JEE, TomCat, Servlet

Edson Orivaldo Lessa Junior

edson.lessa@unisul.br

HTML tag

- <html>
- <head>
- <title>Title of page</title>
- </head>
- <body>
- This is my first homepage.
- This text is bold
- </body>
- </html>

- **Editores WYSIWYG -** acrônimo da expressão em inglês “What You See
- Is What You Get”, que pode ser traduzido para “O que você vê é o que você recebe”.
- Oferecem ambiente de edição no qual o usuário vê o objeto no momento da edição na tela do computador já com a aparência do produto final. Ex.: FrontPage, Dreamweaver.

Servlets

- Servlets são classes Java instanciadas
- Executam em associação com servidores Web
- Respondem as requisições realizadas por meio do protocolo HTTP
- Os objetos Servlets podem enviar a resposta na forma de uma página HTML
- Servlet é uma API para construção de componentes do lado do servidor
- Objetivo é de fornecer um padrão para a comunicação entre clientes e servidores.

Servlets

- Servlets não possuem interfaces gráficas
- Instâncias são executadas dentro de um ambiente JAVA denominado Container
- Container gerencia as instâncias dos Servlets
- Provê os serviços de rede necessários para as requisições e respostas
- O container atua em associação com servidores Web recebendo as requisições re-encaminhadas por eles
- Os Servlets são tipicamente usados no desenvolvimento de sites dinâmicos

Benefícios

Desempenho

- Não há processo de criação para cada solicitação de cliente
- Solicitação é gerenciada pelo processo contentor de servlet
- Após o servlet finalizar o processamento de uma solicitação, ele permanece na memória, aguardando por outra solicitação.

Portabilidade

- São portáteis
- Pode-se mover para outros sistemas operacionais sem dificuldades.

Benefícios

- Servlets têm acesso à biblioteca JAVA
- Auxílio no processo de desenvolvimento.
- Robustez
  - Gerenciados pela JAVA VIRTUAL MACHINE
  - Não necessita controlar o uso de memória
  - Ou coleta de resíduos

Benefícios

- Aceitação difundida
  - JAVA é uma tecnologia amplamente aceita
  - Numerosos fabricantes trabalham com tecnologias baseadas em JAVA
  - Encontrar com facilidade componentes, que se ajustem às suas necessidades
  - Poupa-se tempo de desenvolvimento.

