respectivos livros).

De maneira análoga ao primeiro exemplo, seguem os dados da comunicação entre o navegador e o servidor.

Requisição enviada ao servidor para obter a página da **Figura 5:**

```
http://localhost:8080/AulasAjax1/Controlador/Ajax
```

A resposta do servidor é apresentada na Listagem 3.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
   <title>Modelo Ajax</title>
</head>
<body>

   <script type="text/javascript">
   <!--
   function send() {

     var url = "/AulasAjax1/Controlador/Ajax/Categoria";
     var ajaxReq = new XMLHttpRequest();
     url += "?categoria=" + document.getElementById("campoCategoria").value;

     ajaxReq.onreadystatechange = function() {
      if (ajaxReq.readyState == 4 && ajaxReq.status == 200) {
        var text = ajaxReq.responseText;
        var livros = text.split(';');
        var campoLivro = document.getElementById("campoLivro");
        campoLivro.length = 0;
        for(var i=0; i<livros.length; i++){
          if(livros[i].length>0){
           campoLivro.options[campoLivro.length] = new Option(livros[i], livros[i]);
          }
        }
      }
     };

     ajaxReq.open("GET", url, true);
     ajaxReq.send();
   }
   -->
   </script>

   <p>
   <select id="campoCategoria" name="categoria" size="5" onchange="send()">
     <option value="ficcao">ficcao</option>
     <option value="nao-ficcao">nao-ficcao</option>
     <option value="informatica">informatica</option>
     <option value="auto-ajuda">auto-ajuda</option>
   </select>
   <select id="campoLivro" name="livro" size="5" style="width: 200px;">
   </select>
   </p>

</body>
</html>
```

Listagem 3 - Primeira página HTML retornada do exemplo 2.

Uma vez selecionada uma categoria, a seguinte requisição é enviada ao servidor para obter a página da Figura 6:

```
http://localhost:8080/AulasAjax1/Controlador/Ajax/Categoria?categoria=ficcao
```

A resposta dessa requisição, porém, não é mais um HTML, mas apenas o que deve ser alterado no HTML, ou seja, a lista de livros:

A cabana;
A Ilha sob o mar;
Querido John;
Fallen;
A ultima música;
Caminhos da lei;
Os homens que não amavam as mulheres;
A ilusão da alma;
Amante desperto;
O castelo dos Pirineus.

Notas

1. A segunda resposta contém apenas dados, nada relativo à interface foi enviado novamente.
2. Como dito anteriormente, AJAX não necessariamente requer XML (ver a segunda resposta do segundo exemplo).

Antes de você implementar o Exemplo 2, vamos entender o código do AJAX que foi apresentado.

Ajax Passo a Passo

O ponto chave para enviar e receber informações do servidor de forma assíncrona e sem precisar recarregar a página inteira é o objeto XMLHttpRequest definido pelo World Wide Web Consortium (W3C) em <http://www.w3.org/TR/XMLHttpRequest/>. Esse objeto tem uma API simples que será o foco desta seção.

Para melhor entendimento, após ler cada item abaixo, volte ao código do segundo exemplo e veja onde o mesmo se encontra.

A classe XMLHttpRequest

Logo quando o AJAX começou a ser utilizado, não existia uma API padrão para enviar e receber dados do servidor de forma assíncrona. Visando padronizar e garantir a portabilidade das aplicações web entre os vários navegadores existentes, a W3C definiu uma classe chamada XMLHttpRequest. Essa classe está implementada nos principais (se não todos) navegadores comerciais e contempla as necessidades para a conversação assíncrona com o servidor.

Vídeo 04 - AJAX Introdução

Instanciando o objeto XMLHttpRequest

Para obter uma instância do objeto XMLHttpRequest, basta chamar seu construtor:

```
1 var ajaxReq = new XMLHttpRequest();
```

Tratando as respostas do servidor

Na grande maioria dos casos, é desejável tratar as respostas das requisições enviadas ao servidor (uma exceção seria o envio de um aviso ao servidor, cujo recebimento e processamento não influencia no lado cliente). Para tal, o primeiro passo é associar uma função ao atributo “onreadystatechange” do objeto XMLHttpRequest criado.

Ex.:

