

ORIENTAÇÃO A OBJETOS

AULA 13

Introdução de Java na Web - Servlet

Vandor Roberto Vilardi Rissoli



APRESENTAÇÃO

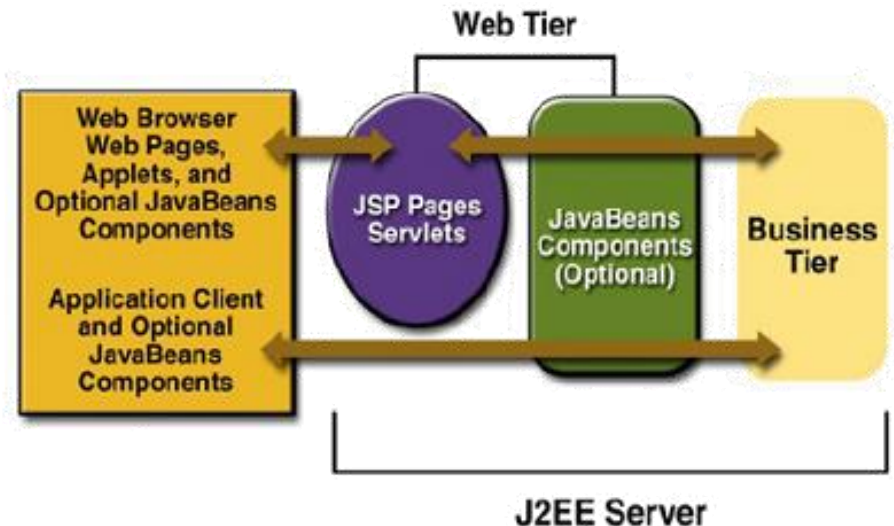
- *Servlet* - fundamentação
- Container
- Ciclo de Vida
- Recursos de Classes e Interfaces
- Referências



Servlet

Servlet é uma das tecnologias elaboradas pela *Sun* para o desenvolvimento de aplicações Web a partir de recursos Java que executem no servidor.

- Classes Java que são instanciadas e executadas em associação com servidores Web, atendendo as requisições do protocolo HTTP
- Expressão representativa de “pequeno servidor”;
- Corresponde a uma das camadas do modelo Web de Multicamadas;



Servlet

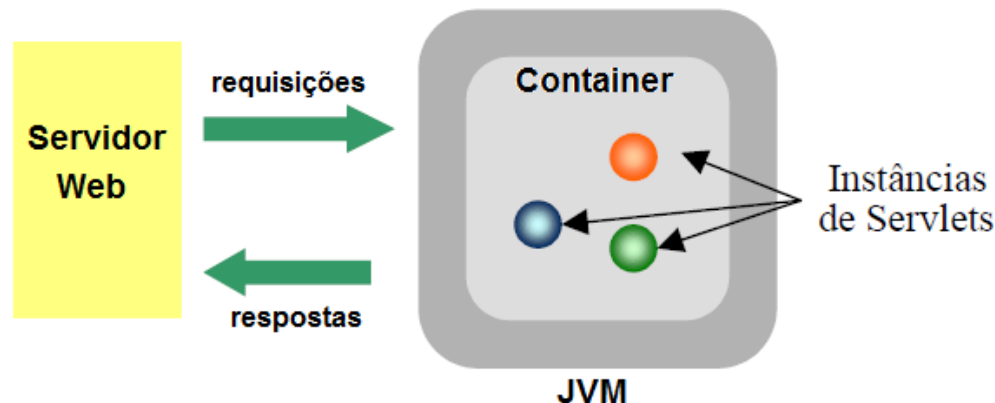
- API dos Servlets não assume nada a respeito do ambiente do servidor, sendo **independentes** de protocolos e plataformas;
- API para construção de componentes do lado servidor com o objetivo de fornecer um **padrão para comunicação** entre clientes e servidor;
- Objetivos:
 - Ler dados contidos nas requisições passadas ao servidor e gerar uma **resposta dinâmica**;
 - Implementa características especiais para o HTTP;
 - Produz saídas imprimindo caracteres como cadeias no fluxo de saída, sendo esta a resposta HTTP;



Servlet

CONTAINER

- Servlets não possuem interface gráfica e suas instâncias são executadas dentro de um ambiente Java denominado de Container;
- Gerencia as instâncias dos Servlets e provê os serviços de rede necessários para req./respostas;
- O Container atua em associação com servidores, recebendo requisições, reencaminhadas por eles.



