Módulo 6

Programação WEB

Lição 6

Páginas JSP Avançadas

Versão 1.0 - Nov/2007

Autor
Daniel Villafuerte

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Angelo de Oliveira
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Sergio Terzella
Thiago Magela Rodrigues Dias
Vanessa dos Santos Almeida
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Em lições anteriores, aprendemos sobre a sintaxe básica das páginas JSP: como utilizar *scriptlets*, expressões, *Javabeans* e *tags* JSP pré-definidas para fornecer dados dinâmicos junto a elementos estáticos. Também estudamos como utilizar diretrizes para manipulação de páginas através de instruções do servidor. Vimos ainda que as páginas JSP servem como esboço, ou seja, papel preliminar dentro de uma aplicação WEB – atuam como camada de apresentação.

Outro ponto abordado em lições anteriores foram páginas JSP que produzem conteúdo dinâmico enquanto nos preocupamos apenas com a programação de elementos. Esta lição dirige-se aos responsáveis pela camada de apresentação que, não necessariamente, precisam ter experiência em Java ou em qualquer outra linguagem. A utilização de formato baseado em texto reduziu muito as linhas de código dos designers. O conteúdo das páginas JSP tornou-se muito semelhante ao padrão HTML. A lógica é processada fora das páginas JSP em classes *Servlets* que retornam os resultados dos *JavaBeans* reduzindo as linhas de código. Até agora a utilização de código Java (com *scriptlets*) dentro das páginas JSP não era indicada. Dessa forma, algumas funcionalidades não poderiam ser executadas pelos elementos de páginas JSP, como a iteração de uma lista e a ramificação da lógica (instruções *if-else*).

Nesta lição, discutiremos dois elementos de páginas JSP que reduzirão a necessidade incluir *scriptlets* e expressões em páginas JSP. Apresentaremos também um método alternativo de executar a funcionalidades que havíamos delegado inicialmente aos *scriptlets* e às expressões em um formato que é mais simples e mais acessível para aqueles com pouco conhecimento de programação.

Ao final desta lição, o estudante será capaz de:

- Compreender a linguagem de expressão das páginas JSP
- Utilizar a biblioteca JSTL

2. Linguagem de expressão das páginas JSP

O primeiro dos dois elementos de JSP que exploraremos nesta lição é a linguagem de expressão de JSP (EL). Esta linguagem de expressão foi introduzida com a especificação de JSP 2.0 e fornece uma sintaxe simples e limpa para a escrita de expressões que executam a lógica ou o acesso simples a valores.

A utilidade da linguagem de expressão e JSP será melhor compreendida no exemplo a seguir. Considerando os métodos discutidos na lição anterior e se quiséssemos recuperar a propriedade de um *JavaBean*, poderíamos utilizar:

```
<% String nome = user.getSobrenome(); %>
```

ou

```
<jsp:getProperty nome="usuario" property="sobrenome"/>
```

Usando o primeiro método, os colaboradores são expostos às construções de programação Java: para recuperar um atributo de um *JavaBean*, os colaboradores têm que empregar métodos *getter* de um *JavaBean*. O colaborador necessita também estar ciente do tipo de propriedade Java. O segundo método é mais neutro: o acesso à propriedade do *JavaBean* é feita através de *tags* que são similares às *tags* de HTML. Entretanto, esta forma de recuperar a propriedade de um *JavaBean* é longa e trabalhosa. Além disso, este método apresenta sua saída diretamente ao browser do cliente; nenhuma manipulação pode ser executada no valor recuperado.

Usando EL, a propriedade *JavaBean* pode ser acessada com:

```
${usuario.sobreNome}
```

Na construção acima o desenvolvedor não precisa conhecer sintaxe Java, pois é curta e objetiva. A construção contém somente o atributo e a propriedade a ser recuperada e pode ser usada igualmente para a saída, indicando diretamente ao usuário o valor.

2.1. Sintaxe EL

2.1.1. Literais

Linguagem de expressões foi desenvolvida para ter a sintaxe simples. Basicamente, uma construção EL pode ser literal ou uma expressão incluída entre ${ e }.