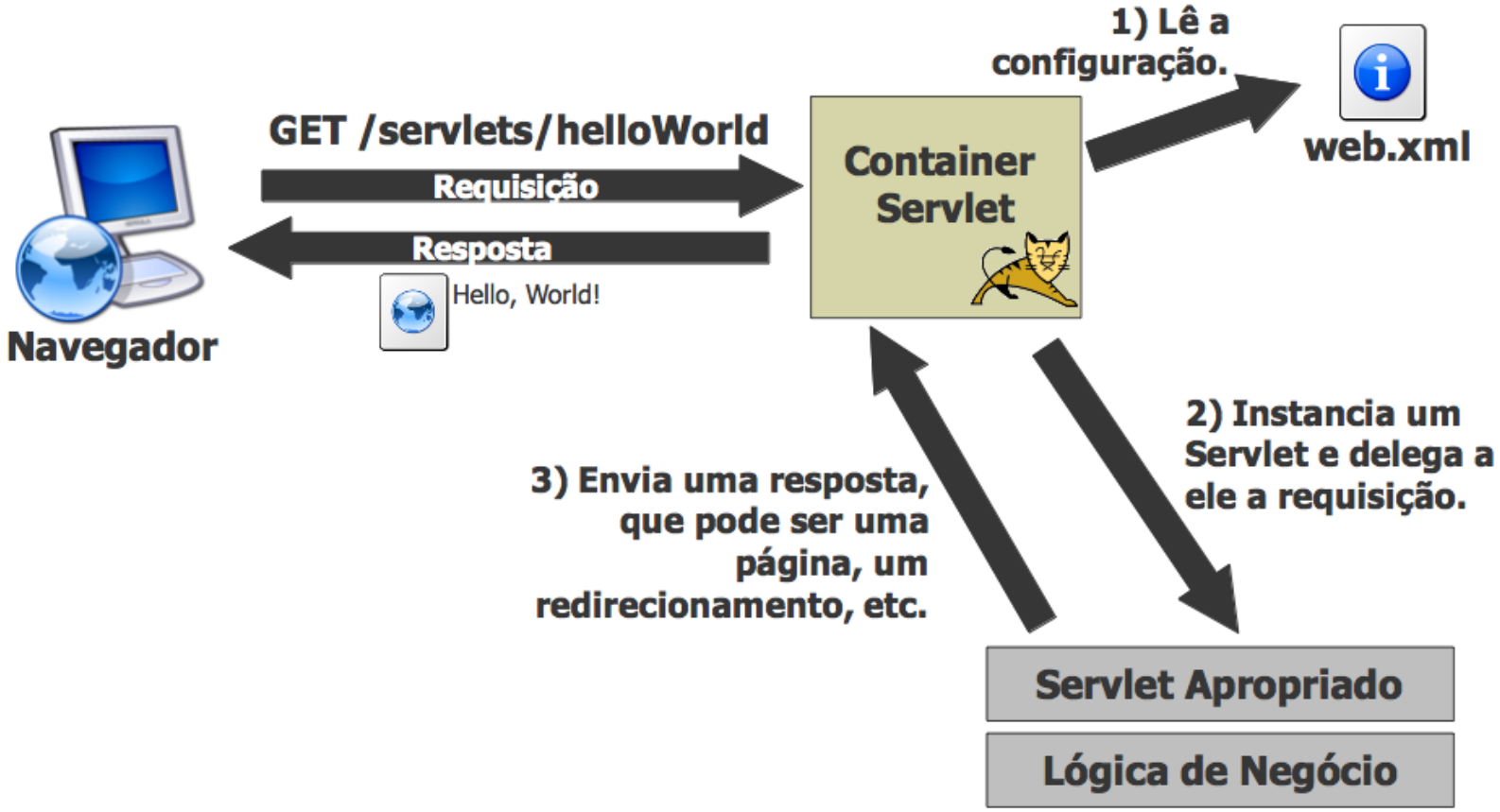
Como funciona um Servlet

- Um servlet é carregado pelo container servlet na primeira vez que é solicitado.
- É encaminhada a solicitação do usuário
- O servlet processa e retorna a resposta ao container servlet.
- O container servlet envia a resposta de volta ao usuário
- O servlet permanece na memória aguardando outras solicitações

Como funciona um Servlet

- Ele não é descarregado da memória
  - Somente quando o container de servlet veja uma diminuição de memória.
- A cada requisição ao servidor, o container de servlet compara o carimbo de horário do servlet carregado com o arquivo de classe servlet.
- Se o carimbo de horário de arquivo de classe for mais recente, o servlet é recarregado na memória.
- Dessa maneira, você não precisa reiniciar o container servlet sempre que atualizar o seu servlet.

Servlets


1) Lê a configuração.
web.xml
GET /servlets/helloWorld
Requisição
Resposta
Hello, World!
Navegador
Container Servlet
2) Instancia um Servlet e delega a ele a requisição.
3) Envia uma resposta, que pode ser uma página, um redirecionamento, etc.
Servlet Apropriado
Lógica de Negócio

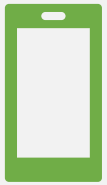
Web Application Resource

- As aplicações desktop possuem o padrão de distribuição JAR
- As aplicações web possuem o padrão WAR
- Arquivo com extensão .war
- Compactação da aplicação
- Independência de plataforma