Servlet

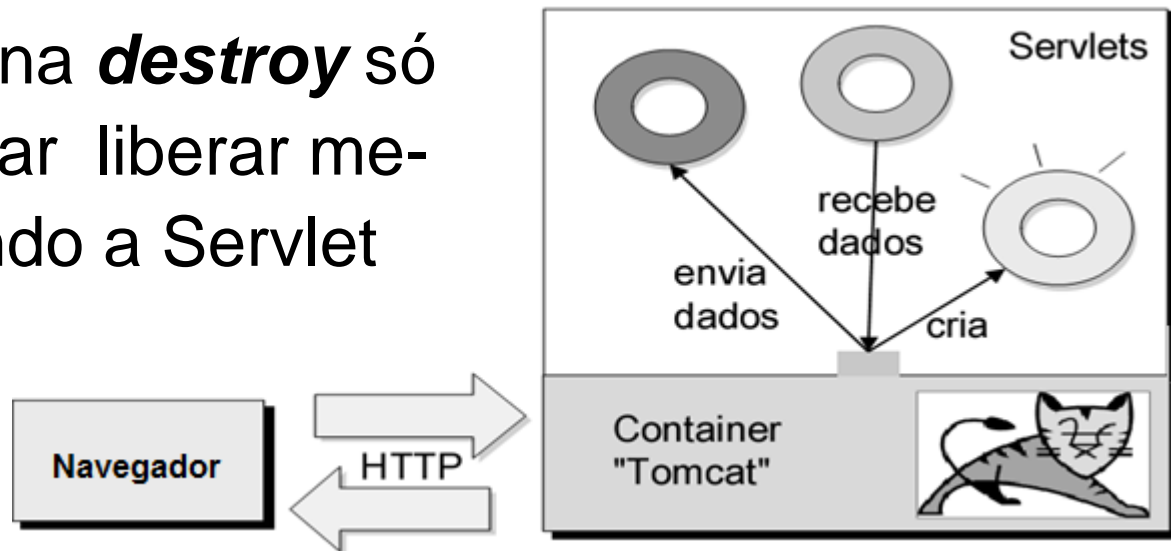
- Geralmente, existe apenas uma instância de cada Servlet, no entanto, o Container pode criar vários ***threads*** de modo a permitir que uma única instância Servlet atenda mais de uma requisição simultaneamente.
- Trabalha com as características típicas do HTTP como métodos GET, POST, PUT, Cookies, etc.



