Fundamentos do JSP

Introdução

Nós veremos imediatamente alguns exemplos, seguidos por uma discussão da sintaxe básica de "tags" do JSP. Em seguida, consideraremos como configurar um "web server" para o uso em páginas JavaServer. Finalmente, nós examinaremos como a tecnologia trabalha realmente, discutindo como a execução do JSP afeta sua operação.

Escrevendo seu primeiro JSP

Nosso primeiro objetivo neste capítulo é apresentar exemplos para ilustrar a flexibilidade e potência do JSP como uma solução para a geração de páginas dinâmicas. Não focalizaremos em demasia os detalhes – haverá uma abundância destes mais tarde. A intenção aqui é simplesmente dar-lhe uma idéia do caminho a ser tomado nos capítulos seguintes.

Sobre os exemplos

Como indicado acima, o diferencial do JSP é sua habilidade em gerar páginas dinâmicas através de uma sintaxe familiar, o HTML. Ao mesmo tempo, pode ser difícil para novatos em JSP, reconhecer imediatamente onde seus elementos estão sendo usados dentro de uma página. Conseqüentemente, nos exemplos neste tutorial, que combinam elementos do JSP com outro conteúdo estático (tipicamente HTML), adotamos a convenção das "tags" do JSP em tais páginas e fragmentos, estarem em negrito. A intenção desta convenção é permitir aos leitores distinguir facilmente o conteúdo JSP do conteúdo estático, circunvizinho do original.

Olá, Mundo!

Nenhum tutorial estaria completo sem um exemplo que imprima "Olá, mundo!". Então aqui está um JSP que faz apenas isso:

```
<HTML>
<BODY>
Olá, Mundo!
</BODY>
</HTML>
```

Neste momento, você está pensando: "Hei! Este não é nada mais que um HTML!". E você está certo. É, não obstante, uma página válida do JSP. Esta página poderia ser adicionada a um "web server" configurado para rodar JSP, sendo interpretada normalmente se estiver com a extensão apropriada ".jsp".

Olá, Mundo! Revisado

Estabelecendo o fato que todos os originais válidos do HTML são também originais válidos do JSP, vamos então considerar um exemplo motivador. Está aqui um arquivo que gera conteúdo dinâmico com o uso de um par de "tags" JSP que suportam scripting:

```
<HTML>
<BODY>
<% String visitor = request.getParameter("name");
if (visitor == null) visitor = "Mundo"; %> Olá, <%= visitor %>!
</BODY>
</HTML>
```

Sem entrar em detalhes, este JSP declara uma variável "String" do Java chamada "visitor", e tenta então iniciá-lo no pedido atual do HTTP. Se nenhum valor para esta variável estiver definido, um valor padrão é atribuído. Uma expressão de JSP é usada então para introduzir o valor desta variável na saída do HTML da página.

Os parâmetros da requisição são passados para páginas JSP usando os mecanismos normais do protocolo HTTP. Para requisições "GET" do HTTP, valores de parâmetro codificados são adicionados simplesmente ao URL. Para requisições "POST" do HTTP, um protocolo mais complicado é usado emitir dados do parâmetro através do cabeçalho da página. Na prática, URLs maiores que 255 caracteres podem ser problemáticas, assim os pedidos "POST" são o padrão quando um grande volume de dados é requerido.

AVISO
Esta limitação do comprimento em URLs permite compatibilidade com browsers antigos. A especificação HTTP/1.1 não impõe realmente nenhum limite a priori no comprimento do URL, mas há uns servidores mais velhos, uns proxies, e uns browsers que não conseguem segurar URLs que excedem este limite 255 caracteres.

Para as finalidades deste exemplo, vamos supor que o URL para esta página JSP é http://server/webdev/fundamentals/helloScript.jsp. Se um "web browser" for usado para solicitar este URL através de um "GET" do HTTP, o recipiente JSP que processar esta página, responderá com o seguinte:

```
Olá, Mundo!
```

Se o valor de parâmetro apropriado fosse adicionado a este URL, entretanto, um resultado diferente seria obtido. Se o URL solicitado for http://servcr/wcbdcv/fundamentals/helloScript.jsp?name=Renato, por exemplo, a resposta do recipiente JSP seria:

```
Olá, Renato!
```

No primeiro caso, sem o valor do parâmetro (name), o script usou o valor padrão para a variável do visitante da página. No segundo caso, o certificado recuperou o valor do parâmetro conhecido do pedido, e assim que pôde gerar um conteúdo dinâmico mais personalizado.

Se você estiver utilizando um "web browser" para exibir estes exemplos, você pode neste momento usar o comando "Visualizar código-fonte" do browser para olhar o HTML desta resposta. Dessa forma, você poderia ver o conteúdo original do JSP, mas ao invés disso, você verá algo como o seguinte:

```
<HTML>
<BODY>
Olá, Renato!
</BODY>
</HTML>
```

Se você for novo na geração de páginas dinâmicas, você pode querer saber o que aconteceu com todas as "tags" JSP (por exemplo, `< % = visitante % >`). Este resultado parece como alguém deslizado no arquivo da seção anterior, introduzindo o nome "Renato" no lugar de algum do texto original.