Servlet Container

Servidor Web Java

Gerencia o ciclo de vida dos servlets

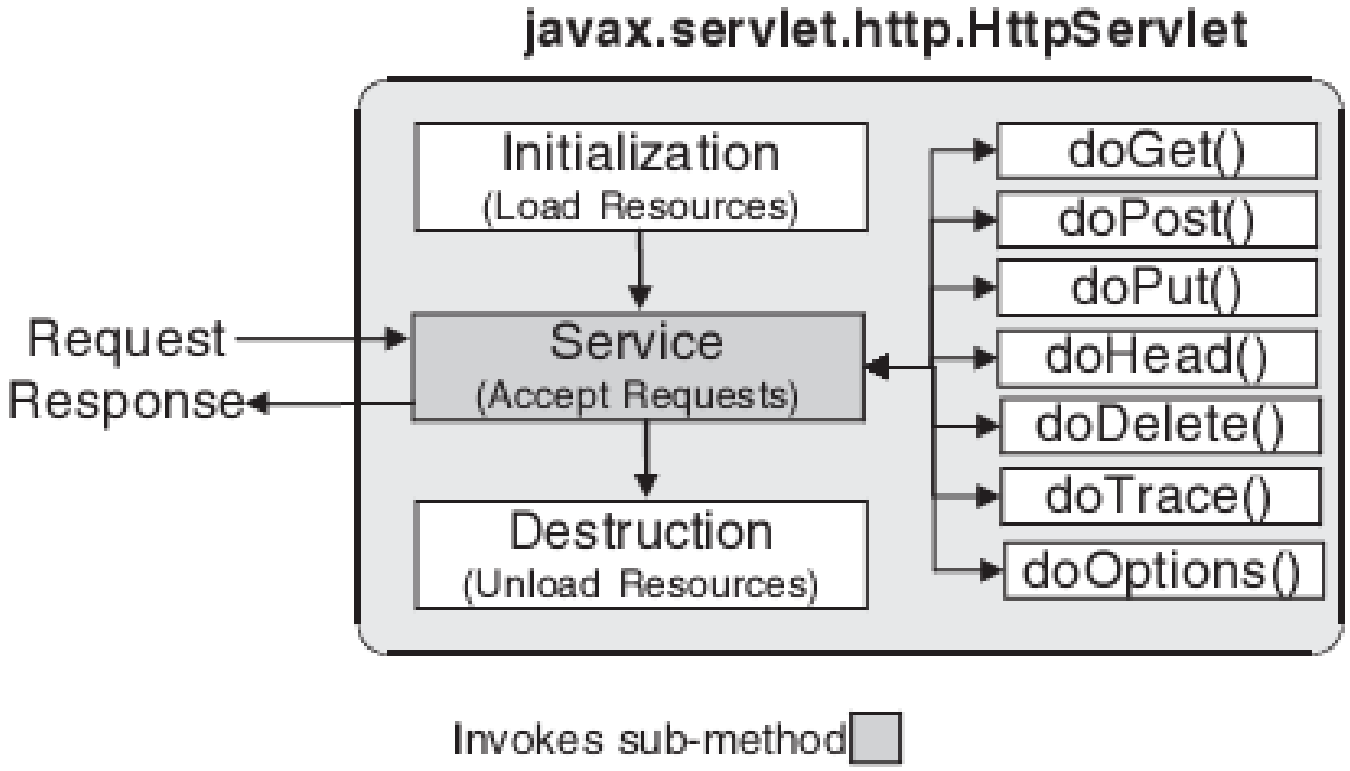


Apresenta parte das funcionalidades de um Servidor de Aplicação J2EE

Servlet

- Classe java que implementa a interface javax.servlet.Servlet
- Lib servlet-api.jar
- Web - javax.servlet.http.HttpServlet
  - init, service(), destroy().
  - doGet(), doPost, doPut(), etc.
  - HttpServletRequest e HttpServletResponse

Servlet – HttpServlet



Servlet – HttpServlet

- **doGet**: é o método mais simples do protocolo HTTP. Seu objetivo é requisitar ao servidor determinado recurso e enviá-lo de volta ao cliente (método doGet() da API de Servlets);
- **doPost**: é semelhante ao GET, porém é possível enviar dados para o servidor no corpo de uma mensagem HTTP. Com o método GET também podemos enviar informações ao servidor, mas na forma de parâmetros (método doPost() da API de Servlets) e está limitado a 255 caracteres;
- **doHead**: solicita apenas o header dos dados que o GET vai retornar. É como um GET sem corpo na resposta (método doHead() da API de Servlets);
- **doTrace**: pede um loopback da mensagem de solicitação. Permite assim que o cliente possa analisar o que está sendo recebido na outra ponta e usar estas informações para diagnóstico, teste ou troubleshooting (método doTrace() da API de Servlets);

