Apresentação

Resumo da Ementa

- Hipertexto e Hipermídia.
- Tecnologias de criação de documentos na Internet.
- Armazenamento e recuperação de documentos digitais.
- Diferenças entre aplicações desktop e aplicações web.
- Conceitos de Gerenciamento Eletrônico de Documentos e Workflow.
- Aplicação Prática de Hipermídia no ambiente web.

Bibliografia

- BALDAM, Roquemar; VALLE, Rogério; CAVALCANTI, Marcos. **GED**: Gerenciamento Eletrônico de Documentos. São Paulo: Érica, 2002. 204 p.
- CRUZ, Tadeu. **WORKFLOW II**: A tecnologia que revolucionou processos. Rio de Janeiro: E-papers, 2004. 212 p.
- SANTANA FILHO, Ozeas Vieira; ZARA, Pedro Marcelo. **Microsoft net**: uma visão geral para programadores. São Paulo: Senac, 2002. 124 p.
- SANTOS, Rafael. **Introdução à programação orientada a objetos usando java**. Rio de Janeiro: Elsevier, 2003. 319 p.

Sobre a Disciplina

Dada a enorme variedade de assuntos envolvidos com "Hipermídia", é impossível encontrar um livro que abranja a totalidade da disciplina; sendo ela um "caldeirão" de conhecimentos. A bibliografia aqui mostrada é a "oficial" que está descrita no programa da Universo, mas é opcional, já que todo material necessário será visto em aula e oferecido em www.geocities.com/ponzoneto/mh, salvo alguma publicação específica, caso seja útil, que poderá ser indicada à parte. Pela quantidade de assuntos, o melhor seria pinçar tópicos em uma enciclopédia. Mas, obviamente, não vou sugerir a compra de uma. A opção (da qual extraí vários trechos que usei nos textos das aulas) é a Wikipedia, porém deve-se tomar extremo cuidado com os textos oferecidos. Embora seja uma excelente iniciativa, democrática, gratuita, etc., a verdade é que muitos verbetes têm falhas, falta de informação importante e pouca qualidade. Todos os textos que peguei de lá (alguns em inglês, por não ter o correspondente em português ou estar muito pior) tive de reescrever em parte, pois inexiste na Wikipedia o mesmo rigor de seleção e revisão de texto de uma Enciclopédia Britânica, por exemplo.

Outro detalhe é quanto ao termo "Modelagem". Ele não deve ser entendido como modelagem de desenvolvimento (análise, projeto, padrões, etc.), pois isso é assunto de outras disciplinas. Tampouco deve ser interpretado como relacionado ao *design* de documentos hipermídia (como páginas *web*), porque para isso existem profissionais de desenho e áreas afins. Claro que, no decorrer do curso, esses pontos serão mencionados, mas, no nosso caso, "modelagem" tem relação com os vários modelos possíveis de soluções de hipermídia, particularmente no ambiente *web*. Além dos conceitos básicos, são objetivos da disciplina: mostrar a aplicação de hipermídia na *web*, entender as tecnologias que permitem criar documentos/programas na Internet e verificar as diferenças entre aplicações *desktop* e aplicações *web*.

Disciplinas Correlatas

- Linguagens de Programação
- Ferramentas de Multimídia e Internet
- Sistemas Distribuídos

JSP e Servlets

No curso, vocês já conheceram programação estruturada tradicional em modo texto (Pascal, C), viram programação *desktop* orientada a objeto com interface gráfica e acesso a bancos de dados (Delphi, Java) e, especificamente com relação à *web client-side*, estudaram HTML com Javascript e *Applets* Java. Dos tópicos mais significativos em relação à programação, o que faltava introduzir era o lado *web server-side*, o que faremos aqui com JSP, seguindo a linha de Java iniciada em LP5. Com isso, forma-se um panorama suficientemente amplo sobre programação.

Introdução

JSP (*JavaServer Pages*) e *Servlets* são tecnologias para desenvolvimento *web* que fazem parte da especificação EE (Enterprise Edition, antes chamada J2EE). JSP é uma forma de embutir Java em páginas *web*, enquanto *Servlets* são classes Java residentes em um servidor, executadas no próprio servidor (ao contrário dos *Applets* que também residem no servidor, mas são enviados para rodar na máquina cliente, como vimos em LP6). Para uma página JSP (ou de qualquer linguagem de *script server-side*) funcionar é preciso que exista um servidor *web*[1] que "entenda" a linguagem e consiga compilar e/ou interpretar a página e enviar ao usuário o seu resultado. No nosso caso, o servidor *web* será o Tomcat embutido no NetBeans.

Vamos criar nossa primeira página JSP. Tomemos como exemplo inicial de página um "alô mundo" em puro HTML:

```
<html>
<body>
  Alô, mundo!
</body>
</html>
```

Pois bem, vamos entrar no NetBeans 5.5[2]. Feche qualquer projeto aberto e crie um novo projeto (Arquivo-Novo Projeto-Web-Aplicação Web). Na tela a seguir, mude apenas o nome do projeto (chame de Alo) e o seu diretório (sugiro c:\Java – o caminho *default* é muito longo) e finalize.