```
ajaxReq.onreadystatechange = function() {
   ...
};
```

Essa função é especificada para tratar as respostas do servidor e pode ser chamada para diversas situações distintas. Por essa razão, devemos escrever o código de tratamento de modo que nosso tratamento seja executado apenas para o cenário que desejamos. Basicamente, os possíveis cenários se restringem a combinações das etapas de envio e do *status* da resposta:

```
if (ajaxReq.readyState == 4 && ajaxReq.status == 200) {
   ...
}
```

Etapa	Valor	Descrição
UNSET	0	A instância do objeto XMLHttpRequest foi criada.
OPENED	1	O método *open* (será vista logo em seguida) foi chamado com sucesso e já é possível definir cabeçalhos (utilizando a função setRequestHeader) e enviar dados ao servidor.

Etapa	Valor	Descrição
HEADERS_RECEIVED	2	Parte da resposta de uma requisição já foi recebida e todos os cabeçalhos da resposta já estão disponíveis.
LOADING	3	O corpo da resposta está sendo recebida.
DONE	4	Toda a resposta foi recebida ou algo de errado ocorreu.

Tabela 1 - Etapas, valores e significado do atributo readyState da instância do objeto XMLHttpRequest.
Fonte: Autoria Própria

Status da resposta

O status da resposta pode variar entre códigos 1xx, 2xx, 3xx, 4xx e 5xx; no entanto, os únicos códigos que representam sucesso na resposta são os 2xx. Ver mais detalhes em: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Notas

Perceba que no Exemplo 2, na linha 17, especificamos as condições para que o código seja executado:

1. A resposta recebida por completo (**ajaxReq.readyState == 4**) e
2. Status HTTP de sucesso (**ajaxReq.status == 200**)

Abrindo uma conexão com o servidor

Antes de começar a enviar dados, é necessário abrir uma conexão com o servidor através do método *open* do objeto XMLHttpRequest criado:

```
1 ajaxReq.open("GET", url, true);
```

Parâmetro	Descrição
Método HTTP	Método HTTP a ser utilizado. Geralmente GET ou POST. Mais detalhes serão explicados no próximo tópico (Enviando os dados). Para a referência completa: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.
URL	Endereço do serviço a ser utilizado.
Modo de comunicação	Variável booleana indicando o modo de comunicação a ser utilizado: Síncrono (*false*) ou assíncrono (*true*).

Tabela 2 - Descrição dos parâmetros do método open().
Fonte: Autoria Própria

Enviando os dados

Após instanciar o objeto XMLHttpRequest, abrir uma conexão com sucesso e definir uma função para tratar a resposta, resta apenas enviar os dados para o servidor utilizando o método send():

```
1 ajaxReq.send();
```

HTTP GET

Caso o método utilizado para enviar os dados ao servidor (primeiro parâmetro do método open()) tenha sido GET, os dados devem ser passados na própria URL (segundo parâmetro do método open()) como exemplificado a seguir:

```
1 var url = "/AulasAjax1/Controlador/Ajax/Categoria";
2 var ajaxReq = new XMLHttpRequest();
3 url += "?categoria=" + document.getElementById("campoCategoria").value;
4
5 ajaxReq.open("GET", url, true);
6 ajaxReq.send();
```

HTTP POST

Quando o método utilizado para enviar dados ao servidor for o POST, os dados não mais devem ser enviados juntos com a URL, mas no corpo da mensagem. No caso do AJAX, isso é feito passando uma variável contendo os dados a serem enviados como parâmetro do método: send(...DADOS...). Além disso, na linha 30, troca-se o “GET” pelo “POST”. Segue abaixo o exemplo anterior modificado utilizando POST:

```
1 var url = "/AulasAjax1/Controlador/Ajax/Categoria";
2 var ajaxReq = new XMLHttpRequest();
3 var data = "categoria=" + document.getElementById("campoCategoria").value;
4
5 ajaxReq.open("POST", url, true);
6 ajaxReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
7 ajaxReq.send(data);
```

Nota: observe que antes de enviar os dados, o método setRequestHeader() foi chamado. Esse método serve para definir cabeçalhos HTTP que serão enviados ao servidor. O cabeçalho definido no trecho acima informa sobre a formatação dos dados que serão enviados ao servidor.

Atividade 02

---

1. Desenvolva uma nova versão do Servlet Controlador que implementa o comportamento do segundo exemplo mostrado.

Código do servidor

E como fica o código do servidor? O código do servidor é formado pelos seguintes componentes:

1. /AulasAjax1/src/exemplos/ajax/Controlador.java :

Classe java responsável por determinar o fluxo de navegação e colocar determinadas variáveis em memória que serão utilizadas pelas páginas JSP.

2. /AulasAjax1/WebContent/modelo_padrao.jsp:

Página JSP responsável por gerar a página HTML.

Vídeo 05 - Requisição

Para que o controlador funcione você precisa baixar o arquivo: <http://repo1.maven.org/maven2/jstl/jstl/1.2/jstl-1.2.jar> e adicionar na pasta WEB-INF/lib.

Vejamos a seguir cada um desses componentes.

/AulasAjax1/src/exemplos/ajax/Controlador.java:

```java
package exemplos.ajax;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Controlador do sistema
 */
@WebServlet({"/Controlador/Ajax", "/Controlador/Ajax/Categoria", "/Controlador/Ajax/LivroInfo"})
public class Controlador extends HttpServlet {

  private static final String CATEGORIA_FICCAO = "ficcao";
  private static final String CATEGORIA_NAO_FICCAO = "nao-ficcao";
  private static final String CATEGORIA_INFORMATICA = "informatica";
  private static final String CATEGORIA_AUTO_AJUDA = "auto-ajuda";

  private static final String[] CATEGORIAS = { CATEGORIA_FICCAO,
     CATEGORIA_NAO_FICCAO, CATEGORIA_INFORMATICA,
     CATEGORIA_AUTO_AJUDA };

  private static final String[] LIVROS_FICCAO = { "A cabana",
     "A Ilha sob o mar", "Querido John", "Fallen", "A ultima música",
     "Caminhos da lei", "Os homens que não amavam as mulheres",
     "A ilusão da alma", "Amante desperto", "O castelo dos Pirineus" };

  private static final String[] LIVROS_NAO_FICCAO = { "1822", "Comprometida",
     "Comer, Rezar, Amar", "Agape", "Incríveis passatempos matematicos",
     "Brasil - uma historia", "Nosso lar", "O Aleph", "Aforismos" };

  private static final String[] LIVROS_INFORMATICA = {
     "VBA e macros para Microsoft Office Excel 2007",
     "Java como programar", "Apresentação Zen",
     "UML2 - Uma abordagem pratica", "O mítico homem-mês",
     "Revis architecture 2010",
     "JQuery - A biblioteca do programador Javascript",
     "CCENT/CCNA ICND1", "Analise de pontos de função",
     "Formulas e funções com Microsoft Office Excel 2007", };

  private static final String[] LIVROS_AUTO_AJUDA = {
     "O que se passa na cabeça dos cachorros",
     "Transformando suor em ouro", "O monge e o executivo",
     "O verdadeiro poder",
     "Qual é a tua obra? - Inquietacoes propositivas",
     "Fora de serie- Outliers", "O futuro hoje",
     "PMBOK - Guia do conjunto de conhecimentos",
     "Os 7 hábitos das pessoas altamente eficazes",
```

```
52      "Faça como Steve Jobs", };
53
54    private static final long serialVersionUID = 1L;
55
56    /**
57     * Metodo que trata/recebe as requisicoes do tipo POST
58     *
59     * @see javax.servlet.http.HttpServlet#doPost(javax.servlet.http.HttpServletRequest,
60     *      javax.servlet.http.HttpServletResponse)
61     */
62    @Override
63    protected void doPost(HttpServletRequest request,
64        HttpServletResponse response) throws ServletException, IOException {
65
66      doGet(request, response);
67    }
68
69    /**
70     * Metodo que trata/recebe as requisicoes do tipo GET
71     *
72     * @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,
73     *      javax.servlet.http.HttpServletResponse)
74     */
75    @Override
76    protected void doGet(HttpServletRequest request,
77        HttpServletResponse response) throws ServletException, IOException {
78
79      String acaoSelecionada = request.getServletPath();
80      if (acaoSelecionada.equalsIgnoreCase("/Controlador/Ajax")) {
81
82        enviarPaginaPadrao(request, response);
83
84      } else if (acaoSelecionada.equalsIgnoreCase("/Controlador/Ajax/Categoria")) {
85
86        enviarNomeLivros(request, response);
87
88      } else if (acaoSelecionada.equalsIgnoreCase("/Controlador/Ajax/LivroInfo")) {
89
90        enviarInformacoesLivro(request, response);
91
92      }
93    }
94
95    private void enviarPaginaPadrao(HttpServletRequest request,
96          HttpServletResponse response) throws ServletException, IOException {
97
98      salvarCategoriasReq(request);
99      salvarLivrosReq(request);
100
101     request.getRequestDispatcher("/modelo_padrao.jsp").forward(request,
102       response);
```

