Primeiros Servlets

Introdução

Este tutorial é uma tradução feita pela Serial Link (http://www.seriallink.com) do Capítulo 2 do livro "Core Servlets and JavaServer Pages", por Marty Hall. Este primeiro tutorial sobre servlets pretende ensinar o básico, apresentando alguns exemplos e ensinando algumas técnicas bem simples, mas que serão muito úteis mais para frente.

Nota sobre a tradução (n.t.): O autor deste livro – Marty Hall – é uma pessoa jovem e utiliza uma linha de pensamento fácil de traduzir. Porém existem palavras particularmente complicadas, próprias do mundo da Internet, que são muito difíceis de se traduzir. Um exemplo disso é a palavra "buffer", que muitos programadores não conhecem exatamente seu significado, apesar de terem consciência do uso no mundo das aplicações "Web". A própria palavra "web" é utilizada normalmente por nós brasileiros sem a devida tradução. De qualquer forma algumas palavras são mais fáceis de se entender que outras. Muitas deixei em inglês mesmo, mas particularmente no caso da palavra "buffer", eu a traduzi para "amortecedor", pois é realmente o significado mais próximo que encontrei. Pois no contexto, essa palavra indica exatamente isso, ou seja, os dados enviados para o cliente são amortecidos e depois enviados de uma só vez, ou dependendo da configuração do amortecedor, podem ser enviadas paulatinamente. Entendendo-se o conceito em português, fica mais facil entender o seu significado.

O Básico da Estrutura de um Servlet

O código abaixo mostra um servlet simples que manipula requisições (requests) GET. As requisições GET, para aqueles que não estão acostumados com o protocolo HTTP, são as requisições comuns de um browser para páginas da web. Um browser gera este tipo de requisição quando o usuário digita uma URL no campo de endereço, clica num link a partir de uma página, ou envia um formulário HTML que não especifica o método (method) a ser utilizado. Servlets podem também trabalhar tranqüilamente com requisições POST, que são geradas exclusivamente quando alguém envia um formulário HTML que especifica explicitamente `METHOD="POST"`. Para maiores detalhes sobre formulários e requisições, consulte um manual de HTML.

Para ser considerada um servlet, a classe deve estender `HttpServlet` e interceptar as requisições `doGet` e/ou `doPost`, dependendo a partir de onde os dados estão sendo enviados: através do método GET ou POST. Se você quiser que o mesmo servlet intercepte ambos os métodos GET e POST, e ainda tire vantagem da mesma ação para cada um deles, você pode simplesmente fazer com que `doGet` chame `doPost`, ou vice-versa.

ServletTemplate.java

```
----------------------------------------
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet template.
*  <P>
 *  Taken from Core Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.coreservlets.com/.
 *  &copy; 2000 Marty Hall; may be freely used or adapted.
 */

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Use "request" para interceptar
    // requisições do cabeçalho HTTP
    // (ex.: cookies) e dados de um
    // formulário HTML (ex.: dados que
    // um usuário digitou no formulário e o enviou).

    // Use "response" para especificar os dados a serem
    // retornados através do cabeçalho HTTP
    // (ex.: content type, cookies).

    PrintWriter out = response.getWriter();
    // Use "out" para enviar dados ao browser
  }
}
```

Ambos os métodos (GET ou POST) manipulam dois argumentos: um `HttpServletRequest` e um `HttpServletResponse`. O `HttpServletRequest` possui um método através do qual você pode manipular informações provenientes de um formulário, por exemplo, ou de uma requisição HTTP, ou ainda o hostname do cliente. O `HttpServletResponse` lhe permite especificar informações que serão devolvidas, como mensagens de condição (200, 404 etc.), ou cabeçalhos (Content-Type, Set-Cookies etc.), e o mais importante, lhe fornece um `PrintWriter`, utilizado para enviar dados de volta ao documento (página) que o cliente está consultando. Em servlets simples, gasta-se muito tempo codificando instruções `println`, que geram a página desejada. Dados de formulários, requisições de cabeçalhos HTTP, e cookies serão discutidos mais a fundo nos próximos tutoriais.

