Desenvolvimento Web I

Aula 02 - Introdução aos Servlets

Apresentação

Vídeo 01 - Apresentação

Olá, meu amigo! A partir desta aula você começará a aprender as tecnologias da linguagem Java utilizadas para a programação de sistemas para a internet. Na disciplina de Autoria Web, você aprendeu a desenvolver as chamadas páginas web estáticas, certo? Aquelas páginas cujo conteúdo é definido previamente, ou seja, é sempre o mesmo e sabido antes mesmo do acesso do usuário (ex.: um texto fixo). Inclusive, este momento é uma boa oportunidade para você relembrar os principais marcadores HTML utilizados na construção das páginas web estáticas. Isso porque eles continuarão a ser utilizados aqui. Então, na dúvida, não hesite em rever AGORA e no decorrer desta aula os principais conceitos sobre esse assunto (marcadores HTML, BODY, FORM, A, INPUT etc.)!

Nesta aula, você irá entender o funcionamento básico da programação de páginas web dinâmicas em Java, aquelas cujo conteúdo pode mudar a cada acesso. Pense no seguinte desafio: o que você pode precisar fazer para implementar uma calculadora na web com as operações básicas de soma, subtração, divisão e multiplicação? Você já sabe criar páginas HTML para representar a calculadora e certamente sabe o código Java que implementa essas operações aritméticas. Falta agora aprender a integração dessas coisas, e isso é o que você verá nesta aula.

Boa leitura!

Objetivos

- Entender o que são Servlets e seus ciclos de vida.
- Compreender o funcionamento dos métodos de processamento de requisições web dos Servlets.
- Saber a configuração da aplicação web necessária para o funcionamento dos Servlets.

Páginas web estáticas *versus* dinâmicas

Para começar, vamos ilustrar a diferença entre páginas estáticas e dinâmicas. Pense em um sistema que implementa as operações básicas de uma calculadora. Sua página de entrada pode ser a mostrada na Figura 1. Ao digitar os valores 20 e 10 nos campos de nome valor 1 e valor 2, e então clicar no botão *, a tela da Figura 2 é apresentada. Entretanto, se for clicado o botão +, a tela da Figura 3 é que deve ser apresentada.

Figura 01 - Tela HTML de um sistema de calculadora na web

Figura 02 - Tela dinâmica de resposta para a multiplicação de 2 x 10

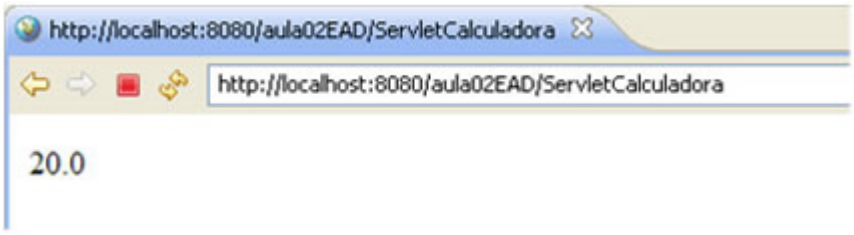
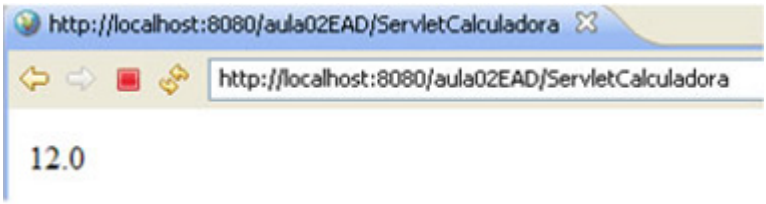


Figura 03 - Tela dinâmica de resposta para a soma de 2 + 10



Como podemos notar, a montagem da página web de resposta da calculadora precisa ser feita de forma dinâmica, ou seja, em tempo de execução. Dizer que uma página é dinâmica equivale dizer que seu conteúdo é gerado dinamicamente ou gerado em **tempo de execução**. Em todos esses casos, o que queremos dizer é que

o conteúdo da página não pode ser definido previamente em sua totalidade. No caso da calculadora, o conteúdo da página de resultado a ser apresentado depende dos valores informados pelo usuário.

Vídeo 02 - Introdução aos Servlets

Para criarmos páginas web dinâmicas em Java, fazemos uso dos chamados Servlets, como iremos lhe mostrar a seguir.

Atenção!