```
103   }
104
105   private void enviarNomeLivros(HttpServletRequest request,
106       HttpServletResponse response) throws ServletException, IOException {
107
108       String category = request.getParameter("categoria");
109       String[] livros = null;
110       if (category != null) {
111
112         if (category.equals(CATEGORIA_FICCAO)) {
113           livros = LIVROS_FICCAO;
114         } else if (category.equals(CATEGORIA_NAO_FICCAO)) {
115           livros = LIVROS_NAO_FICCAO;
116         } else if (category.equals(CATEGORIA_INFORMATICA)) {
117           livros = LIVROS_INFORMATICA;
118         } else if (category.equals(CATEGORIA_AUTO_AJUDA)) {
119          livros = LIVROS_AUTO_AJUDA;
120         }
121       }
122
123       String resp = "";
124       for (int i = 0; i < livros.length; i++) {
125         resp += livros[i] + ";";
126       }
127
128       response.setContentType("text/html; charset=ISO-8859-1");
129       response.getWriter().write(resp);
130       response.flushBuffer();
131   }
132
133   private void enviarInformacoesLivro(HttpServletRequest request,
134       HttpServletResponse response) throws IOException {
135
136     String livro = request.getParameter("livro");
137     if (livro == null) {
138       livro = "null";
139     }
140
141     response.setContentType("text/html; charset=ISO-8859-1");
142     response.getWriter().write("Essas sao as informacoes sobre o livro " + livro);
143     response.flushBuffer();
144   }
145
146
147
148   private void salvarCategoriasReq(HttpServletRequest request) {
149     request.setAttribute("categorias", CATEGORIAS);
150   }
151
152   private void salvarLivrosReq(HttpServletRequest request) {
153     String category = request.getParameter("categoria");
```

```
154     if (category != null) {
155
156       if (category.equals(CATEGORIA_FICCAO)) {
157         request.setAttribute("livros", LIVROS_FICCAO);
158       } else if (category.equals(CATEGORIA_NAO_FICCAO)) {
159         request.setAttribute("livros", LIVROS_NAO_FICCAO);
160       } else if (category.equals(CATEGORIA_INFORMATICA)) {
161         request.setAttribute("livros", LIVROS_INFORMATICA);
162       } else if (category.equals(CATEGORIA_AUTO_AJUDA)) {
163         request.setAttribute("livros", LIVROS_AUTO_AJUDA);
164       }
165     }
166   }
167 }
```

A tabela a seguir apresenta uma descrição dos principais métodos da classe Controlador.

Método	Descrição
doPost	Recebe as requisições do tipo POST e encaminha para ser tratada pelo método doGet.
doGet	Recebe as requisições do tipo GET e, baseado na URL chamada (usando o request.getServletPath()) definir a acaoSelecionada.
enviarInformacoesLivro	Retorna informações do livro informado através do parâmetro "livro" (utilizado pela página modelo_ajax.jsp).
enviarPaginaPadrao	Redireciona o navegador para a página "/AulasAjax1/WebContent/modelo_padrao.jsp"
salvarCategoriasReq	Salva a variável "categorias" contendo as categorias pré-definidas no escopo da requisição (variável utilizada em todas as páginas para popular a lista de categorias).

Método	Descrição
salvarLivrosReq	Salva a variável "livros" contendo os livros da categoria selecionada no escopo da requisição (variável utilizada pela página modelo_padrao.jsp para popular a lista de livros).
enviarNomeLivros	Retorna os nomes dos livros da categoria selecionada através do parâmetro categoria (utilizado pela página modelo_ajax.jsp).

/AulasAjax1/WebContent/modelo_padrao.jsp:

Veja a seguir o código da página JSP.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Modelo Ajax 2 (POST)</title>
</head>
<body>

<script type="text/javascript">


function buscarLivros() {

  var ajaxReq = new XMLHttpRequest();
  var url = "/AulasAjax1/Controlador/Ajax/Categoria";
  var data = "categoria=" + document.getElementById("campoCategoria").value;

  ajaxReq.onreadystatechange = function() {
    if (ajaxReq.readyState == 4 && ajaxReq.status == 200) {
      var text = ajaxReq.responseText;
      var livros = text.split(';');
      var campoLivro = document.getElementById("campoLivro");
      campoLivro.length = 0;
      for(var i=0; i<livros.length; i++){
        if(livros[i].length>0){
          campoLivro.options[campoLivro.length] = new Option(livros[i], livros[i]);
        }
      }
    }
  };

  ajaxReq.open("POST", url, true);
  ajaxReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  ajaxReq.send(data);
}

function buscarInfoLivro() {

  var ajaxReq = new XMLHttpRequest();
  var url = "/AulasAjax1/Controlador/Ajax/LivroInfo";
  var data = "livro=" + document.getElementById("campoLivro").value;

  ajaxReq.onreadystatechange = function() {
    if (ajaxReq.readyState == 4 && ajaxReq.status == 200) {
      var campoInfoLivro = document.getElementById("campoInfoLivro");
```

```
      campoInfoLivro.value = ajaxReq.responseText;
    }
  };

  ajaxReq.open("POST", url, true);
  ajaxReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  ajaxReq.send(data);
}


</script>

<p>
<select id="campoCategoria" name="categoria" size="5"
onchange="buscarLivros()">
<c:forEach var="c" items="${categorias}">
  <option value="${c}">${c}</option>
</c:forEach>
</select>


<select id="campoLivro" name="livro" size="5" style="width: 200px;"
onchange="buscarInfoLivro()">
</select>

</p>

<textarea id="campoInfoLivro" rows="5" cols="40">
</textarea>


</body>
</html>
```

