DSD I — Introdução a Servlets e JSP

Flávio Velloso Laper

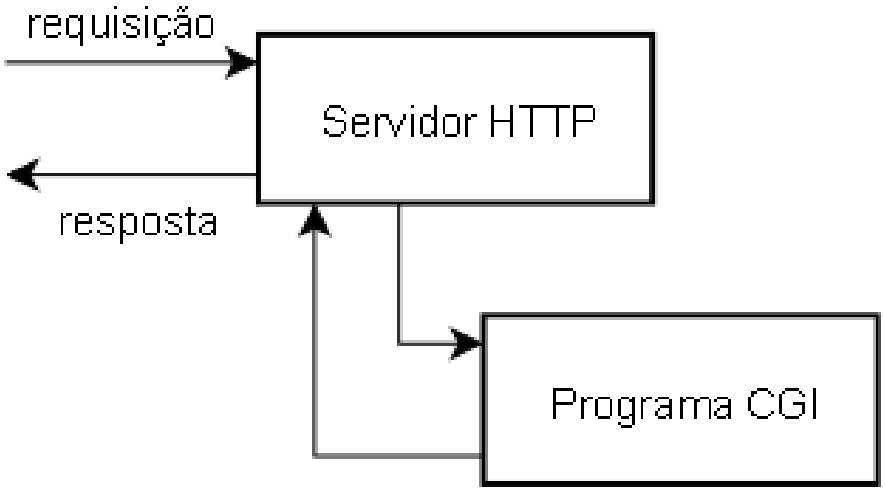
19 de agosto de 2019

Tecnologias — lado servidor

- Estendem as funções básicas de um servidor HTTP.
- Tipos:
  - CGI (Common Gateway Interface).
  - API's de Servidor: ISAPI, NSAPI, APACHE API, Servlets.
  - Scripts: ASP, JSP, PHP, etc.
- Rodam do lado do servidor (não recebem suporte por parte dos navegadores).
- Interceptam o curso normal da comunicação:
  - Recebem dados via requisição HTTP.
  - Devolvem dados via resposta HTTP.

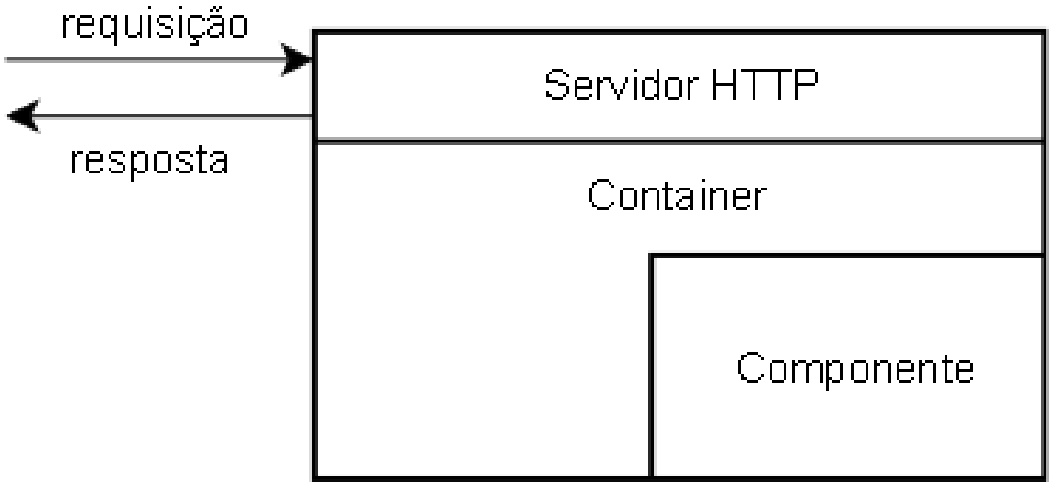
CGI

- Especificação que determina como construir uma aplicação que será executada pelo servidor HTTP.
- Programas CGI podem ser escritos em qualquer linguagem.
- A especificação determina apenas os formatos de entrada e saída.
- CGI requer que o servidor execute um processo a cada requisição:
  - Consome muitos recursos do sistema operacional.
  - Processos externos não têm acesso aos recursos do servidor.
  - O compartilhamento de dados é difícil.