Tempo de execução é utilizado em contraste ao chamado tempo de compilação. No **tempo de compilação**, o programador está alterando o código e compilando para achar erros e gerar os arquivos executáveis. Depois disso, podemos rodar o programa. O período durante o qual o programa está rodando é chamado de **tempo de execução**.

Tecnologia de Servlets Java

Para gerarmos páginas dinâmicas no servidor de aplicações web, fazemos uso de componentes de software chamados **Servlets**. Você lembra o que são classes em Java? Aqueles módulos do programa que possuem atributos e métodos, os quais por sua vez descrevem respectivamente as características e comportamento (operações) comuns aos objetos dessa classe. Pois bem, os **Servlets são nada mais que classes em Java que herdam código (atributos e métodos) direta ou indiretamente de uma classe especial chamada HttpServlet existente no pacote javax.servlet.http**, como mostrado no exemplo da Listagem 1.

Nesta aula, você irá entender o funcionamento básico da programação de páginas web dinâmicas em Java, aquelas cujo conteúdo pode mudar a cada acesso. Pense no seguinte desafio: o que você pode precisar fazer para implementar uma calculadora na web com as operações básicas de soma, subtração, divisão e multiplicação? Você já sabe criar páginas HTML para representar a calculadora e certamente sabe o código Java que implementa essas operações aritméticas. Falta agora aprender a integração dessas coisas, e isso é o que você verá nesta aula.

```
1  package aula02;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import javax.servlet.ServletException;
7  import javax.servlet.annotation.WebServlet;
8  import javax.servlet.http.HttpServlet;
9  import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 @WebServlet("/ServletExemplo")
13 public class ServletExemplo extends HttpServlet {
14    private static final long serialVersionUID = 1L;
15    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Servle
16       // TODO Auto-generated method stub
17       PrintWriter saida = response.getWriter();
18       saida.write("<HTML><BODY>Olá!</BODY></HTML>");
19       saida.close();
20    }
21
22 }
23
```

Listagem 1 - Meu primeiro Servlet.

Atenção!

Caso esteja em dúvida sobre conceitos de orientação a objetos, como herança, polimorfismo e sobrescrita de método, aproveite para reler as aulas da disciplina de Programação Orientada a Objetos.

A classe **ServletExemplo** mostrada herda de **HttpServlet** e sobrescreve o método de nome **doGet**, cujos detalhes serão vistos mais adiante. O conteúdo da página resultante pela execução do Servlet é escrito através do objeto referenciado

pela variável **saida** que é um objeto que representa o canal de comunicação do servidor para o cliente, sendo inicializada através do comando **response.getWriter()**.

Em seguida utilizamos o método write(String msg) desse objeto de saída para escrever o conteúdo da página HTML a ser gerada. O parâmetro passado para esse método é uma String que vai formar o conteúdo da página, sendo, nesse caso, o texto "<HTML><BODY>Olá!</BODY></HTML>". Na verdade, você pode chamar o método write várias vezes dentro de um Servet, como veremos em outros exemplos.

Por fim, chamamos o método close() para fechar a conexão entre o cliente e o servidor. Nesse momento, o navegador web do usuário irá saber que todo o conteúdo da página já foi gerado e pode ser apresentado.

Atenção!

Os detalhes sobre os parâmetros request e response do método doGet serão apresentados na próxima aula.

Para visualizarmos o resultado da execução do ServletExemplo, primeiro precisamos informar ao servidor web que ServletExemplo é um Servlet e que ele pode ser executado pelos usuários da aplicação web. Em versões anteriores do Java Web, isso era feito alterando-se o arquivo **web.xml**, como mostrado na Listagem 2.

Veja um vídeo que mostra o processo de criação de um outro projeto de exemplo com um servlet ajudar a entender todo o processo.

Vídeo 03 - Meu Primeiro Servlet

Não é necessário alterar o aquivo **web.xml** no nosso caso. Esta configuração aplica-se apenas a versões anteriores do Java Web.

```
1  < ?xml version="1.0" encoding="UTF-8"?>
2    <web-app
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xmlns="http://xmlns.jcp.org/xml/ns/javaee"
5          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
6          id="WebApp_ID"
7          version="3.1">
8   <display-name>aula02</display-name>
9   <welcome-file-list>
10    <welcome-file>index.html</welcome-file>
11  </welcome-file-list>
12  <servlet>
13    <description></description>
14    <display-name>ServletExemplo</display-name>
15    <servlet-name>ServletExemplo</servlet-name>
16    <servlet-class>aula02.ServletExemplo</servlet-class>
17  </servlet>
18  <servlet-mapping>
19    <servlet-name>ServletExemplo</servlet-name>
20    <url-pattern>/ServletExemplo</url-pattern>
21  </servlet-mapping>
22 </web-app>
```