Voltando ao exemplo, os métodos `doGet` e `doPost` geram exceções, e você precisa incluí-los na seção de declarações. Finalmente, você também deve importar classes de `java.io` (para `PrintWriter` etc.), `javax.servlet` (para

`HttpServlet` etc.), e `javax.servlet.http` (para `HttpServletRequest` e `HttpServletResponse`).

Tecnicamente falando, `HttpServlet` não é o único ponto de partida para se criar servlets, considerando-se que servlets podem, em princípio, estender e-mail, FTP, ou outros tipos de servidores. Os servlets para estas finalidades podem estender classes customizadas derivadas da Generic-Servlet, que são parentes muito próximas da classe `HttpServlet`. Na prática, entretanto, servlets são usados quase sempre para servidores de comunicação via HTTP (ex.: aplicações Web e servidores), e este será o assunto abordado neste tutorial.

Um Servlet Simples para Geração de Texto Puro

O próximo código mostra um servlet simples que apenas gera texto puro, mostrado na figura mais abaixo. O próximo tópico (Um Servlet para Geração de HTML) abordará a criação de código para gerar saída em HTML, que é mais comumente utilizada. Entretanto, antes de nos aprofundarmos neste assunto, devemos empenhar algum tempo no processo de instalação, compilação e execução deste exemplo mais simples. Você pode achar isso um pouco tedioso a primeira vista, mas seja paciente; uma vez aprendido este processo, o utilizaremos largamente nos próximos exemplos. Por isso, se você já está acostumado com esse processo, recomendo que você pule para o próximo tópico, mas se ainda tem dúvidas, não custa nada rever alguns passos importantes.