EL suporta os seguintes tipos de literais:

- Boolean
- Long
- Float
- String
- null

Atributos EL são tratados como se fossem código Java. Por exemplo, valores lógicos podem ser *true* ou *false* (verdadeiro ou falso), Strings precisam estar entre aspas duplas (") ou aspas simples ('), e o valor null define um valor inexistente.

2.1.2. Operadores

São compostos pelos operadores básicos (**+**, **-**, *****, **/**, **%**), os lógicos (**&&**, **||**, **!**), e os de comparação (**==**, **!=**, **<**, **<=**, **>**, **>=**).

2.1.3. Palavras reservadas

EL também define as palavras reservadas que imitam a funcionalidade de alguns dos operadores: and (&&), eq (==), gt (>), ge (>=), or (||), not (!), ne (!=), lt (<), le (<=), div (/) e mod (%).

Outras palavras reservadas disponíveis:

- true – corresponde ao valor do atributo lógico.

- false - corresponde ao valor da atributo lógico.
- instanceof – similar à palavra reservada em Java.
- null – similar à palavra reservada em Java. Determina um valor nulo.
- empty – pode ser usado em diversos casos. Por exemplo, quando se declara um atributo do tipo String e determina que não terá valor, ou até mesmo um vetor ou uma instância.

Abaixo, exemplos de expressões EL.

${'JEDI'} – palavra 'JEDI' sendo atribuída a uma String
${5 + 37} – retorna o valor 42
${ (10 % 5) == 2} – retorna verdadeiro
${empty ''} - retorna verdadeiro

2.2. Acessando atributos e propriedades

Apesar de avaliar expressões literais simples ser útil, EL mostra sua importância ao acessar e processar atributos e propriedades em escopos diferentes.

O acesso ao atributo é simples com EL (simplesmente referenciada pelo nome). As propriedades, os métodos, e as disposições do *JavaBean* podem ser acessadas usando o nome do atributo "." notação.

Exemplo:

```
${usuario.sobreNome}
```

Este acesso ao *JavaBean* pode ser referenciado pelo nome do usuário e recupera o valor de sua propriedade (sobreNome). Note que o escopo do *JavaBean* não importa. EL executa a pesquisa, verificando no escopo da página, do pedido, da sessão, e da aplicação para ver se há um *JavaBean* com o nome especificado. Se tentarmos acessar o nome de um *JavaBean* que não exista dentro de nenhum escopo, vazio será retornado.

Os métodos/propriedades são acessados da mesma maneira. Se quisermos recuperar o comprimento do sobrenome de um usuário, nós podemos primeiramente recuperar a propriedade do sobrenome chamando então o método .length, como no exemplo abaixo:

```
${usuario.sobreNome.length}
```

2.3. Objetos implícitos EL

Não obstante a pesquisa automática de nomes facilitar a codificação, especificar as variáveis de espaço explicitamente faz com que a construção seja fácil de entender para um futuro desenvolvedor. EL disponibiliza diversos objetos implícitos que representam um mapa dos objetos dentro dos diferentes escopos, como a seguir.

- pageScope
- requestScope
- sessionScope
- applicationScope

Por exemplo, se nosso objeto 'usuario' foi localizado dentro do escopo da sessão:

```
${sessionScope.usuario.sobreNome}
```

Além do exemplo acima, EL define também os seguintes objetos implícitos:

- **param** – representa um mapa de parâmetros da requisição com nomes e valores. Por exemplo, para invocar o parâmetro ${param.loginNome} seria equivalente a instrução request.getParameter("loginNome"). Os nomes armazenados aqui apontam para um único valor em tipo String.
- **paramValues** – um mapa que conecta os argumentos dos nomes a vetores de Strings que representa os valores para o nome. Invocar ${paramValues.escolhas} é equivalente a instrução request.getParameterValues("escolhas").
- **header** – um mapa que representa os cabeçalhos disponíveis de uma determinada

requisição. Seus índices são similares àqueles recuperados chamando o método do getHeader de um objeto *ServletRequest*.

- **headerValues** - um mapa que representa os cabeçalhos disponíveis de uma determinada requisição. Seus índices são similares àqueles recuperados chamados pelo método *getHeaders* de um objeto *ServletRequest*.
- **cookies** – retorna os *cookies* disponíveis em uma requisição. Isto é similar a invocar o método dos *getCookies* de um objeto de *HttpServletRequest*.