Listagem 2 - Indicação dos Servlets do sistema

Observe primeiro o marcador **<servlet>**, que vai da linha 12 à 17. Ele é utilizado para indicar a existência do ServletExemplo. O marcador **<servlet-class>** tem como objetivo informar ao servidor o nome completo da classe Java. Já o marcador **<servlet-name>** indica um apelido para o Servlet, no caso, está igual ao nome ServletExemplo. Este será então o nome usado em outras partes do arquivo web.xml. Por fim, os marcadores **<description>** e **<display-name>** são utilizados para fins de documentação.

Com relação ao marcador **<servlet-mapping>** (da linha 18 à 21), ele é utilizado para habilitar o ServletExemplo a ser executado via web por um usuário do sistema. Basicamente ele é o responsável pelo mapeamento de uma URL relativa ao Servlet. O marcador **<servlet-name>** indica o nome (apelido) do Servlet, que deverá ser o mesmo nome definido na tag **<servlet-name>** do marcador **<servlet>** (conforme as linhas 16 e 20 da Listagem 2), enquanto o marcador **<url-pattern>** indica a URL efetivamente a qual o Servlet está associado. No exemplo mostrado, o ServletExemplo está associado à URL relativa **/ServletExemplo.**

Porém, caso o seu sistema já esteja utilizando a versão 3.0 ou superior do Servlet, é adicionada a anotação @WebServlet que remove a necessidade de criação do mapeamento do servlet no web.xml. A anotação já define o webservlet e o mapeamento de acordo com o nome do arquivo java criado. Seu aplicativo estará configurado desta forma caso esteja seguindo o passo a passo desta aula.

Caso o nome de sua aplicação web no servidor seja ProgramacaoWeb e considerando que o servidor web está rodando na porta 8080, a URL para acessar esse Servlet é http://localhost:8080/ProgramacaoWeb/ServletExemplo.

Esse trabalho de configuração deve ser feito para todos os Servlets criados, o que pode ser um trabalho nada motivador quando for necessário criar dezenas de Servlets. Felizmente, ferramentas de desenvolvimento, como o Eclipse, realizam esse trabalho de configuração de forma automática. Para isso basta utilizar a opção **New / Servlet** do Eclipse ao invés de New / Class.

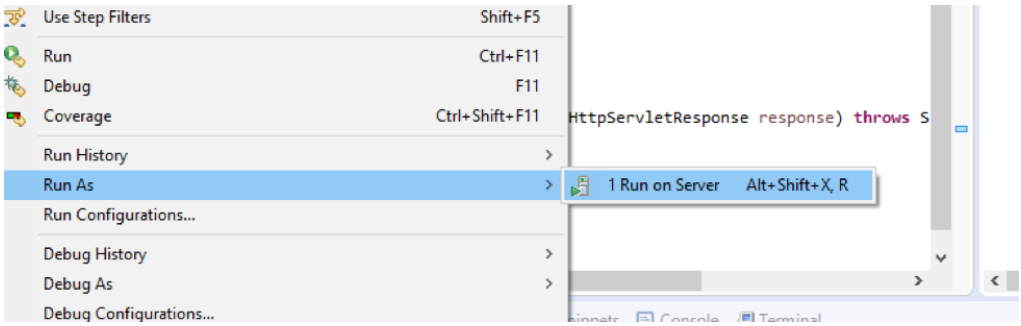
Dica

Utilize a opção New / Servlet do Eclipse ao invés de New / Class para criar novos Servlets. Assim, a configuração dos Servlets será feita automaticamente, seja através do web.xml ou da adição da anotação @WebServlet na classe criada.