Para ver a página basta acessar a URL http://localhost:8080/AulasAjax1/Controlador/Ajax e ao clicar em uma categoria você verá a lista de livros dessa categoria.Ao clicar em um livro você verá suas informações. Repare que somente os dados necessários são enviados por Ajax do servidor para o navegador, não sendo necessário recarregar toda a página.

Curiosidade

Aponte seu navegador para o mesmo endereço que a requisição em AJAX faz e veja o que aparece no seu navegador.

Endereço: <http://localhost:8080/AulasAjax1/Controlador/Ajax/Categoria?categoria=ficcao>/p>

Resultado: A cabana;A Ilha sob o mar;Querido John;Fallen;A ultima música;Caminhos da lei;Os homens que não amavam as mulheres;A ilusão da alma;Amante desperto;O castelo dos Pirineus;

Repare que o resultado é somente um texto com o nome dos livros separado por ";", e não um HTML completo. Esse resultado é retornado por AJAX e em utilizando Javascript o navegador consegue separar o nome dos livros e popular as opções do SELECT com os livros visualmente.

Chegamos ao fim da aula de hoje, agora você é capaz de entender e desenvolver um código AJAX para permitir uma melhor interatividade na sua página web, além de torná-la mais rápida e leve para atualizar dados vindos do servidor.

Resumo

Nesta aula, você praticou o desenvolvimento de aplicações AJAX usando Java na parte do cliente e servidor. Como pôde ver, diversas tecnologias foram integradas para resolver o problema: HTML para exibir o conteúdo, JavaScript para permitir tratamentos de eventos e interações com os componentes HTML, AJAX para melhorar a interação com esses componentes e principalmente com o servidor, e Java para executar as operações propriamente ditas.

Autoavaliação

1. Pesquise e liste que outras operações da classe XMLHttpRequest podem ser úteis no dia a dia, indicando sua utilidade.

2. No exemplo apresentado anteriormente, perceba que é verificado o status da resposta HTTP e caso seja sucesso (ajaxReq.status == 200) é que as informações do livro são carregadas. Melhore o código para tratar a situação em que não houve sucesso, como por exemplo, a perda de conexão (ex: retire o cabo de rede do computador para testar).

Referências

ESPECIFICAÇÃO do Objeto XMLHttpRequest. Disponível em: <http://www.w3.org/TR/XMLHttpRequest/>. Acesso em: 30 jul. 2012.

ESPECIFICAÇÃO dos códigos de respostas de um servidor HTTP. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>. Acesso em: 30 jul. 2012.

ESPECIFICAÇÃO do Objeto XMLHttpRequest. Disponível em: <http://www.w3.org/TR/XMLHttpRequest/>. Acesso em: 01 ago. 2012.