2.4. A notação []

Com exceção da notação ".", EL também fornece a notação "[]" em variáveis, em métodos e em acesso a vetores. Em muitas maneiras as duas notações são similares. Por exemplo, ${usuario.sobreNome} é o mesmo que ${usuario [sobreNome]}.

3. JSTL

A linguagem de expressões de JSP fornece um método simples e conveniente de acesso a propriedades de um escopo para executar de modo simples as expressões. Entretanto, não elimina o uso das *scriptlets* dentro de nossas páginas JSP. Por isso, apresentamos as *custom tags*, com a biblioteca padrão do Java.

3.1. Custom Tags

Um dos pontos fortes da especificação de JSP é que ela permite um grau de customização e de extensão com o uso de *tags* feitos sob medida. Existem *tags* especializadas dentro da especificação que executam tarefas específicas: as *tags* de JSP padrão <jsp:useBean>, <jsp:getProperty> e <jsp:setProperty> são exemplos. As *tags* fornecem uma funcionalidade reusável dentro de uma página JSP usando uma relação muito simples ao ocultar sua execução interna. Os desenvolvedores podem criar suas próprias *tags* para executar seu código com este tipo do benefício.

Com esta reusabilidade compacta, *custom tags* disponibilizadas em determinado JAR podem ser usadas em qualquer aplicação WEB. Não teríamos nenhuma vantagem se as *custom tags* não pudessem ser usadas por vários *frameworks* ou servidores JSP. Existem várias bibliotecas *tags* (*tag libraries*) disponíveis e é impossível evitar que tenham alguma sobreposição nas funcionalidades que fornecem.

Java reconhece isso e, em cooperação com a comunidade de programação, forneceu uma biblioteca padrão de *tags* que se dirige às áreas de sobreposição, chamada de *Java Standard Tag Library* ou JSTL.

3.2. Incluindo JSTL em nossa aplicação

Para incluir a biblioteca JSTL em nossa aplicação, no NetBeans acesse *Properties* do Projeto, em seguida selecione a opção *Libraries* e pressione o botão *Add Library...,* selecione a opção JSTL 1.1, pressione o botão *Add Library* e por fim pressione o botão OK.

Existem vários pacotes de bibliotecas JSTL. Somente a biblioteca *core* será discutida nesta lição.

3.3. Biblioteca Core

Core fornece funcionalidades úteis para todo o projeto WEB. *Core* pode ser sub-categorizada em:

- *Tag* de uso geral
- Repetição
- Condicional
- Manipulação de URL

Obs.:Para cada página JSP que possuir a biblioteca *Core*, a seguinte diretriz deve ser adicionada à página:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

Isto vincula as *tags* dentro da biblioteca *Core* usando o prefixo c.

3.3.1. Tags de uso geral

As *tags* de uso geral na *Core* executam tarefas comuns, embora uma delas tenha se tornado irrelevante com a liberação da especificação de JSP 2.0. As *tags* que compreendem o uso geral são: *out*, *set*, *remove* e *catch*.

<c:out>

A *tag* <c:out> recebe uma expressão, avalia seu conteúdo e retorna o resultado diretamente ao objeto corresponde à saída da página. Esta *tag* funciona da mesma maneira que a *tag* <jsp:getProperty>, com a exceção que um *JavaBean* não necessita acessar a funcionalidade.

Entretanto a especificação **JSP 2.0**, esta *tag* tornou-se obsoleta: as expressões EL podem ser avaliadas diretamente na *stream* de saída em qualquer parte da página JSP sem utilizar nenhuma *tag*.

<c:set>

Esta *tag* executa a mesma funcionalidade de uma *tag* <jsp:setProperty> que é capaz de ajustar valores dentro de um *JavaBean*. Pode também ajustar um atributo dentro de um escopo especificado que pode ser usado mais tarde pela JSP ou em outra parte da aplicação.