No fato, isso é muito perto de o que está acontecendo realmente. Mantenha em mente que o "web browser" vê somente a resposta do "web server" a seu pedido, que pode ou não corresponder a algum código dinâmico. O browser não tem nenhuma maneira de saber que, quando o "web server" recebe um pedido que corresponde a uma página JSP, esse pedido é enviado ao recipiente JSP para processar. É o recipiente JSP que lê e interpreta o código do arquivo correspondente para gerar o conteúdo dinâmico, introduzindo os resultados no conteúdo estático já na página, e retornando a página terminada ao servidor HTTP. Esta é então a página que é emitida para o browser, com nenhuma evidência qualquer da atividade do que ocorreu atrás das cenas. Tudo que o browser vê é apenas uma página HTML. É este HTML de resposta que é apresentado pelo browser quando você utiliza o comando "Visualizar código-fonte".

Olá, Mundo! Edição Bean

Além de suportar scripting, o JSP inclui "tags" para interagir com Java-Beans. Para dar a este assunto de "Olá, Mundo!" a atenção merecida, forneceremos também um terceiro exemplo JSP que demonstra esta aproximação. Antes mesmo de apresentar um JSP utilizando "tags" JavaBeans, necessitamos primeiramente de um Bean para acessar. Está aqui o código de fonte para um Bean chamado HelloBean (OláBean), que tem uma propriedade, chamada `name` (nome):

```
package myBeans;
public class HelloBean implements java.io.Serializable {
String name;

   public HelloBean () {
      this.name = "World";
   }
   public String getName () {
       return name;
   }
public void setName (String name) {
       this.name = name;
   }
}
```

Se você for um programador de Java, você pode ver neste código que os JavaBeans são executados através das classes ordinárias do Java, que aderem a um conjunto de convenções para instanciar objetos, e para alcançar e ajustar suas propriedades.

Por ora, note que a classe de `HelloBean` tem um construtor que não recebe nenhum argumento, entretanto, atribui um valor padrão à propriedade `name` (nome). Esta propriedade, cujo o valor é uma instância da classe "String" do Java, é alcançada através de um método chamado `getName()`. Que é modificado através de um método chamado `setName()`, que faz exame de um único argumento do mesmo tipo que a própria propriedade (isto é, "String"). Dada esta definição da classe `HelloBean`, pode então ser usada em um arquivo JSP como segue:

```
<HTML>
<BODY>
<jsp:useBean id="hello" class="myBeans.HelloBean"/>
<jsp:setProperty name="hello" property="name" param="name"/>
Olá, <jsp:getProperty name="hello" property="name"/>!
</BODY>
</HTML>
```

A primeira "tag" JSP a aparecer nesta página é a `<jsp:useBean>`. Como o seu próprio nome sugere, é usada para indicar que um Bean de uma classe particular estará sendo usado nesta página. Aqui, a classe de `HelloBean` é especificada. Além, um identificador, `hello`, é especificado como um identificador para instanciar o Bean. Este identificador pode ser usado para consultar o Bean mais tarde no código do JSP.

A "tag" `<jsp:setProperty>` aparece em seguida. No formulário mostrado aqui, esta "tag" é usada para modificar a propriedade `name` do Bean `hello` baseada no valor do parâmetro `name`. De fato, esta Tag diz que o pedido deve ser procurado pelo parâmetro `name` e, se se for encontrado, este valor deve ser copiado do pedido (request) à propriedade `name` do Bean. A "tag" final do JSP em questão, `<jsp:getProperty>`, obtém o valor de uma propriedade do Bean e o apresenta na página no lugar da "tag" original. Neste caso, a "tag" recupera a propriedade `name` do Bean chamado `HelloBean`, associado ao identificador `hello`. Suponha, então, que o URL para esta página JSP é http://server/webdev/fundamentals/helloBean.jsp. Se esta página for acessada através de um "GET" do HTTP, como antes teremos:

```
Olá, Mundo!
```

Isto é porque o valor padrão da propriedade `nome` é o seguinte valor do tipo String: "Mundo". Nenhum parâmetro foi fornecido para que a "tag" `<jsp:setProperty>` pudesse ajustar o valor da propriedade, assim o valor padrão - ajustado pelo construtor que foi chamado em conseqüência da "tag" `<jsp:useBean>` - foi usado. Suponha, embora, que o URL http://server/webdev/fundamentals/hello-Bean.jsp?name=Alan fosse usado para acessar esta página. Neste caso, a "tag" `<jsp:setProperty>` encontraria um parâmetro com o mesmo nome que uma das propriedades do Bean (isto é, `name`). O valor de parâmetro é usado então para ajustar o valor da propriedade correspondente, tal que a resposta a este pedido seria:

```
Olá, Alan!
```

Neste caso, utilizamos dois arquivos para criar este conteúdo, o arquivo do código-fonte Java que definiu a classe Bean e o arquivo JSP que criou, modificou, e acessou o Bean a fim produzir esse índice. No exemplo anterior, todo o código foi incluído no único arquivo JSP.

É evidente entre estes dois exemplos, que a utilização de Beans promove a reutilização do código (vários JSPs poderiam fazer uso da classe de `HelloBean` sem a necessidade de rescrever todo o código adicional em Java) e também separa a apresentação da execução , ou seja, um arquivo contém somente o HTML e "tags" *XML-based*, o outro arquivo contem somente Java.

/------
/ Tutorial retirado da Serial Link Millenium 3000
/ www.seriallink.com
/ Tradução do livro: Web Development With JavaServerPages
/------