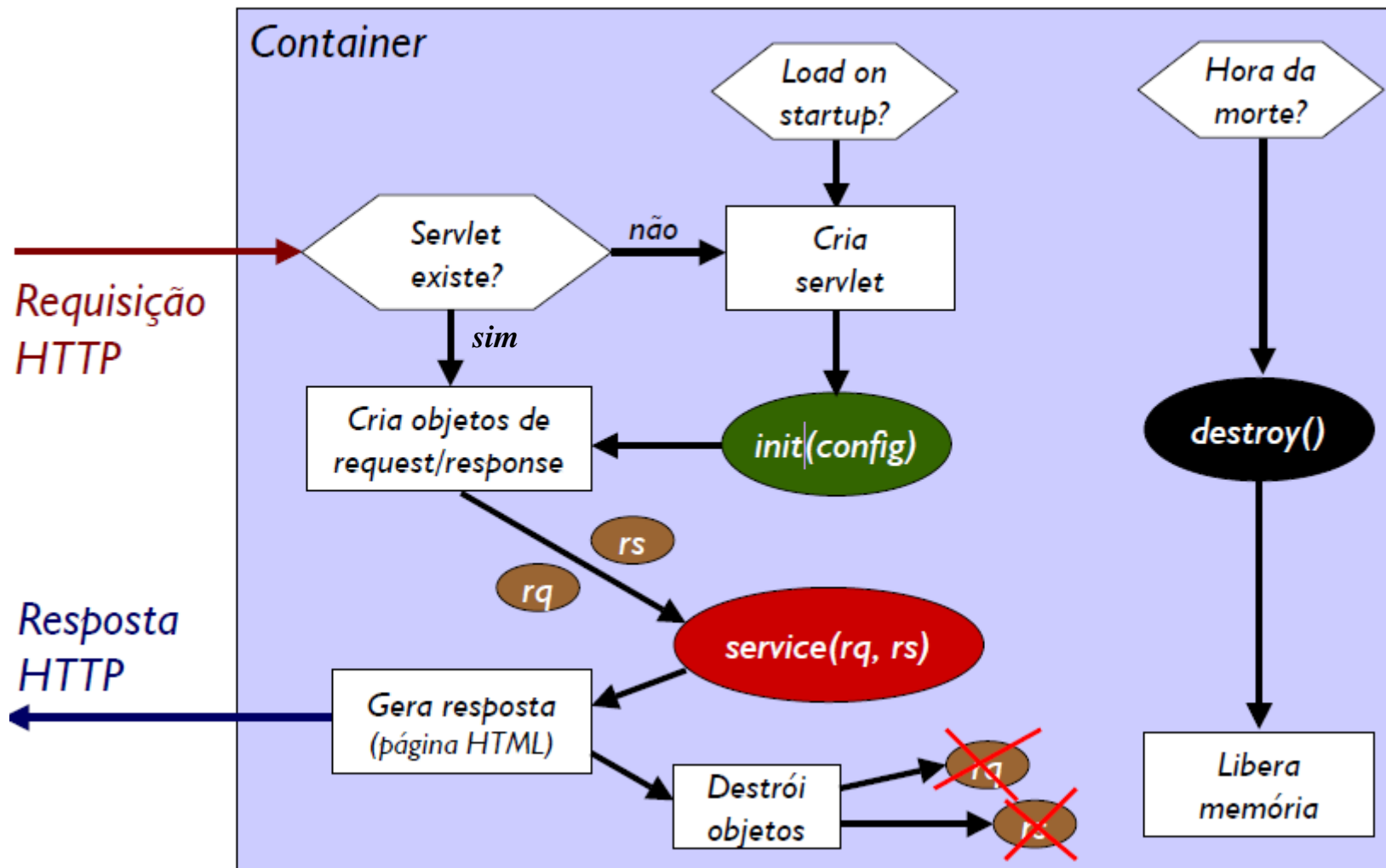
Servlet

Ciclo de Vida da Servlet

- Ciclo controlado pelo Container;
- Servidor recebe a requisição e reencaminha para o container que a repassa para a Servlet;
- Container cria objetos de Req./Resposta e aciona o método **service** passando parâmetros;
- Destrói objetos quando resposta é enviada;
- Container aciona **destroy** só quando desejar liberar memória finalizando a Servlet