Esta ação tem os seguintes atributos:

- **value** – o valor que será ajustado no *JavaBean* especificado. Pode ser uma expressão EL
- **var** – o nome de um atributo que será declarado
- **scope** – define o escopo do atributo especificado pelo atributo var. Os valores podem ser: page, request, session ou application
- **target** – o nome do *JavaBean* cuja a propriedade será ajustada
- **property** – o nome da propriedade dentro do *JavaBean* que receberá o valor

Como foi mencionado antes, há dois usos preliminares para esta *tag*. Para ajustar o valor dentro de um *JavaBean*, a *tag* utiliza somente *value* (valor), *target* (alvo) e os atributos da propriedade:

```
<c:set target="usuario" property="nome" value="JEDI"/>
```

A chamada anterior é equivalente a empregar a seguinte expressão de JSP:

```
<%
   user.setName("JEDI");
%>
```

Para declarar um atributo em um escopo definido, a *tag* <c:set> utiliza somente o valor, var, e os atributos do escopo, embora o atributo do escopo seja opcional. Quando o atributo do escopo for omitido, será utilizado o escopo de página.

```
<c:set var="minhaString" value="Este é um teste String" scope="session"/>
```

No intuito de limitar a JSP para finalidades de apresentação, devemos limitar o uso desta *tag*. Ajustar propriedades do *JavaBean* ou ajustar variáveis em vários escopo são procedimentos que devem ser executados em outras partes do código, pois não são atribuições da camada de apresentação.

<c:remove>

Esta *tag* fornece uma maneira de remover os atributos de um escopo definido. Possui dois parâmetros:

- **scope** – o escopo do atributo a ser removido.
- **var** – o nome do atributo a ser removido do escopo definido.

Use esta *tag* para remover o atributo no escopo da sessão anterior criado pela *tag* <c:set>:

```
<c:remove var="minhaString" scope="session"/>
```

Como a *tag* <c:set>, o uso desta *tag* deve ser evitado. Remover as variáveis de um escopo não é atribuição dos objetos da camada de apresentação; este tipo da procedimento deve ser executado em outra parte da aplicação.

<c:catch>

A *tag* <c:catch> fornece a funcionalidade da manipulação de erros em áreas específicas de uma página JSP. É simples utilizar: coloca-se a instrução JSP dentro da *tag* <c:catch>

Este *tag* tem somente um atributo:

- **var** – define o nome que será usado na exceção. O atributo criado terá o escopo da página; isto é, será acessível mesmo depois que o bloco terminar.

Por exemplo, para obter exceções inesperadas dentro da página JSP, o seguinte código poderia

ser colocado:

```
<c:catch var="exception">
 <%-- Forçamos um erro aqui para a funcionalidade desta Tag --%>
 <%
 if (true) {
   throw new Exception("Eu sou um erro inesperado");
 }
 %>
</c:catch>

<%-- A tag seguinte é discutida com mais ênfase na seção condicional --%>
<c:if test="${! empty exception}">
 Ocorreu um erro na aplicação : ${exception.message}
</c:if>
```

3.3.2. Repetição

As repetições de JSTL são compostas por *do-while*, *while* e *for* em nossa página de JSP como *scriptlets*. JSTL fornece duas *tags*: *forEach* e *forTokens*.

<c:forEach>

Geralmente esta *tag* é mais utilizada para a repetição. Permite acesso por interação aos *arrays*, instâncias de java.util.Collection, java.util.Iterator, java.util.Map e java.util.Enumeration. Percorre cada elemento expondo o valor atual da repetição no código da página JSP contido na *tag*.

Esta *tag* possui os seguintes atributos:

- **var** – define o nome do atributo usado para exibir o valor atual da repetição na *tag*. Seu tipo é dependente da coleção que está sendo processada
- **items** – define a coleção da repetição. Pode ser especificado como uma expressão EL
- **varStatus** – (opcional) define o nome do atributo que pode ser acessado pelo laço para pegar o status do laço atual
- **begin** – (opcional) um valor interno que define o índice que será usado como o ponto de início da repetição. Se este valor não for fornecido, o índice de repetição começa com 0. Pode ser uma expressão de *runtime*
- **end** – (opcional) um valor inteiro que define o índice que será usado como ponto de parada do laço
- **step** – (opcional) um valor do tipo int que define o incremento a ser utilizado através das iterações. Por exemplo, definir este valor para 2 significa que os elementos serão acessados de 2 em 2, definir o valor 3 significa que os elementos serão acessados a cada 2 elementos contando do primeiro