Servlet – HttpServlet

- **doPut**: solicita para colocar a informação anexada (o corpo) na URL requisitada (método doPut() da API de Servlets);
- doDelete: solicita a remoção de um determinado recurso no servidor a partir da URL requisitada (método doDelete() da API de Servlets);
- **doOptions**: solicita uma lista de métodos HTTP, que podem ser respondidos pela URL requisitada (método doOptions() da API de Servlets);
- **doConnect**: é usado para efetuar tunneling. Se você precisa acessar um serviço, bloqueado pelo firewall, é possível criar um túnel entre a máquina cliente e a servidora para que se comuniquem entre si, passando assim pelo firewall. É importante lembrar que não existe um método doConnect() na API de Servlets.

Servlet x URL

- Os servlets são implementados como classe Java.
- Os navegadores entendem apenas a sintaxe de URL.
- Como fazer o mapeamento entre URL e Servlets?
  - web.xml

Mapeamento o web.xml

- Mapea-se a URL esecífica para ao servlet
- Utiliza-se o web.xml

```
<servlet>
  <servlet-name>primeiraServlet</servlet-name>
  <servlet-class>br.com.caelum.servlet.OiMundo</servlet-class>
</servlet>
```

- O mapeamento da url é feito na tag servlet-mapping

```
<servlet-mapping>
  <servlet-name>primeiraServlet</servlet-name>
  <url-pattern>/oi</url-pattern>
</servlet-mapping>
```

url-pattern

- url-pattern é responsável por dar a flexibilidade para disponibilizar uma servlet para várias URLS

```
<servlet-mapping>
  <servlet-name>primeiraServlet</servlet-name>
  <url-pattern>/oi/*</url-pattern>
</servlet-mapping>
```

- Configurando extensões de arquivos para suas servlets

```
<servlet-mapping>
  <servlet-name>primeiraServlet</servlet-name>
  <url-pattern>*.php</url-pattern>
</servlet-mapping>
```

Exemplo

```
package br.unisul.sequencial.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletBasico extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    // Recuperando o objeto Writer do HttpResponse para poder escrever o que será mostrado ao usuário.
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("<p>");
    out.println("Bem Vindo a página de Teste.");
    out.println("</p>");
    out.println("</body>");
    out.println("</html>");
  }
}
```

Exemplo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>servletBasico</servlet-name>
        <servlet-class>
 br.unisul.extensao.web.ServletBasico</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>servletBasico</servlet-name>
        <url-pattern>/servletBasico</url-pattern>
    </servlet-mapping>
</web-app>
```

Cuidados

- Esquecer da barra inicial no URL pattern:
  <url-pattern>oi</url-pattern>
- Digitar errado o nome do pacote da sua servlet:
  <servlet-class>br.unisul.servlet.OiMundo</servlet-class>
- Esquecer de colocar o nome da classe no mapeamento da servlet:
  <servlet-class>br.com.unisul.servlet</servlet-class>

Facilidades Servlets 3.0

- Utilizando anotação para definição de URL
- @WebServlet

```
@WebServlet("/oi")
public class OiServlet3 extends HttpServlet {
  ...
}
```

- Equivalente a configurar a Servlet acima com a url-pattern configurada como /oi.
- Parâmetro opcional chamado: name
  - equivalente ao servlet-name

Facilidades Servlets 3.0

- Definir mais de uma URL para acessar a Servlet utiliza-se: urlPatterns

```
@WebServlet(name = "MinhaServlet3", urlPatterns = {"/oi", "/ola"})
public class OiServlet3 extends HttpServlet{
  ...
}
```

- Servlet estando anotado com @WebServlet(), ele deve obrigatoriamente realizar um extends javax.servlet.http.HttpServlet
- Atributo chamado metadata-complete da tag <web-app> no web.xml
  - True as classe com anotação não serão procuradas pelo servidor de aplicação
  - False as classes na WEB-INF/classes ou em algum .jar dentro WEB-INF/lib serão examinadas

Facilidades Servlets 3.0

- @WebInitParam(): declara parâmetros como padrão para acessá-los dentro de um vetor