Tecnologia de Servlets Java II

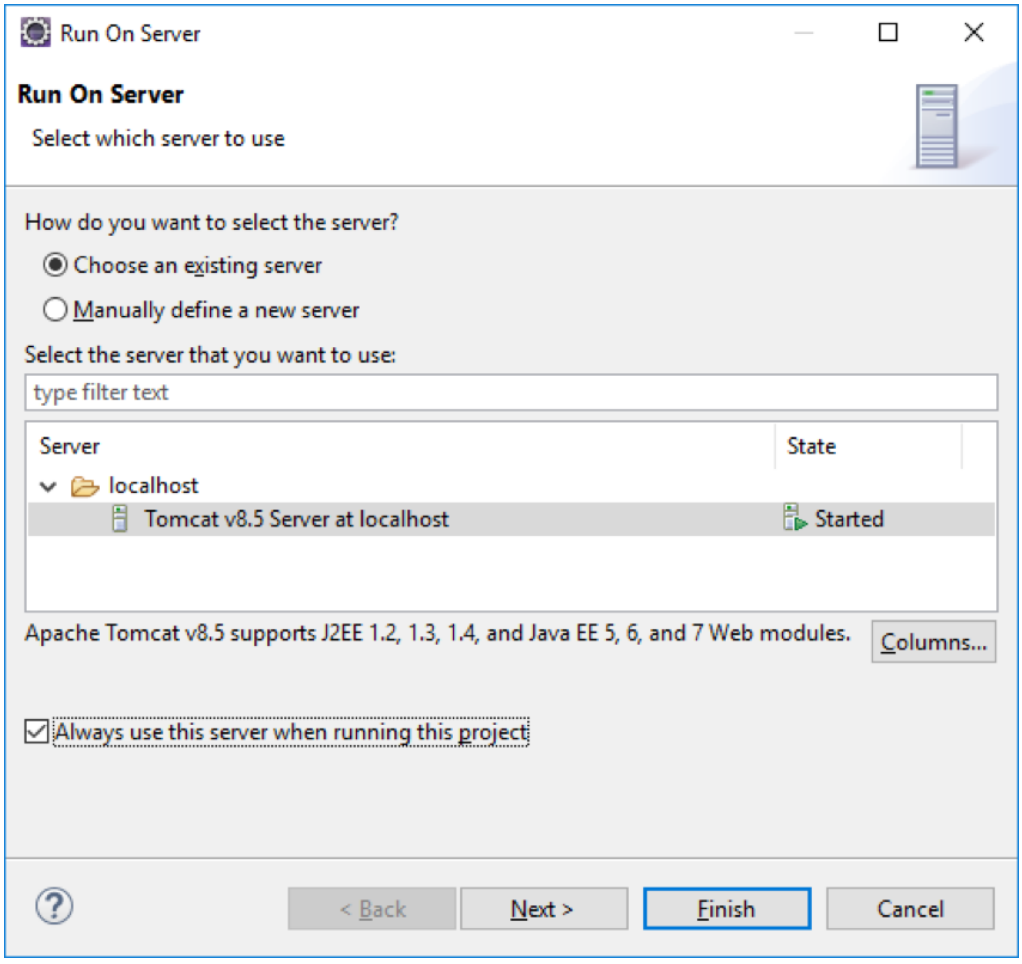
Para executar um Servlet no Eclipse, utilize a opção **Run / Run as / Run on Server** que aparece no Eclipse ao se clicar com o botão direito em cima do Servlet (veja a Figura 4).

Figura 04 - Executando um Servlet no Eclipse



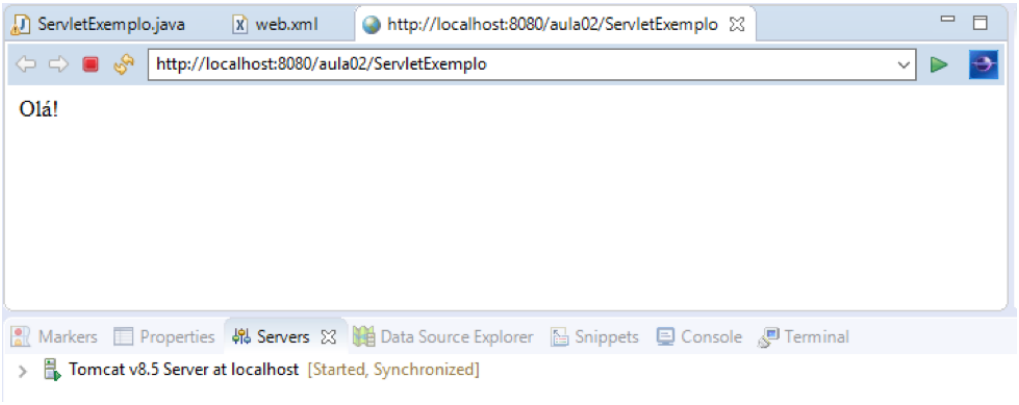
Na primeira vez, caso o seu projeto não esteja associado a um servidor web integrado ao Eclipse, a Figura 5 será apresentada. Nesse caso, selecione o servidor instalado para ser o servidor web da sua aplicação. Lembre-se de marcar a opção *Always use this server when running this project*, evitando que essa tela se abra nas próximas vezes.