Um uso comum para esta *tag* é interar sobre os resultados de um processamento realizado pela aplicação (provavelmente uma *servlet*). Peguemos como exemplo o seguinte cenário: temos um módulo da aplicação que recupera de um banco de dados os detalhes do usuário que resulta em uma categoria específica de procura. Naturalmente, queremos realizar o acesso lógico ao banco de dados através de uma servlet e passar os dados para apresentação pela JSP.

Abaixo, segue um código retirado de uma *servlet* que irá lidar com o acesso:

```
...
// carrega os parâmetros de usuário em um Map
Map parameters = loadUserParameters();
UserDataService service = new UserDataService();
// realiza uma pesquisa em banco e armazena os resultados em uma Collection.
Collection results = service.searchUsers(parameters);

// armazena os resultados para pesquisa futura
request.setAttribute("searchResults", results);
// repassa a requisição para a JSP exibir
request.getRequestDispatcher("/results.jsp").forward(request, response);
...
```

A página JSP a seguir é responsável por exibir o resultado. Assumiremos que o conteúdo de uma

coleção são instâncias de um objeto da classe *User* a qual tem um nome e um endereço *String* acessíveis via métodos *get* públicos.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<H1>Os seguintes usuários estão de acordo com seu critério de pesquisa : </H1>
<br/>
<c:forEach var="user" items="${requestScope.searchResults}">
    <li> ${user.name} - ${user.address}
</c:forEach>
```

Na página JSP, utilizamos a *tag forEach* para iterar sobre os resultados da procura. Isso é possível ao apontar a *tag forEach* para a instância da coleção armazenada no escopo da requisição usando EL. Então, será exposto cada elemento da coleção utilizando-se a variável de usuário que foi definida pelo atributo **var** e usando EL para exibir os valores.

Compare com o código abaixo como seria sem uso da JSTL:

```
<H1>Os usuários abaixo coincidem com seu critério de pesquisa : </H1> <br/>

<%
Collection results = (Collection) request.getAttribute("searchResults");
Iterator iter = results.iterator();

while (iter.hasNext()) {
 User user = (User) iter.next();
%>
 <li> <%= user.getName() %> - <%= user.getAddress() %>
<%
}
%>
```

É obvio que a versão em JSTL é muito mais compreensiva, especialmente para os designers de sites sem conhecimento em Java.

<c:forTokens>

Outra *tag* para repetição provida pela JSTL é a <forTokens>. Esta *tag* recebe uma *String* e analisa e extrai seu conteúdo em *tokens* baseados em um dado delimitador. Todos os atributos de uma tag forEach são compartilhados por esta *tag*. Além destes, o seguinte atributo também está disponível:

- **delims** – define o delimitador a ser passado quando extrair a *String* em *tokens*.

O atributo **items** agora possui um novo propósito nesta *tag*. Define a String a ser quebrada.

Segue um exemplo abaixo:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<c:forTokens items="item1,item2,item3,item4" delims="," var="item">
<li> ${item}
</c:forTokens>
```

As instruções JSP anteriores resultam na seguinte página:

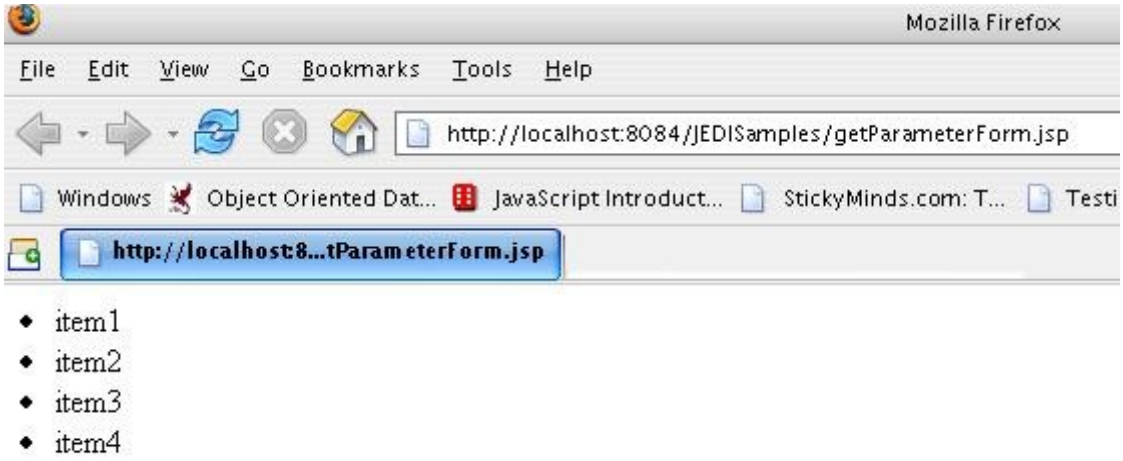

Figura 1: Saída resultante

3.3.3. Condicionais

O conjunto de *tags* nesta categoria imita a funcionalidade provida pelos conhecidos *if* e *else-if* que podem ser encontrados no padrão Java. Usar estas tags através do padrão Java permite um código mais limpo. Mudar dos scripts executados no servidor para uma saída normal em HTML que cria condicionais com *scriptlets* mais difíceis de compreender e gerenciar.

Exitem dois conjuntos principais de condicionais: a *tag* <c:if> que imita um simples *if* do Java e a *tag* <c:choose> relacionado as *tags* <c:when> e <c:otherwise>. Desta forma, estas *tags* relacionadas imitam a funcionalidade de um código *switch*.

<c:if>

A *tag* <c:if> permite a execução de seu conteúdo caso o valor da expressão a ser avaliada seja verdadeira. Caso seja falsa, nunca será executada.

Amostra:

```
<c:if test="${empty sessionScope.user}">
 Não está atualmente logado! Favor corrigir antes de continuar a adição.
</c:if>
```

<c:choose>, <c:when>, <c:otherwise>

Estas três *tags* trabalham juntas para prover funcionalidade semelhante ao comando *if-else* em sua aplicação. Basicamente, uma ou mais *tags* são colocadas na *tag* "choose". A *tag* "choose" avalia cada atributo de teste da *tag* "when" e executa a *tag* "when" apropriada quando avaliada como verdadeira. Caso não exista uma *tag* "when" relacionada com a condição, "choose" irá chamar a tag "otherwise", se incluída no código.

Amostra:

```
<c:choose>
 <c:when test="${userGrade >95}">Proeminente!</c:when>
 <c:when test="${userGrade > 80}">Bom!</c:when>
 <c:when test="${userGrade < 60}">Falhou!</c:when>
 <c:otherwise>Parabéns...</c:otherwise>
</c:choose>
```

3.3.4. Manipulação de URL

A JSTL disponibiliza algumas *tags* personalizadas que fornecem a manipulação básica de URL. Estas *tags* constroem e melhoram as ações já disponíveis na JSP.

<c:import>

A *tag* <c:import> funciona da mesma forma que a *tag include* da JSP, de forma melhorada. A *tag include* da JSP permite que somente páginas dentro da aplicação WEB sejam referenciadas e

incluídas com o conteúdo da página atual. Já a *tag* <c:import> permite aos desenvolvedores especificarem URLs absolutas que apontam para recursos externos.

O atributo a ser usado por esta *tag* é:

- **url** – o valor da URL para importar. Pode ser tanto uma URL relativa apontando para um recurso em uma aplicação WEB como uma URL absoluta que aponta para um recurso externo.

Esta *tag* contém vários outros atributos. Entretanto, são usados para modificar o conteúdo de um recurso incluído, que seria melhor se fosse feito fora da página JSP.

<c:param>

A *tag* <c:param> é versátil usada em conjunto com um número de outras *tags* com o grupo de manipulação de URL. Enfatizando, esta *tag* NÃO pode ser usada sozinha, mas somente como uma *tag* filha de outras *tags* disponíveis na JSTL. Sendo usada para disponibilizar um modo de incluir os parâmetros e valores de uma requisição através de uma URL.