Servlet



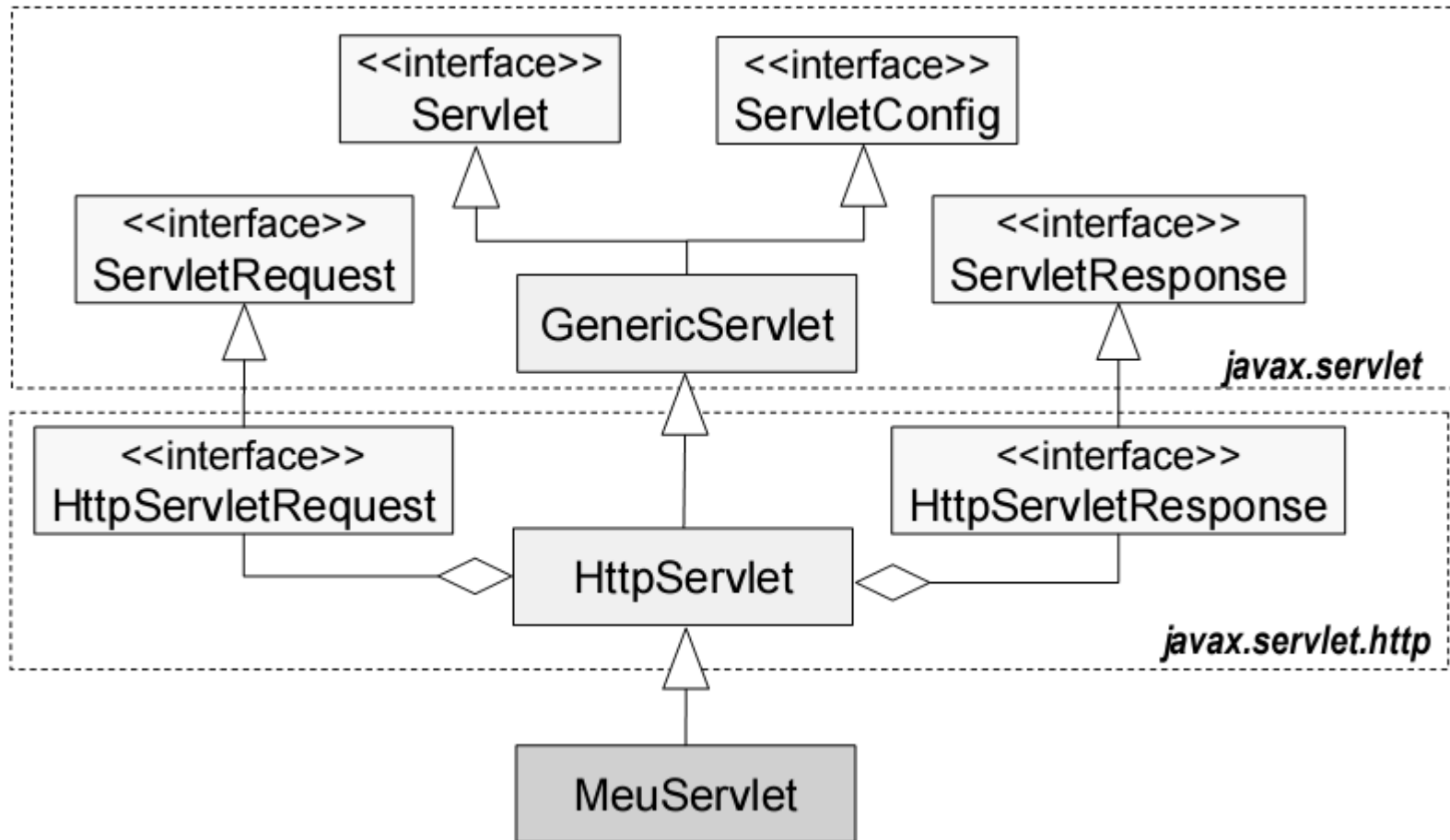
Servlet

API Servlet

- É composta por conjuntos de Interfaces e Classes;
- Seu componente mais básico é a interface Servlet;
- Ele define o comportamento básico de um Servlet;
- Métodos a serem implementados:
 - **service()** - responsável pelo tratamento de todas das requisições dos clientes;
 - **init()** - acionado quando o Servlet é carregado;
 - **destroy()** - descarregado do container;
 - **getServletConfig()** retorna objeto ServletConfig que contém os parâmetros de inicialização do Servlet; (ainda possui outros métodos)

Servlet

- O restante dessa API se organiza hierarquicamente a partir da interface Servlet