Figura 05 - Indicando o servidor Web a ser utilizado



Enfim, após o ServletExemplo ser executado, a janela da Figura 6 será apresentada mostrando que tudo está configurado e funcionando corretamente. Caso isso não ocorra, observe qual foi o problema através da mensagem de erro que irá aparecer na aba Console, na parte inferior da janela.

Figura 06 - Resultado da execução do ServletExemplo



Atenção!

Lembre-se sempre de configurar o arquivo web.xml após criar um novo Servlet, caso isso não seja feito de forma automática pela ferramenta de desenvolvimento utilizada. Nas próximas aulas, iremos considerar sempre que a criação dos Servlets será feita pelo menu New | Servlet, o que garantirá que o arquivo web.xml seja configurado corretamente com o Servlet criado.

Atividade 01

1. Defina a classe ServletExemplo como mostrado anteriormente e execute-a no Eclipse, conforme instruções já apresentadas (Run on server). Reporte se você encontrou algum problema.

2. Crie um Servlet chamado de ServletMeuNome que gera uma página com seu nome. Execute o Servlet e veja se ele está funcionando corretamente.

3. Crie uma classe ServletQualMeuNome para gerar uma página com a pergunta “Qual o meu nome?”. Esse texto deve possuir um link de forma que, quando clicado, execute o ServletMeuNome. Execute o Servlet e veja

se ele está funcionando corretamente. **Dica:** use o marcador <a> para criar um link para a URL que executa o ServletMeuNome

4. Crie um Servlet chamado ServletAgenda que receba parâmetros de uma entrada de agenda (nome, telefone do usuário e data de nascimento) e que retorna uma página HTML apresentando os dados recebidos. Crie também uma classe chamada ServletPaginaAgenda que retorna uma página HTML com um formulário para enviar os dados para o ServletAgenda criado anteriormente.

Ciclo de vida dos Servlets

Muito bem, agora que você já sabe criar um Servlet e executá-lo, vamos avançar nos conceitos e recursos dessa tecnologia, que é a base da programação web em Java. Em primeiro lugar, é importante saber que os Servlets seguem um ciclo de vida. Quando uma requisição web realizada por um cliente é mapeada para um Servlet, ou seja, requer a execução de um Servlet, o contêiner web realiza os passos a seguir.

1. Se o Servlet ainda não tiver sido carregado (ou seja, é a primeira vez que o Servlet é acessado pelo contêiner):
    a. Carrega a classe do Servlet, isto é, **lê o bytecode** (código compilado) da classe.
    b. Instancia a classe, ou seja, cria um objeto a partir da classe carregada. Para isso, **executa-se o construtor vazio** do Servlet.
    c. Executa o método **init()** do Servlet, responsável por qualquer inicialização ou configuração que se faça necessária para o funcionamento do Servlet.
2. Executa o método **service()** definido na superclasse HttpServlet (e assim herdado pelos Servlets). Esse método será responsável por identificar o tipo da requisição web e invocar o método de tratamento apropriado (**doGet(), doPost()** etc.), como descrito em mais detalhes na próxima seção desta aula.

Por fim, quando os Servlets não são mais necessários (ex.: quando o servidor web está sendo desligado ou reiniciado), o contêiner web executa o método destroy() de cada Servlet que foi instanciado.