Os atributos usados por esta *tag* são:

- **name** – indica o nome do parâmetro de requisição.
- **value** – indica o valor de um parâmetro de requisição.

A utilidade desta *tag* é melhor ilustrada com um rápido exemplo. Vamos considerar a *tag* que discutimos:

```
<c:import url="myResource.jsp?username=JEDI Program&location=here"/>
```

Caso queira incluir um recurso dinâmico para valores definidos como parâmetros de requisição, veja o trecho de código abaixo. Entretanto, o problema com o exemplo abaixo é que não está de acordo com a regras de codificação de URL. Existem diversas regras para codificar as URLs. Uma delas diz que os escopos precisam ser escritos usando um caractere especial. A tag <c:param> é capaz de abstrair estas regras de codificação. Em vez de se preocupar em como reescrever a URL com um conjunto de regras de codificação, podemos simplesmente fazer:

```
<c:import url="myResource.jsp">
   <c:param name="username" value="Programa JEDI"/>
   <c:param name="location" value="here"/>
</c:import>
```

<c:url>

A *tag* <c:url> é usada para prover um modo de codificar automaticamente a URL com as informações necessárias para determinada sessão. Lembre-se da discussão anterior sobre gerenciamento de sessões que dita que a reescrita da URL é um modo de permitir uso de sessões caso os *cookies* estejam desabilitados.

O atributo relevante nesta *tag* é:

- **value** – URL a ser processada.

Para esclarecimento, esta *tag* nem sempre permite codificar as URLs. Isto é feito SOMENTE quando se detecta que o *browser* do cliente não suporta *cookies*.

Abaixo segue um exemplo de uso da *tag*:

```
<HTML>
   <BODY>
      Clique <a href="
        <c:url value="continue.jsp"/>
      ">aqui</a> para entrar na aplicação.
   </BODY>
</HTML>
```

Assim como a tag "import", esta *tag* pode ter nenhuma ou várias *tags* <c:param> relacionadas aos parâmetros de inclusão.

4. Exercícios

1) Considere que um componente de sua aplicação WEB adicionou um Vector em um escopo de sessão contendo uma ou mais instâncias de um objeto *MenuItem* definido abaixo:

```
public class MenuItem {
  private String itemName;
  private String description;
  private double price;
  // implementação de métodos get e set aqui
}
```

Criar uma página JSP que irá obter um *Vector* que irá interagir sobre seu conteúdo mostrando cada propriedade dos objetos. Use EL e JSTL em sua implementação JSP.

2) Considere o cenário abaixo para uma página JSP que você irá criar: a página espera um parâmetro chamado *isDebug* que retorna um valor verdadeiro (true) ou falso (false). Caso o valor do parâmetro seja avaliado como verdadeiro, então seja exibir os seguinte itens:

   - o nome do usuário logado atualmente
   - o ID de usuário atualmente logado

   Assuma que estes valores são obtidos de um objeto user contendo as propriedades *name* e *userID*, respectivamente. A instância *User* foi armazenada no escopo de sessão usando a chave "*userObject*".

   De qualquer forma, sendo ou não verdadeiro o parâmetro *isDebug*, a página deve mostrar o seguinte conteúdo:

   "Grato por usar esta aplicação. Atualmente, a página requisitada está em construção".

3) Construir uma aplicação que permita que os usuários naveguem através de uma lista de classes e obtenha seus detalhes. Uma servlet pode ser utilizada para realizar uma pesquisa dada uma chave como parâmetro que está disponível e acessível no mapeamento */SearchServlet*.

   Criar uma página JSP que irá disponibilizar aos usuários uma caixa de texto para digitar a chave de procura. Caso a página seja submetida, este formulário irá transferir seu controle para a *Servlet* mencionada anteriormente. Do lado da caixa de texto, a página irá prover *links* nomeados de A a Z. Quando clicados, será submetida uma requisição para a *servlet*, onde os valores das chaves de pesquisa irão definir as letras que o resultado representa.

   Criar uma implementação para esta página utilizando JSTL e EL. NÃO representar de forma fixa cada *link* da página individualmente, ou seja, os *links* serão criados dinamicamente. Dica: interaja sobre a lista de letras para evitar um código fixo dos *links* na página.

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.