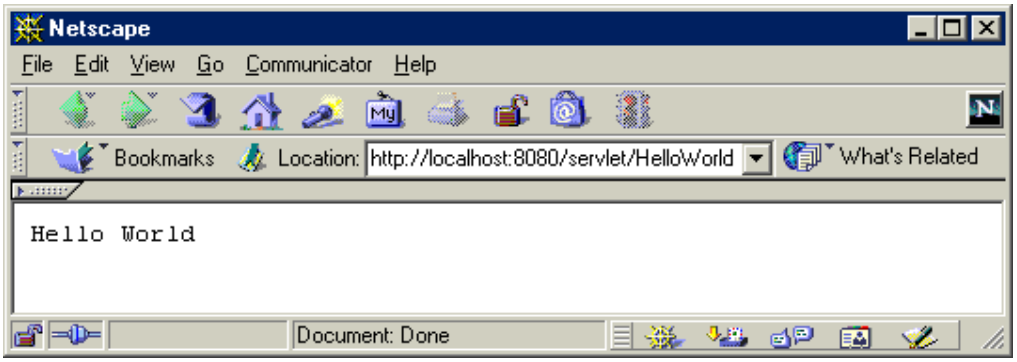
HelloWorld.java

```
----------------------------------------
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet bem simples para gerar texto puro.
 *  <P>
 *  Taken from Core Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.coreservlets.com/.
 *  &copy; 2000 Marty Hall; may be freely used or adapted.
 */

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```

Compilando e Instalando o Servlet

A primeira coisa que você precisa fazer é se certificar que seu servidor está configurado corretamente, e que a variável de ambiente `CLASSPATH` aponta para o arquivo `.JAR` que contém as classes padrão de servlets. Se você tem dúvidas sobre isso, consulte os tutoriais sobre como instalar um servidor JSP, na Serial Link (http://www.seriallink.com).

O próximo passo é decidir onde colocar seus servlets. Esta localização depende de servidor para servidor, portanto consulte a documentação do servidor que



você está utilizando para obter maiores detalhes (n.t.: No JWS2.0, o diretório para servlets é o `c:\JWS2.0\servlets` – onde JWS2.0 é o diretório onde você instalou o JavaWebServer. No Tomcat 4.0 você precisa criar um sub-diretório, dentro do diretório da sua aplicação web, chamado `WEB-INF`, e dentro dele então, um outro sub-diretório chamado `classes`. Dentro desse diretório `classes` você deve colocar seus servlets). Entretanto, existem algumas convenções comumente usadas, como descreverei abaixo.

1. **Um diretório para servlets que são compilados com freqüência**
   Servlets neste diretório são automaticamente recarregados quando o arquivo `.class` destes muda, portanto você deve utilizar este tipo de diretório durante os testes de desenvolvimento. Por exemplo, normalmente usamos `install_dir/servlets` no JavaWebServer da Sun e no WebSphere da IBM; para o BEA WebLogic, usamos `install_dir/myserver/servlet-classes`, embora a maioria dos servidores permitam ao administrador definir um diretório diferente. Servidores como o Tomcat ou o JSWDK não suportam recarga automática de servlets. Não obstante, eles possuem diretórios similares, para colocar os servlets; você precisa apenas parar e reiniciar o mini-servidor cada vez que uma recompilação se faz necessária. No Tomcat 3.0, coloque os servlets em `install_dir/webpages/WEB-INF/clasess` (n.t.: No Tomcat 4.0, utilize `install_dir/webapps/ROOT/WEB-INF/clasess`). No JSWDK 1.0.1, utilize `install_dir/webpages/WEB-INF/servlets`.
2. **Um diretório para servlets que <u>não</u> são compilados com freqüência**

Servlets que são colocados neste diretório executam com maior eficiência, pois o servidor não precisa ficar verificando se o arquivo `.class` mudou. Entretanto, mudanças nos servlets deste diretório não serão notadas a não ser que você reinicie o servidor. Esta opção (ou a Opção 3, logo abaixo) é muito utilizada em servlets que já estão em fase de produção. Este diretório normalmente se parece com `install_dir/classes`, que são comuns em servidores como o Tomcat, JSWDK e JavaWebServer. Como o Tomcat e o JSWDK não suportam servlets que são compilados com freqüência, esta opção acaba se tornando a mais utilizada entre os desenvolvedores que utilizam estes servidores.

3. **Um diretório para servlets que <u>não</u> são compilados com freqüência em arquivos .JAR**
   Assim como a opção acima (Opção 2), os servlets são colocados diretamente dentro dos diretórios `classes`, ou em sub-diretórios destes, correspondentes ao nome do pacote ao qual pertencem. Mas neste caso, os arquivos `.class` são empacotados em arquivos `.jar`, e por sua vez os arquivos `.jar` são colocados nestes diretórios. Em servidores como o Tomcat, JavaWebServer, JSWDK, e na maioria dos servidores, este diretório é o `install_dir/lib`. Como na Opção 2, o servidor também precisa ser reiniciado a cada modificação nos servlets.

Uma vez que você tenha escolhido qual opção usar, dentre as acima, configure seu `CLASSPATH` e coloque o servlet no diretório especificado. Utilize então o comando `javac HelloWorld.java` para compilar o servlet em questão. Em ambientes de produção, entretanto, os servlets são normalmente colocados em pacotes para evitar o problema de nomes iguais, em servlets criados por desenvolvedores diferentes. Utilizar pacotes `.jar` envolve uma série de passos extras que serão abordados num próximo tutorial. Também é comum utilizar formulários HTML como interface para os servlets. Para usá-los, você precisa saber onde colocar os arquivos HTML, afim de torná-los acessíveis para o servidor. Este lugar varia de servidor para servidor, mas no JSWDK e no Tomcat, você deve colocá-los em um diretório como `install_dir/webpages/path/file.html` e então acessá-lo no endereço correspondente, como http://localhost:8080/path/file.html (substitua `localhost` pelo real hostname do seu servidor, se estiver acessando-o remotamente). As páginas JSP podem ser colocadas em qualquer lugar, assim como as páginas HTML.

Executando o Servlet

Como vimos, os arquivos `.class` dos servlets podem ser colocados em vários locais diferentes, dependendo do servidor que você está utilizando. Entretanto, existe uma padronização quanto à execução (URL) destes em qualquer servidor. Para executar os servlets, a convenção utilizada é http://hostname:8080/servlet/ServletName. Note que a URL se refere a um diretório chamado `servlet` – no singular – e não `servlets` – no plural – como

realmente é o nome do diretório comumente utilizado. Mesmo quando o diretório, em alguns servidores, chama-se `classes` ou `lib`, o diretório que aparece na URL é sempre `servlet`.

A figura acima, que mostra o servlet `HelloWorld` em execução, lhe dá um exemplo da URL a ser usada. Repare que `localhost` é o hostname da máquina local, ou seja, o PC que estou trabalhando.

Um Servlet para Geração de HTML

A maioria dos servlets geram HTML, não texto puro como no nosso exemplo anterior. Para retornar HTML para o browser do cliente, você precisará de duas coisas:

1. "Dizer" ao browser que você está enviando HTML de volta; e
2. Modificar o comando `println` para construir uma página HTML válida.

O primeiro passo, você obtém configurando o Content-Type do cabeçalho de resposta HTTP. Normalmente os cabeçalhos de resposta HTTP são configurados através do método `setHeader` do `HttpServletResponse`, mas configurar o tipo de conteúdo a ser retornado (Content-Type) é uma tarefa tão comum que existe um método `setContentType` especialmente para esta finalidade. A maneira de retornar HTML para o browser é utilizando o tipo `text/html`. Então o comando para isso seria:

```
response.setContentType("text/html");
```

Embora o HTML seja o tipo de documento mais comum que um servlet pode gerar, isto não quer dizer que seja incomun gerar outros tipos de conteúdo. Por exemplo, os servlets podem construir e retornar imagens personalizadas, ao se configurar o Content-Type para `image/gif`. Um outro exemplo é utilizar servlets para gerar e retornar planilhas de cálculo do Excel, configurando o Content-Type como `application/vnd.ms-excel`.

Não fique preocupado se você não está familiarizado com os tipos de cabeçalhos de resposta HTTP existentes. Isso será discutido num próximo tutorial. No momento, tudo o que você precisa saber é configurar o Content-Type para retornar HTML.

Note que você precisa configurar o cabeçalho de resposta antes de enviar qualquer conteúdo para o `PrintWriter`. Isto acontece pois uma resposta HTTP consiste em uma linha de condição (status line), um ou mais cabeçalhos, uma linha em branco (blank line), e o documento atual, nesta mesma ordem. Os cabeçalhos podem aparecer em qualquer ordem, e os servlets amortecem (buffer) estes e os enviam todos de uma só vez. Portanto é válido configurar o código de condição (status code) – parte da primeira linha retornada – antes

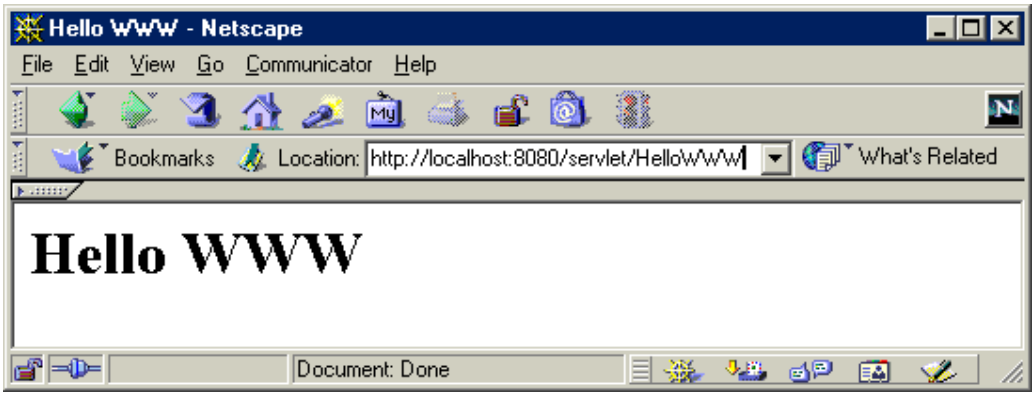
mesmo de configurar o cabeçalho de retorno. Mas os servlets necessariamente não amortecem o documento em si, impossibilitando que o usuário veja partes de uma página longa que está sendo carregada. Na versão 2.1 da especificação dos servlets, a saída `PrintWriter` não é amortecida, portanto quando o servlet executa o `PrintWriter`, já é tarde demais para enviar dados para o cabeçalho. Na especificação 2.2, os servlets permitem amortecer parcialmente a saída, mas o tamanho do amortecedor (buffer) é desconhecido. Neste caso você pode utilizar o método `getBufferSize` do `HttpServletResponse` para determinar o tamanho e ativar o amortecedor. Na especificação 2.2, com o amortecedor ligado, você pode configurar o cabeçalho mesmo enquanto a resposta está sendo amortecida para o cliente. Se você não está certo se a resposta terminou de ser amortecida, utilize o método `isCommitted` para verificar.

O segundo passo para escrever um servlet que gere um documento HTML é escrever instruções `println` que enviem HTML para o browser e não mais texto puro. A estrutura de um documento HTML será discutida mais para frente neste tutorial; mesmo assim deve ser familiar para a maioria dos leitores. Abaixo podemos ver um código que envia HTML para o browser. Mais abaixo poderemos ver o resultado desse exemplo ao ser executado.

HelloWWW.java