Vídeo 04 - Funcionamento do Servlet

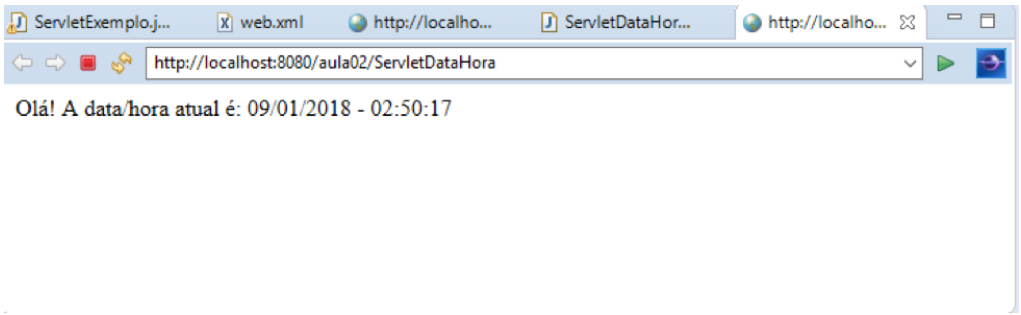
Vejamos através da Listagem 3 o código de um Servlet utilizado para imprimir a data e a hora na qual ele foi carregado, executado e finalizado.

```
package aula02;

   import java.io.IOException;
   import java.io.PrintWriter;
   import java.text.SimpleDateFormat;
   import java.util.Date;

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

   @WebServlet("/ServletDataHora")
   public class ServletDataHora extends HttpServlet {
   /**
   *
   */
   private static final long serialVersionUID = 1L;
   private SimpleDateFormat format =
   new SimpleDateFormat("dd/MM/yyyy - hh:mm:ss");

   @Override
   public void init() throws ServletException {
   super.init();
   log("ServletDataHora iniciado às "
   + format.format(new Date()));
}

@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
PrintWriter saida = response.getWriter();
saida.write("<HTML><BODY>Olá! A data/hora atual é: ");
saida.write(format.format(new Date()));
saida.write("</BODY></HTML>");
saida.close();
}

@Override
public void destroy() {
log("ServletDataHora finalizado às "
+ format.format(new Date()));
super.destroy();
}
}
```

Listagem 3 - Servlet que demonstra seu ciclo de vida

Para funcionar, o **ServletDataHora** possui um atributo do tipo **SimpleDateFormat**, utilizado para formatar datas e horas em um String de acordo com o padrão definido pelo programador, ou seja, segundo o formato "dd/MM/yyyy - hh:mm:ss". Esse objeto é primeiramente utilizado na implementação do método **init()**, o qual faz uso do método **log** (herdado de HttpServlet) para imprimir na saída padrão a data e hora na qual o Servlet foi inicializado, como pode ser visto na aba Console do Eclipse após iniciarmos o servidor web e acessarmos o Servlet (veja a Figura 7).

Figura 07 - ServletDataHora mostrando a data e hora atual formatadas



O código similar é visto no método **destroy()**. Observe que esses dois métodos devem executar os métodos sobrescritos **super.init()** e **super.destroy()**, visando garantir uma correta inicialização e liberação de recursos definidos no código herdado da superclasse.

Observe também o uso do **@override**, anotação no código Java que indica ao compilador que os métodos anotados estão sobrescrevendo métodos das superclasses. É uma boa prática usar esse tipo de anotação, pois dessa forma o compilador garante que o nome e tipos dos parâmetros dos métodos definidos estão de acordo com os da superclasse.

Você pode estar pensando agora: “Por que redefinimos o método **doGet()** e não o método **service()**?” A resposta para essa pergunta será dada mais adiante, ainda nesta aula.

Atividade 02

---

1. Defina a classe ServletDataHora como mostrado anteriormente e execute-a. Observe se as mensagens são impressas ao inicializar o Servlet, ao

executá-lo e ao finalizar a execução do servidor web. Reporte caso você encontre algum problema para executar a tarefa.

2. Altere a classe ServletDataHora para mostrar também o dia da semana atual (segunda-feira, terça-feira etc.). Dica: utilize a classe java.util.Calendar.

Métodos de execução dos Servlets

Até o momento, todos os Servlets que foram implementados redefiniram o método doGet() da classe HttpServlet. Para entendermos o porquê disso, precisamos estar cientes de que o **HTTP** (HyperTextTransferProtocol) é o protocolo mais utilizado na comunicação entre clientes e servidores web e ele pode ser utilizado de acordo com vários métodos (GET, POST, HEAD, PUT, etc.). Os métodos **GET e POST** são os mais utilizados e estão brevemente descritos no Quadro 1. Os detalhes sobre esses dois métodos não serão abordados neste curso.!