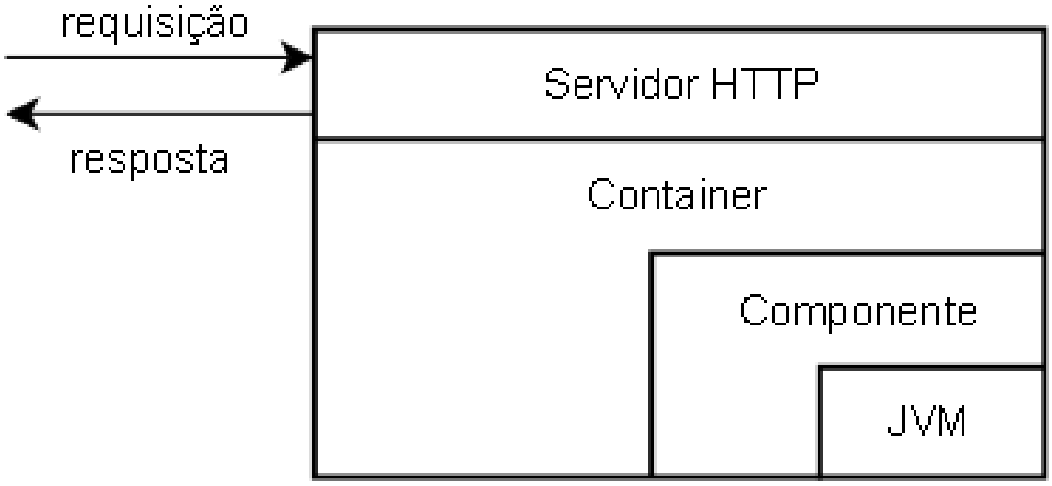
API's de Servidor

- Substituem o CGI com vantagens:
  - Toda a funcionalidade do servidor pode ser usada.
  - Em geral, utilizam processos internos (threads).
  - São mais rápidos e eficientes.
- Desvantagens:
  - Dependem de plataforma, fabricante e linguagem.
  - São soluções proprietárias.
- Exemplos: ISAPI, APACHE API, Servlets, etc.



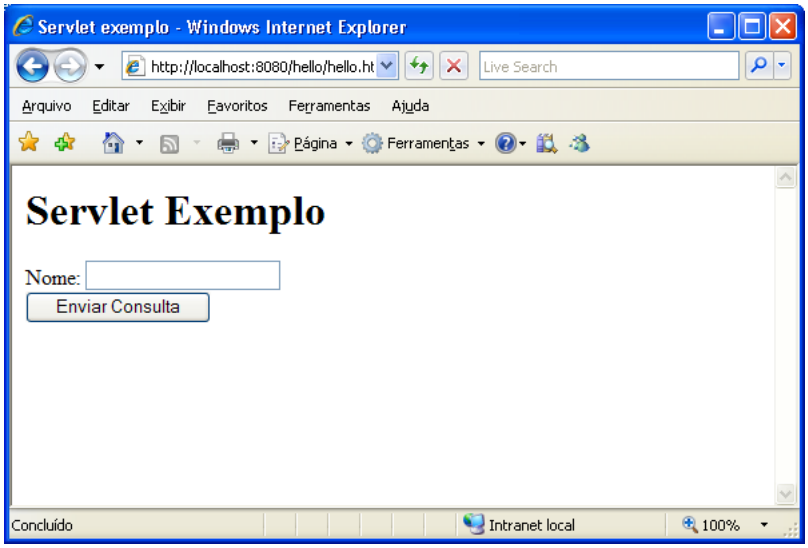
Servlets

- API desenvolvida para ser independente de plataforma e (praticamente) independente de fabricante.
- Componentes são escritos em Java e se chamam Servlets.
- Rodam dentro de um servidor através de uma JVM.
- Nativo em diversos servidores, disponível como plugin para outros.



Exemplo (servlet)

- Suponha-se o seguinte formulário:



- O código de um servlet que trata as submissões do formulário encontra-se abaixo (o servlet apenas exibe "Hello Fulano" onde "Fulano" é o nome informado, ou "Hello World" se o nome for deixado em branco).

```
public class HelloServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServlestException,IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("text/html");
    String nome = request.getParameter("nome");
    if (nome == null)
      nome = "World";
    out.println("<html><head><title>");
    out.println("Servlet exemplo");
    out.println("</title></head><body>");
    out.println("<h1>Servlet Exemplo</h1>");
    out.println("Hello" + nome);
    out.println("</body></html>");
    out.close();
  }
}
```

Scripts do servidor

- Colocam a linguagem de programação dentro do HTML (e não o contrário).
- Exemplos: ASP, JSP, PHP, COLD FUSION.
- A página possui um nome com extensão diferente que a identifica como SCRIPT.

Exemplo (JSP)

- O trecho de código abaixo representa um JSP com a mesma funcionalidade do servlet acima.