Surgirá um arquivo chamado index.jsp, que é o nome padrão inicial de qualquer projeto JSP (assim não é necessário indicar esse nome na barra de endereços do navegador, apenas o caminho). Apague todo o conteúdo que já vem como padrão de exemplo - pois vamos começar do zero - e escreva exatamente o HTML visto acima. Mande rodar.

Pronto. Aí está sua primeira página de JSP[3]! *Peraí*... Isso não é puro HTML?!? É, mas o HTML sempre faz parte de uma página JSP (ou outras linguagens *web*). O HTML é a linguagem entendida pelo *browser*, então é óbvio que ele figure em páginas JSP, PHP, ASP, etc. É claro que, normalmente, não tem muito sentido uma página JSP que só contenha HTML, mas nem por isso deixa de ser JSP. Quer a confirmação? Então clique com o botão direito do mouse no nome da página index.jsp na guia Projetos e escolha "Exibir Servlet". Agora se convenceu, não é? Olha o Java aí!

Mas o que aconteceu? De onde veio esse código em Java? O fato é que sempre que um JSP é rodado, ele é primeiro traduzido para um *Servlet* e depois se executa o *Servlet*. Para quê? Para que, da próxima vez que se chamar esse JSP, a execução seja mais rápida. Lembre-se lá dos primórdios da computação: compilado é mais rápido que interpretado. Claro que a primeira vez vai demorar mais, mas,

---

[1] Falaremos mais de servidores *web* posteriormente.

[2] Outras versões de NetBeans talvez funcionem de forma diferente: na 5.5 a opção de desenvolvimento *web* pode estar instalada por default, mas em versões posteriores pode precisar ser carregada como *plug-in* adicional no item "Ferramentas".

[3] Se tiver dado erro é porque provavelmente não foi especificado um navegador para exibir páginas. Vá até "Ferramentas-Opções" e escolha o "Navegador Web" desejado.

daí pra frente, se ganha tempo. É uma grande vantagem sobre algumas outras linguagens *web* que não têm essa possibilidade.

Dê uma olhada lá do meio para o fim do *Servlet*. Veja as instruções out.write(...). Esse objeto (out) é predefinido nos *Servlets* e seu método write é que envia a resposta dinamicamente. Ou seja: constrói a página HTML que é enviada ao navegador do usuário. Se você olhar o código fonte no navegador do cliente não vai ver nada de Java, apenas a resposta em HTML gerada com o out.write(...).

Uma pergunta que pode surgir é: posso programar *Servlets* direto, na mão? Pode e eventualmente será desejável, mas tem dois problemas: fazer um *Servlet* é mais complicado que fazer uma página JSP e, além disso, chamá-lo direto pode resultar em problemas de compatibilidade entre *browsers*. A criação de *Servlets* isolados não é tratada nesta disciplina.

Injetando Código Java

Vamos prosseguir, colocando um pouquinho de Java no código.

```
<html>
<body>
  <% out.write("Alô, mundo!"); %>
</body>
</html>
```

Rode. Qual o resultado? O mesmo, só que agora colocamos algo de Java embutido na página. Repare nos símbolos <% e %>. É dentro deles que está o Java (no caso dando um out.write como vimos dentro do *Servlet*). Esses blocos de programação são conhecidos como "*scriptlets*". Vamos adicionar algum processamento mais significativo:

```
<html>
<body>
  <%
    for (int i=0;i<10;i++) out.write("Alô, mundo!<br>");
  %>
</body>
</html>
```

Rode e veja que há um processamento realmente em andamento. Isso significa que qualquer programa de cálculo fixo, por exemplo, que você tenha feito em outras linguagens (sem entradas do usuário ainda) já pode ser "traduzido" para JSP e ficar disponível na *web*.

Exercício 1: crie uma página JSP que dê saída com 7 vezes o seu nome em um *loop*, um nome por linha e usando <*font size* = 1 a 7.
Exercício 2: idem, porém usando de <*h1*> até <*h6*>.

Uma outra forma de embutir código é dar saída de expressões diretamente com <%= e %>. Veja o exemplo abaixo:

```
<html>
<body>
  2 + 2 = <%=2+2%>
</body>
</html>
```

Também poderíamos fazer algo assim:

```
<html>
<body>
  <% int i=2+2; %>
```

```
  2 + 2 = <%=i%>
</body>
</html>
```

Repare que a variável criada (i) continuou disponível no bloco seguinte, porque é gerado apenas um *Servlet* por JSP e não um para cada trecho.

Exercício 3: refaça os exercícios 2 e 3 usando codificação sem o objeto *out*.

Querendo comentar algum trecho, basta colocá-lo entre <%-- e --%> (se estiver dentro do código Java, pode-se usar o padrão da linguagem: /* e */ ou // após uma linha).

Exercício 4: gere uma tabela com a tabuada dinamicamente. Use a tag <table> do HTML inserindo comandos Java (for, no caso) dentro da estrutura de linhas e células. Resolva de duas maneiras diferentes:

1. usando out.write(expressão) para escrever tudo diretamente;
2. usando <%= expressão %> para enxertar os valores na tabela, misturado no código HTML.

São duas formas distintas de construção de páginas JSP que funcionam de forma semelhante, porém a segunda (sem os *out.write*) permite separar melhor o desenho da página (que pode ser feito por um *designer*) e a lógica Java (que pode ser inserida posteriormente por um desenvolvedor). O resultado, de qualquer forma, deve se parecer com:

TABUADA

x	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81