```
@WebServlet(
  name = "OiServlet3",
  urlPatterns = {"/oi"},
  initParams = {
    @WebInitParam(name = "param1", value = "value1"),
    @WebInitParam(name = "param2", value = "value2")}
  )
public class OiServlet3 {
  ...
}
```

Exemplo

- Hello World

Exemplo

```
package br.unisul.aula.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletBasico extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse
  response)
      throws ServletException, IOException {
    request.setAttribute("nome", "Edson");
    String nome =
      (String)request.getAttribute("nome");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("Nome:" + nome );
    out.println("</body>");
    out.println("</html>");

}
}
```

Formulários HTML

- Todo formulário em HTML é construído usando elementos dentro de um bloco <FORM>
- O bloco <FORM> define a URL que receberá o formulário e pode definir também o método usado

<FORM ACTION="URL para onde serão enviado os dados"
METHOD="método HTTP (pode ser GET ou POST)"
</FORM>

Envio de dados com Formulários

- Vários elementos HTML servem para entrada de dados e são usados dentro de formulários. Todos os elementos de entrada de dados têm um nome e enviam um valor
- Exemplo de formulário para entrada de dados

```
<FORM ACTION="/cgi-bin/catalogo.pl"
METHOD="POST">
<H3>Consulta preço de livro</H3>
<P>ISBN: <INPUT TYPE="text" NAME="isbn">
<INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

Formulários e links

- Formulários são similares a links.
- Um par formulário-botão tem o mesmo efeito que um link criado com <A HREF>
  - O link está no formulário e o evento no botão
- O bloco

  <FORM ACTION="/dados/tutorial.html">

  <INPUT TYPE="submit" VALUE="Tutorial">

  </FORM>

- gera a mesma requisição que

  <A HREF="/dados/tutorial.html">Tutorial</A>

- que é

  GET / dados/tutorial.html HTTP/1.0

Parâmetros de Request

- É comum a troca de informações (dados) entre uma página e outra. Esse tipo de troca de informações pode acontecer de duas formas:
  - GET: quando o parâmetro é passado na url, ex: http://localhost:8080/App/PrimeiroServlet?id=1&nome=jonas. Neste caso passamos dois parametro: id, com valor 1; e nome, com valor jonas;
  - POST: utilizado geralmente em formulários, onde pode ser enviado até Megabytes de informações. Os campos são passados de acordo com a postagem de campos em um formulário do tipo <form method="post">.
- Para obter os parâmetros, independetemente de enviados por GET ou POST, deve-se fazer o seguinte:

Parâmetros de Request

- Para obter os parâmetros, independetemente de enviados por GET ou POST, deve-se fazer o seguinte:

```
protected void doGet(HttpServletRequest
request, HttpServletResponse response)
                                    throws
ServletException, IOException {
                        int id =
Integer.parseInt(request.getParameter("id"));
                        String nome =
request.getParameter("nome");
                }
```

Atributos de Request

- Servlet1

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        Usuario usuario = usuarioBusiness.get(1);
        request.setAttribute("usuario", usuario);

        RequestDispatcher dispatcher = request.getRequestDispatcher("/Servlet2");
        dispatcher.forward(request, response);
}
```

Atributos de Request

- Servlet2

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        Usuario usuario = request.setAttribute("usuario", usuario);
}
```

Escrevendo - Response

- Através do HttpServletResponse é possível gerar a saída das informações no browser.
- response.setContentType(): seta o tipo da saída, exemplo: text/html;character-set:utf-8
- response.addCookie(): adiciona uma cookie que será enviada ao browser do cliente;
- response.getWriter(): retorna um objeto do tipo PrintWriter.