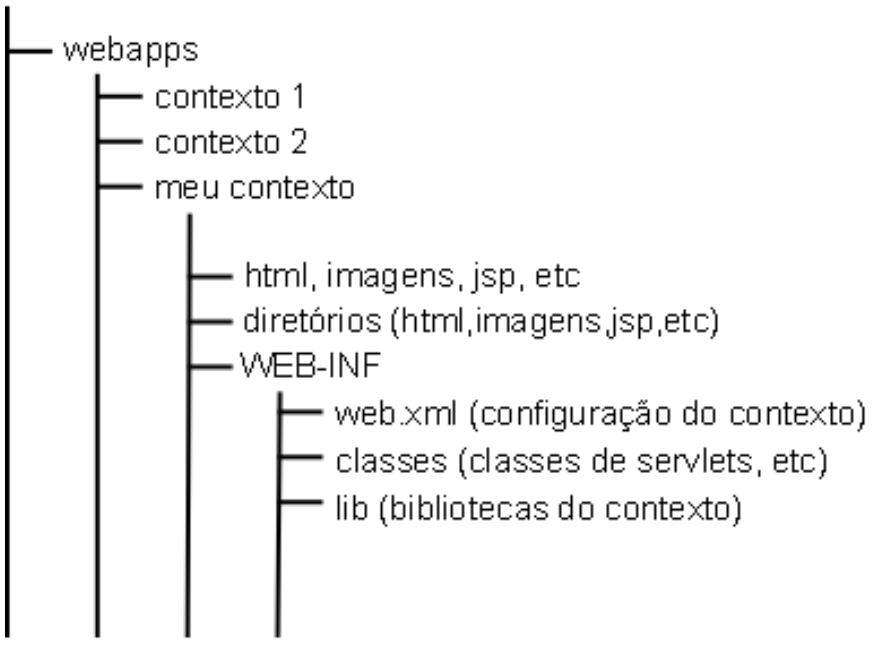
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

```
<html>
 <head><title>JSP exemplo</title></head>
<body>
  <h1>JSP exemplo</h1>
  <% String nome = request.getParameter("nome");
     if(nome == null)
        nome = "World";
  %>
   Hello <%= nome %>
</body>
</html>
```

Tomcat

- O Apache Tomcat é a implementação de referência para servlets e jsp´s.
- Alguns diretórios importantes:
  - <tomcat>/bin : executáveis.
  - <tomcat>/conf : arquivos de configuração.
  - <tomcat>/lib : bibliotecas comuns.
  - <tomcat>/webapps : contextos (aplicações).

Estrutura de um contexto (dentro de webapps)



- O diretório WEB-INF, com todo o seu conteúdo, delimita a *área privativa* do contexto (não é possível acessá-lo externamente).
- Os recursos nos demais diretórios podem ser diretamente acessado por requisições HTTP simplesmente informando o caminho correspondente.
- Exemplo: para acessar um arquivo HTML denominado *meu.html* na raiz do contexto acima, o caminho é:

  http://meuhost:8080/meucontexto/meu.html
- As classes correspondentes aos servlets devem ser publicadas dentro do diretório *classes* em WEB-INF. Os subdiretórios de *classes* devem espelhar a estrutura de pacotes das classes da aplicação.

- Um servlet deve ser mapeado no arquivo *web.xml* para que possa ser acessado:

```
<web-app>
    <servlet>
        <servlet-name> Hello </servlet-name>
        <servlet-class> pacote.HelloClasse </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name> Hello </servlet-name>
        <url-pattern> /hello </url-pattern>
    </servlet-mapping>
</web-app>
```

- Observações
  - A tag *servlet* cadastra o servlet no contexto, informando um identificador (*servlet-name*) e a classe que o implementa (*servlet-class*).
  - A tag *servlet-mapping* cadastra a url que levará as requisições até o servlet, informando o mesmo identificador acima (*servlet-name*) e o padrão de url desejado (*url-pattern*).

Implantação de Recursos em um contexto existente

1. Páginas JSP (e outros recursos estáticos):
   (a) Copiar o recurso para o diretório desejado na área publica do contexto. (Ex: contexto→jsp→hello.jsp).
   (b) Acessar informando o caminho do recurso (Ex: http://dominio:porta/contexto/jsp/hello.jsp). Não é necessário reiniciar o contexto.
2. Servlets: Utilizar o seguinte procedimento (Classe de exemplo: Hello.java):
   (a) Compilar a classe: javac -cp <caminho de *servlets-api.jar*> Hello.java.
   Obs: se necessário, configurar o *path* do sistema operacional.
   (b) Copiar arquivo compilado para a estrutura correta no diretório de classes: Ex: contexto→WEB-INF→classes→Hello.class
   (c) Configurar o servlet no arquivo *web.xml* (tal como indicado acima).
   (d) Reiniciar o contexto e acessar com o caminho mapeado.

Criação de um novo contexto com um WEB ARCHIVE

Utilizar o seguinte procedimento:

1. Criar em um diretório separado (fora do container) uma estrutura que reproduza a estrutura de diretórios de um contexto.
2. Implantar nesta estrutura os recursos desejados, tal como descrito anteriormente.
3. Criar um arquivo ".war":
   cd contexto
   jar -cvf contexto.war .
4. Copiar o arquivo gerado para a pasta *webapps* do container.