```
----------------------------------------
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet para gerar conteudo HTML.
 *  <P>
 *  Taken from Core Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.coreservlets.com/.
 *  &copy; 2000 Marty Hall; may be freely used or adapted.
 */

public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
```

```
}
```



Empacotando Servlets

Em um ambiente de produção, vários programadores desenvolvem servlets para um mesmo servidor. Colocar todos os servlets no mesmo diretório, resulta em um monte de pedaços do programa, difíceis de se organizar. E ainda corre-se o risco de dois desenvolvedores acidentalmente atribuirem o mesmo nome para servlets diferentes. Pacotes são a melhor solução para este problema. Usar pacotes resulta em mudar o modo de desenvolver os servlets, o modo de compilá-los, e o modo de executá-los. Vamos discutir esses modos, um de cada vez, nas três subseções seguintes. As duas primeiras mudanças são exatamente as que ocorrem com qualquer classe Java que utilize pacotes.

Criando Servlets em Pacotes

Dois passos necessários para colocar servlets em pacotes:

1. Mova os arquivos `.class` para um sub-diretório que tenha o mesmo nome do pacote a ser usado.
   Por exemplo, irei usar o pacote `coreservlets` na maioria dos exemplos deste tutorial. Então os arquivos `.class` precisam ser gravados em uma pasta chamada `coreservlets`.
2. Insira uma instrução `package` (pacote) no arquivo `.class`.
   Por exemplo, para colocar um arquivo `.class` em um pacote chamado `somePackage`, a primeira linha do código deve ser:

   ```
   package somePackage;
   ```

Para exemplificar, veja o próximo código que apresenta uma variação ao `HelloWWW`, mostrado anteriormente, agora incluído ao pacote `coreservlets`. No Tomcat ou no JSWDK, o arquivo deveria ser colocado em um diretório como `install_dir/webapps/ROOT/WEB-INF/classes/coreservlets`. No

JavaWebServer poderia ser o diretório `install_dir/servlets/coreservlets`.

HelloWWW2.java

```
----------------------------------------
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet para gerar conteudo HTML.
 *  <P>
 *  Taken from Core Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.coreservlets.com/.
 *  &copy; 2000 Marty Hall; may be freely used or adapted.
 */

public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Compilando Servlets em Pacotes