Servlet

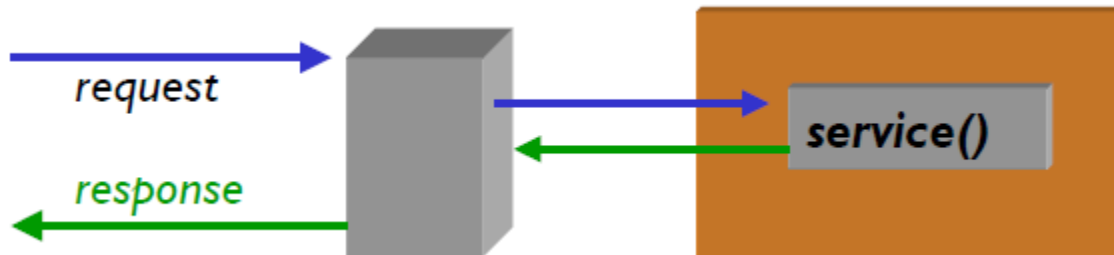
Método Abstrato *service*

- Método que implementa operações de resposta executadas quando o cliente envia uma requisição
 - Esse método sempre recebe 2 parâmetros, sendo de ***ServletRequest*** e ***ServletResponse***
- Tarefas usuais do método *service*:
 - extrair informações da requisição
 - acessar recursos externos
 - preencher a resposta (para HTTP é preencher os cabeçalhos de resposta, obter um *stream* de resposta e escrever os dados no *stream*)



Servlet

- Definido em *javax.servlet.Servlet*
public void service(ServletRequest, ServletResponse);
- Sempre que um servidor repassar uma requisição para um Servlet, ele acionará o método *service*;
- Um servlet genérico deverá sobrepor este método e usar seus objetos enviados por parâmetros (*ServletRequest* , *ServletResponse*) para ler os dados da requisição e formar a resposta



Servlet

- Para rodar um Servlet é preciso implementar um ***Container***
- Para serviços Web deve-se usar Servlets HTTP
 - API específica para lidar com o HTTP
 - Método *service()* dividido e específico para tratar os diferentes métodos do HTTP, sendo os mais utilizados **Get** e **Post** (entre outros)
- HttpServlet redireciona os pedidos encaminhados ao *service()* para métodos correspondentes HTTP

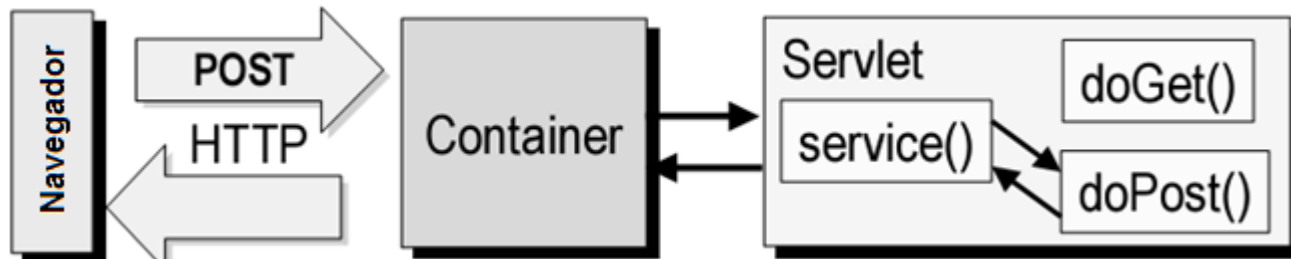
```
public void doGet(HttpServletRequest, HttpServletResponse)
```

```
public void doPost(HttpServletRequest, HttpServletResponse)
```



Servlet

- Um Servlet HTTP genérico deverá estender HttpServlet e implementar pelo menos um dos métodos doGet() ou doPost()



- Uma requisição HTTP feita pelo navegador (cliente) tipicamente contém vários cabeçalhos e os métodos de HttpServletRequest permitem extrair dados de qualquer um deles

```
GET /docs/index.html HTTP/1.0
Connection: Keep-Alive
Host: localhost:8080
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
```

Servlet

Alguns métodos de *HttpServletRequest*

- Enumeration `getHeaderNames()` - obtém nomes dos cabeçalhos
- String `getHeader("nome")` - obtém primeiro valor do cabeçalho
- Enumeration `getHeaders("nome")` - todos os valores do cabeçalho
- String `getParameter(param)` - obtém parâmetro HTTP
- String[] `getParameterValues(param)` - obtém parâmetros repetidos
- Enumeration `getParameterNames()` - obtém nomes dos parâmetros
- Cookie[] `getCookies()` - recebe cookies do cliente
- HttpSession `getSession()` - retorna a sessão
- `setAttribute("nome", obj)` - define um atributo obj chamado "nome".
- Object `getAttribute("nome")` - recupera atributo chamado nome
- String `getRemoteUser()` - obtém usuário remoto (se autenticado, caso contrário devolve null)
- entre outros...