Escrevendo - Response

```
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<body>");
out.println("Helo world");
out.println("</body>");
out.println("</html>");
```

Exemplo

- Página HTML com formulário
- Servlet

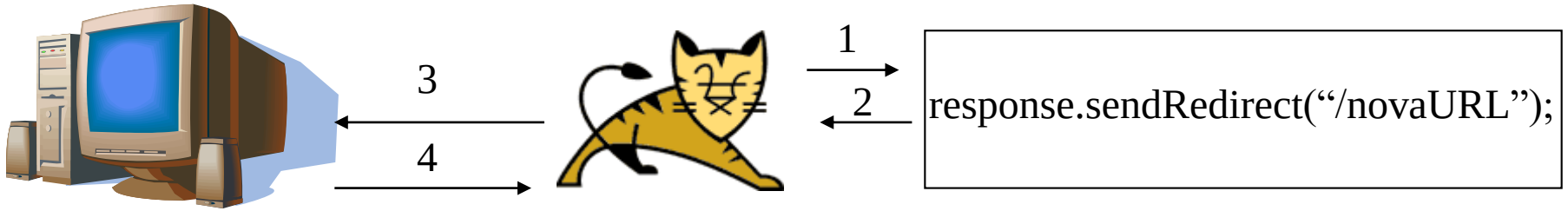
HttpSession

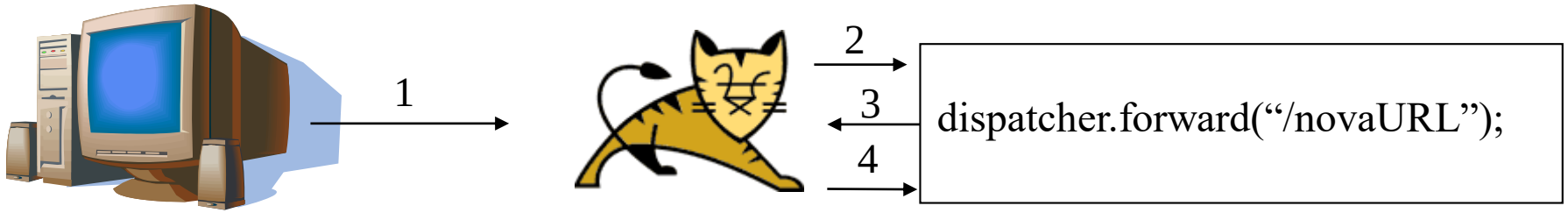
- Cada conexão de um cliente no servidor pode ou não possuir uma Sessão. Uma sessão é uma forma de armazenar temporariamente os dados do usuário (ex: login, carrinho de compras, etc.).
  - Uma sessão é inicializada ao chamar request.getSession(true). Caso já exista uma sessão, será retornada sem que uma nova seja criada;
  - Para adicionar atributos na sessão utiliza-se request.getSession().setAttribute("nome", "valor"). O valor pode ser qualquer tipo de dados (Object);
  - Igualmente, para obter um atributo request.getSession().getAttribute("nome"), e remove-lo: request.getSession().removeAttribute("nome");
  - Uma sessão ficará viva até que aconteça um timedout (usuário nao acessar a aplicação, ou seja, ficar inativo); ou ainda, quando chamado o método invalidate();

HttpSession

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {


                HttpSession session = request.getSession(true); // se não existe, então cria
                session.setAttribute("usuario", new String("Nome Do Usuario"));

                session.getAttribute("usuario"); // retorna Nome Do Usuario

                session.removeAttribute("usuario");

        }
```

Forward x Redirect





Exemplo

```
protected void doGet(HttpServletRequest request,HttpServletResponse
   response) {       response.sendRedirect("/novaURL");
   }
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) {
        ServletContext context = getServletContext();
        RequestDispatcher dispatcher =
        context.getRequestDispatcher("/novaURL");
        dispatcher.forward(request, response);
   }
```

Exercícios

- Faça um aplicativo que envie dados de um formulário de cadastro de Usuários, utilizando o método POST, para um Servlet. Este servlet deverá fazer a inclusão dos dados em uma Lista.
- Após inserir, o usuário deverá visualizar a lista completa de usuários;