Existem duas maneiras de compilar classes que fazem parte de pacotes. A primeira opção é colocar o sub-diretório de seu pacote diretamente dentro do diretório onde o servidor espera encontrar os servlets. Então você precisa configurar a variável `CLASSPATH` apontando para o diretório exatamente um nível acima do sub-diretório do seu pacote, ou seja, o próprio diretório de servlets. Você pode então compilar os servlets normalmente a partir do sub-diretório do seu pacote. Por exemplo, se o diretório base dos servlets do seu servidor é `c:\JWS2.0\servlets` e o nome do seu pacote – também portanto o nome do sub-diretório – é `coreservlets`, e você está usando o Windows, então você deveria fazer assim:

```
set CLASSPATH=c:\JWS2.0\servlets;%CLASSPATH%
cd c:\JWS2.0\servlets\coreservlets
javac HelloWWW2.java
```

A primeira linha, configurando o CLASSPATH, você provavelmente gostaria de torná-la permanente, ao invés de ter que digitá-la cada vez que você abre uma janela nova do DOS. Para isso, no Windows 95/98/ME, você precisa colocar essa instrução set CLASSPATH=... no seu arquivo c:\autoexec.bat, em algum lugar após a instrução CLASSPATH que aponta para servlet.jar. No Windows NT/2000 você precisa ir no menu Iniciar (Start), selecionando Configurações (Settings), Painel de Controle (Control Panel), Sistema (System) e finalmente clicando em Ambiente (Environment). Então digite o nome da variável e seu respectivo valor. No Unix (C shell), você deve configurar a variável CLASSPATH assim:

```
setenv CLASSPATH /install_dir/servlets:$CLASSPATH
```

Coloque esta instrução no seu arquivo .cshrc para torná-la permanente. No Linux é um pouquinho diferente. Você deve colocar esta instrução no seu arquivo /etc/profile, e depois exportá-la, assim:

```
CLASSPATH="/install_dir/servlets:$CLASSPATH"
export PATH USER LOGNAME ... INPUTRC CLASSPATH
```

Se o seu pacote for da seguinte forma pacote1.pacote2.pacote3, ao invés de pacote1 como estamos utilizando, mesmo assim o CLASSPATH ainda deve apontar para o diretório de servlets.

A segunda opção para compilar classes que estão em pacotes, é mantendo o código fonte (arquivo .java) num lugar diferente do servlet (arquivo .class). Primeiro, você coloca o diretório do seu pacote em qualquer lugar que você ache conveniente, e configure o CLASSPATH apontando para ele. Segundo, use a opção -d do javac para instalar o arquivo .class no diretório onde o seu servidor espera. Veja o exemplo a seguir para entender melhor:

```
cd c:\MeusServlets\coreservlets
set CLASSPATH=c:\MeusServlets\coreservlets;%CLASSPATH%
javac -d c:\tomcat\webapps\ROOT\WEB-INF\classes HelloWWW2.java
```

Manter o código fonte separado dos arquivos .class é um método que eu utilizo na etapa de desenvolvimento. Para complicar minha vida mais ainda, eu tenho uma variedade de configurações para o CLASSPATH, que utilizo em meus projetos. E ainda uso o JDK 1.2, e não o JDK 1.1, como meu servidor espera. No Windows, eu acho este método mais conveniente para automatizar o processo de compilação dos servlets, com um arquivo de lote (batch) chamado servletc.bat, mostrado abaixo (a linha do CLASSPATH foi quebrada para facilitar a leitura, mas lembre-se de colocar tudo na mesma linha na hora de implementar no seu sistema). Eu coloquei este arquivo de lote no diretório c:\windows\command, mas você pode colocá-lo em qualquer lugar indicado no

PATH do seu autoexec.bat. Feito isso, eu vou para o meu diretório c:\MeusServlets\coreservlets e executo:

```
servletc HelloWWW2.java.
```