Servlet

- Uma resposta HTTP é enviada pelo servidor ao navegador (cliente) e contém informações sobre os dados anexados

```
HTTP/1.0 200 OK
Content-type: text/html
Date: Mon, 7 Apr 2003 04:33:59 GMT-03
Server: Apache Tomcat/4.0.4 (HTTP/1.1 Connector)
Connection: close
Set-Cookie: jsessionid=G3472TS9382903

<HTML>
  <h1> Página Web de Resposta </h1>
</HTML>
```

- Os métodos de ***HttpServletResponse*** permitem construir um cabeçalho



Servlet

Alguns métodos de *HttpServletResponse*

- `addHeader(String nome, String valor)` - adiciona cabeçalho HTTP
- `setContentType(tipo MIME)` - define o tipo MIME que será usado para gerar a saída (text/html, image/gif, etc.)
- `sendRedirect(String location)` - envia informação de redirecionamento para o cliente (Location: url)
- `Writer getWriter()` - obtém um `Writer` para gerar a saída. Ideal para saída de texto.
- `OutputStream getOutputStream()` - obtém um `OutputStream`. Ideal para gerar formatos diferentes de texto (imagens, etc.)
- `addCookie(Cookie c)` - adiciona um novo cookie
- `encodeURL(String url)` - envia como anexo da URL a informação de identificador de sessão (sessionid)
- `reset()` - limpa toda a saída inclusive os cabeçalhos
- `resetBuffer()` - limpa toda a saída, exceto cabeçalhos
- entre outros...

Servlet

- Use **doGet()** para receber requisições GET
 - Links clicados ou URL digitadas diretamente
 - Alguns formulários que usam GET
- Use **doPost()** para receber dados de formulários
- Não sobreponha *service()* para implementar os 2 métodos, mas os implemente - **doGet()**, **doPost()**

```
public class ServletWeb extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response) {  
        processar(request, response);  
    }  
    public void doPost (HttpServletRequest request,  
                      HttpServletResponse response) {  
        processar(request, response);  
    }  
    public void processar(HttpServletRequest request,  
                          HttpServletResponse response) {  
        ...  
    }  
}
```

Servlet

```
package aula13.exemplo;    // criando uma classe em Java Resources
import javax.servlet.http.HttpServlet;
    // Possui vários imports (digite CTRL+SHIFT+O) para fazer todos
public class MeuServlet extends HttpServlet {
    protected void service(HttpServletRequest request,
                           HttpServletResponse response)
        throws ServletException, IOException {
    // Escritor de saída (resposta)
    PrintWriter saida = response.getWriter();
    // Resposta para Requisição
    saida.println("<html>");
    saida.println("<header>");
    saida.println("<title>Meu Primeiro Servlet</title>");
    saida.println("</header>");
```

Servlet

// Continuação...

```
saida.println("<body>");
saida.println("<h1>Funcionou meu primeiro Servlet.</h1>");
saida.println("</body>");
saida.println("</html>");
}
}
```



Servlet

Configuração para o Container (arquivo web.xml)
(arquivo localizado na pasta WEB-INF no WebContent)

<!-- Primeiro Servlet sem dados dinamicos -->

<servlet>

<servlet-name>primeiroServlet</servlet-name>

<servlet-class>aula13.exemplo.MeuServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>primeiroServlet</servlet-name>

<url-pattern>/primeiro</url-pattern>

</servlet-mapping>



Servlet

Envolvendo a criação e passagem de parâmetros as páginas ficarão mais dinâmicas. Assim, crie a nova classe a seguir:

// Mantenha no mesmo projeto do Eclipse

```
public class TestaParametros extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
                           HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        PrintWriter writer = response.getWriter();  
        writer.println("<html>");  
        writer.println("Recebendo parâmetro = " +  
                        request.getParameter("idade"));  
        writer.println("</html>");  
    }  
}
```

Servlet

É necessário criar um arquivo HTML para interação com o usuário e fornecimento de um valor que será o parâmetro.