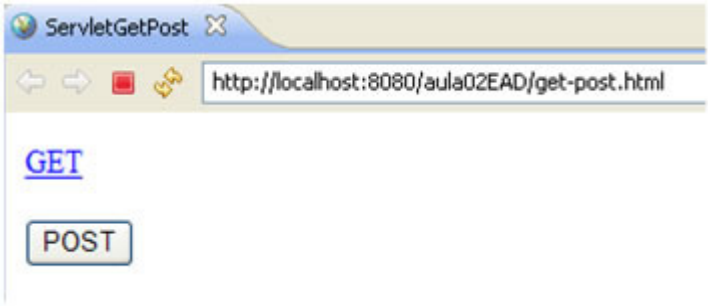
Quadro 1 - Métodos GET e POST do protocolo HTTP

Método	Descrição
GET	Método de acesso ao servidor web utilizado ao se: digitar uma URL em um navegador web; clicar em uma URL de uma página; acionar um formulário que contenha o atributo method="get". No caso, utiliza a própria URL para envio desses dados. Por exemplo, a URL <http://www.meusite.com/arquivo?p1=10> é utilizada para acessar o recurso <http://www.meusite.com/arquivo> passando-se o parâmetro p1 com valor 10.
POST	Método de acesso ao servidor web utilizado ao se acionar um formulário web que contenha o atributo method="post". No caso, os parâmetros e valores de acesso a um recurso web não aparecem na própria URL – eles são enviados pelo navegador ao servidor de forma separada.

A implementação do método **service()** que é dada pela classe HttpServlet reconhece o tipo de método utilizado na requisição web e delega a execução a métodos específicos de acordo com o método utilizado. O termo "delega" quer dizer que o método executa outro método, ou seja, deixando para o segundo o trabalho de processamento da requisição. No caso dos métodos GET e POST, o método service() delega a tarefa de processar as requisições para os métodos **doGet() e doPost()**, respectivamente.

Para você entender melhor esse funcionamento, observe o exemplo da Figura 8, o qual possui uma página HTML com um link e com um botão que levam à execução do mesmo Servlet. No caso, o link usa o método GET e o botão usa o método POST (observe os atributos **action** e **method** do marcador **form** no código da Listagem 4).

Figura 08 - Tela com link para acesso via método GET e botão para acesso via método POST



O código da página HTML vista na Figura 8 é mostrado a seguir.

```
1  <html>
2      <head>
3      <title>ServletGetPost</title>
4      </head>
5      <body>
6      <a href="ServletGetPost">GET</a><br>
7      <form action="ServletGetPost" method="post">
8      <input type="submit" value="POST">
9      </form>
10     </body>
11   </html>
```

Listagem 4 - Código da página HTML com link e botão

Já a implementação da classe **ServletGetPost** para atender às requisições acionadas pelo link ou botão dessa página é vista a seguir. Esse Servlet redefine tanto o método **doGet()** como o método **doPost()** de HttpServlet, apresentando essa informação na mensagem de resposta (veja a Figura 9).

```
1  package aula02;
2
3      import java.io.IOException;
4      import java.io.PrintWriter;
5
6      import javax.servlet.ServletException;
7      import javax.servlet.annotation.WebServlet;
8      import javax.servlet.http.HttpServlet;
9      import javax.servlet.http.HttpServletRequest;
10     import javax.servlet.http.HttpServletResponse;
11
12     @WebServlet("/ServletGetPost")
13     public class ServletGetPost extends HttpServlet {
14     private static final long serialVersionUID = 1L;
15
16     protected void doGet(HttpServletRequest request,
17     HttpServletResponse response)
18     throws ServletException, IOException {
19     PrintWriter saida = response.getWriter();
20     saida.write("<HTML><BODY>Olá! ");
21     saida.write("Mesma resposta para o método GET.");
22     saida.write("</BODY></HTML>");
23     saida.close();
24   }
25
26   protected void doPost(HttpServletRequest request,
27   HttpServletResponse response)
28   throws ServletException, IOException {
29   PrintWriter saida = response.getWriter();
30   saida.write("<HTML><BODY>Olá! ");
31   saida.write("Resposta para o método POST.");
32   saida.write("</BODY></HTML>");
33   saida.close();
34 }
35 }
```

Listagem 5 - Código do ServletGetPost

Figura 09 - Telas geradas pelo ServletGetPost ao usarmos requisição via método POST e GET, respectivamente