Servletc.bat

```
-----------------------------------------
@echo off

rem Esta versão é para Java Web Server
rem lembre-se de colocar as linhas abaixo na mesma linha!

set CLASSPATH=C:\JavaWebServer2.0\lib\servlet.jar;
     C:\JavaWebServer2.0\lib\jsp.jar;
     C:\MYDOCU~1\Marty\CSAJSP\Java-Code;
     C:\MYDOCU~1\Marty\CSAJSP\Java-Code\JDBC

javac -d C:\JavaWebServer2.0\servlets %1%
```

Executando Servlets em Pacotes

Para executar um servlet que esteja dentro de um pacote, utilize a seguinte URL:

http://localhost:8080/servlet/nomePacote.nomeServlet

Ao invés de:

http://localhost:8080/servlet/nomeServlet

Utilitário Simples para Montagem de HTML

Um documento HTML normalmente é estruturado da seguinte forma:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
     <TITLE>...</TITLE>
</HEAD>
<BODY>
...
</BODY>
</HTML>
```

Você provavelmente se sentirá tentado a omitir parte desta estrutura, especialmente a linha DOCTYPE, que a maioria dos browsers ignora ao

interpretar o HTML. Embora nas novas especificações do HTML 3.2 e 4.0, esta linha seja necessária. A vantagem da linha DOCTYPE é que ela indica aos browsers qual versão do HTML está sendo usada. Os browsers então podem fazer a validação do HTML corretamente. Esta validação é muito útil em serviços de verificação da sintaxe (debugging), o ajudando a encontrar erros no código, que podem rodar perfeitamente em determinado browser e apresentar problemas em outros. Os dois mais populares serviços de validação on-line são World Wide Web Consortium (http://validator.w3.org) e Web Design Group (http://www.htmlhelp.com/tools/validator). Estes sites lhe permitem indicar uma URL, onde eles recuperam a página, verificando a sintaxe conforme a especificação indicada no HTML, reportando os erros existentes. Como um servlet constrói HTML dinamicamente, este pode ser validado normalmente, a não ser que uma requisição POST seja necessária para retornar a página a ser validada. Lembre-se que as requisições GET enviam dados anexados à URL, então podendo ser validados normalmente.

Na verdade, é um pouco incômodo gerar HTML com instruções println, especialmente linhas longas como a declaração DOCTYPE. Alguns programadores resolvem este problema escrevendo extensos utilitários para geração de HTML em Java. Eu tenho minhas dúvidas sobre a utilidade dessas bibliotecas monstruosas para este fim. Afinal de contas a inconveniência da geração de conteúdo HTML é facilmente resolvida usando-se JSP (como discutido nos tutoriais sobre JSP). O JSP é uma solução muito mais amigável para se construir HTML complexos. Um segundo aspecto é que rotinas para geração de HTML podem ser realmente tediosas e mesmo assim não cobrir todos os atributos do HTML, como CLASS ou ID para style sheets, cor de fundo para células de tabelas etc. Apesar de ser questionável o valor de se construir bibliotecas de geração HTML, se você sentir que está repetindo muitos trechos, muitas vezes; então considere a possibilidade de usar pequenos utilitários para simplificar a codificação destes trechos repetitivos. Para servlets comuns, existem dois trechos de HTML (DOCTYPE e HEAD), que aparecem em todas as páginas e utilizam a mesma sintaxe quase sempre. Podemos nos beneficiar então, incorporando-os em um pequeno arquivo utilitário, conforme mostram os códigos abaixo. Repare que uma modificação foi feita no HelloWWW2.java para fazer uso deste arquivo utilitário.

ServletUtilities.java

```
-----------------------------------------
package coreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";

  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
```

```
            "<HTML>\n" +
            "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }
}
```

HeloWWW3.java

```
-----------------------------------------
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWWW3 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(ServletUtilities.headWithTitle("Hello WWW") +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Tradução por: Renato Graccula
http://www.seriallink.com