// Mantenha no mesmo projeto do Eclipse

```
<html>
  <head>
    <title>Com Parametro</title>
  </head>
  <body>
    <a href="/jspteste/testaIdadeInt?idade=25">Clique parâmetro</a>
  </body>
</html>
```

Quando não se identifica o método de comunicação será assumido o valor padrão que é o **GET**

Servlet

Configuração para o Container (arquivo web.xml)

(atualizando a pasta WEB-INF no WebContent)

`<!-- Servlet com parâmetro idade -->`

```
<servlet>
```

```
    <servlet-name>idade</servlet-name>
```

```
    <servlet-class>aula13.exemplo.TestaParametros</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
    <servlet-name>idade</servlet-name>
```

```
    <url-pattern>/testaIdade</url-pattern>
```

```
</servlet-mapping>
```



Servlet

Elabore uma nova página web (HTML) com um formulário para leitura desta idade pelo usuário e sua apresentação.

// Mantenha no mesmo projeto do Eclipse

```
<html>
  <head>
    <title>Parametro com Formulario</title>
  </head>
  <body>
    <form action="/jspteste/testaIdade" method="POST">
      <input type="text" name="idade" value="0"/>
      <input type="submit" value="Enviar"/>
    </form>
  </body>
</html>
```

Servlet

Veja que realmente não existe o método **doPost** para ser executado na Servlet.

// Mantenha no mesmo projeto do Eclipse

```
public class TestaParametros extends HttpServlet {  
    protected void doPost(HttpServletRequest request,  
                           HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        PrintWriter writer = response.getWriter();  
        writer.println("<html>");  
        writer.println("Recebido do Formulário = " +  
                        request.getParameter("idade"));  
        writer.println("</html>");  
    }  
}
```

Servlet

Quando existir uma situação como esta, a lógica da resposta será a mesma para requisição GET ou POST, é possível acionar uma da outra e não permitir tal erro.

```
public class TestaParametros extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, IOException  
    {  
        PrintWriter writer = response.getWriter();  
        writer.println("<html>");  
        writer.println("Dados= " + request.getParameter("idade"));  
        writer.println("</html>");  
    }  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, IOException  
    {  
        doGet(req, resp)  
    }  
}
```

Exercícios de Fixação

- 1) Faça um formulário que tenha a capacidade de registrar a sua matrícula e nome completo, apresentando estes dados depois de lidos em uma página para o usuário confirmar seu registro correto. Esta solução deverá estar disponível para execução pela Internet (páginas web).
- 2) Suponha que você deseje armazenar o nome do seu time de futebol preferido e quantos títulos internacionais ele já ganhou. Para isso, implemente uma classe de dados com as propriedades de programação orientada a objeto e somente se o usuário fizer um registro válido você deverá apresentar os dados para conferência dele. Caso algum dado seja inválido seu programa deverá terminar informando que o “Cadastro não pode ser efetuado!”. Esta solução deverá estar disponível para execução pela Internet (páginas web).



Referência de Criação e Apoio ao Estudo

Material para Consulta e Apoio ao Conteúdo

- OLIVEIRA, A. P. Apostila Servlet/JSP. Universidade Federal de Viçosa. 2001.
 - <http://www.cin.ufpe.br/~wsr/ApostilaServletJSP.pdf>
- ROCHA, H. 2 Servlet. J550.
 - http://www.argonavis.com.br/cursos/java/j550/j550_2.pdf
- Caelum Ensino e Inovação – site com apostila HTML
 - <http://www.inf.ufrgs.br/~tsrodrigues/utilidades/javaWeb.pdf>
- Universidade Católica de Brasília – site
 - <http://cae.ucb.br/conteudo/programar>
(escolha no menu superior a opção Programação Computacional seguida de **Laboratórios** e **LAB III**)