Se na hora de implementar um Servlet, você notar que não faz diferença entre os métodos GET e POST (esse é o caso mais comum!), então você pode implementar apenas um dos métodos (digamos, o GET) e fazer com que o outro chame o

primeiro. Isso pode ser observado no código do **ServletTantoFaz** apresentado a seguir.

```
1  package aula02;
2
3     import java.io.IOException;
4     import java.io.PrintWriter;
5
6     import javax.servlet.ServletException;
7     import javax.servlet.annotation.WebServlet;
8     import javax.servlet.http.HttpServlet;
9     import javax.servlet.http.HttpServletRequest;
10    import javax.servlet.http.HttpServletResponse;
11
12    @WebServlet("/ServletTantoFaz")
13    public class ServletTantoFaz extends HttpServlet {
14
15    private static final long serialVersionUID = 1L;
16
17    protected void doGet(HttpServletRequest request,
18    HttpServletResponse response)
19    throws ServletException, IOException {
20    doPost(request, response);
21 }
22
23 protected void doPost(HttpServletRequest request,
24 HttpServletResponse response)
25 throws ServletException, IOException {
26 PrintWriter saida = response.getWriter();
27 saida.write("<HTML><BODY>Olá! ");
28 saida.write("Mesma resposta para o GET ou POST.");
29 saida.write("</BODY></HTML>");
30 saida.close();
31 }
32
33 }
```

Listagem 6 - Código do ServletTantoFaz

Caso você não redefina os métodos doGet() ou doPost(), a sua implementação padrão é mostrar uma mensagem de erro padrão indicando que o método HTTP utilizado não é suportado.

Atividade 03

1. Crie a classe ServletGetPost e execute-a através da página HTML apresentada para acessar o Servlet através dos métodos GET ou POST. Reporte se você teve alguma dificuldade na tarefa.

2. Altere a classe ServletGetPost para apresentar a data atual se for usado o método GET, e a hora atual se for utilizado o método POST. Dica: baseie-se no código do ServletDataHora e altere a String do formato da data/hora para mostrar só a data ou só a hora. Ex.: "dd/MM/yyyy" para mostrar só a data.

3. Crie uma página HTML com um formulário e utilize-o para acessar o ServletMeuNome através do método POST. Analise se o Servlet respondeu como você estava esperando.

4. Altere o ServletMeuNome definido por você em atividades anteriores desta aula para que ele funcione tanto com o método GET como com o método POST.

5. Reescreva o ServletAgenda para permitir ambos os métodos POST e GET. Para testar, se necessário, altere o ServletPaginaAgenda para enviar os dados via método GET ou POST.

Leitura Complementar

Veja os métodos existentes nas classes Servlet, GenericServlet e HttpServlet usando a seguinte referência:

Servlet API Documentation.

- <http://tomcat.apache.org/tomcat-5.5-doc/servletapi/index.html>

Resumo

Para poder funcionar, os Servlets precisam ser registrados no arquivo web.xml ou receberem a anotação @WebServlet antes da definição da classe. Eles possuem um ciclo de vida. Primeiro são carregados, depois instanciados (quando o método init() é executado) e então executados (método service()) ao serem acessados pelos clientes Web. Quando o Servlet não é mais necessário, seu método destroy() é executado antes do objeto ser liberado da memória. A implementação do método service() é dada pela classe HttpServlet, que delega a execução a métodos de tratamentos específicos, como os métodos doGet() e doPost().

Autoavaliação

1. Descreva o ciclo de vida dos Servlets.
2. Identifique quais métodos da classe HttpServlet estão envolvidos no processamento de requisições web. Descreva como esses métodos são utilizados para processar as requisições feitas pelos clientes através dos navegadores web.
3. Descreva os passos necessários para configurar uma aplicação web para que o Servlet aluno.MeuServlet possa ser executado.
4. Crie e execute um Servlet que, dado um número, calcula e mostra o seu fatorial.

Referências

AHMED, K. Z.; UMRYSH, C. E. **Desenvolvendo aplicações comerciais em Java com J2EE J2EE e UML.** Rio de Janeiro: Ciência Moderna, 2003. 324 p.

CATTELL, Rick; INSCORE, Jim. **J2EE:** criando aplicações comerciais. Rio de Janeiro: Editora Campus, 2001.

HALL, Marty. **More Servlets and JavaServer Pages (JSP)**. New Jersey: Prentice Hall PTR (InformIT), 2001. 752 p. Disponível em: <http://pdf.moreservlets.com/>. Acesso em: 11 maio 2012.

HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages (JSP).** 2nd ed. New Jersey: Prentice Hall PTR (InformIT), 2003. 736p. (Core Technologies, 1). Disponível em: <http://pdf.coreservlets.com/>. Acesso em: 11 maio 2012.

HYPERTEXT Transfer Protocol: HTTP/1.1 – Methods Definition. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. Acesso em: 11 maio 2012.

J2EE Tutorial. Disponível em: <http://java.sun.com/javaee/reference/tutorials/>. Acesso em: 11 maio 2012.