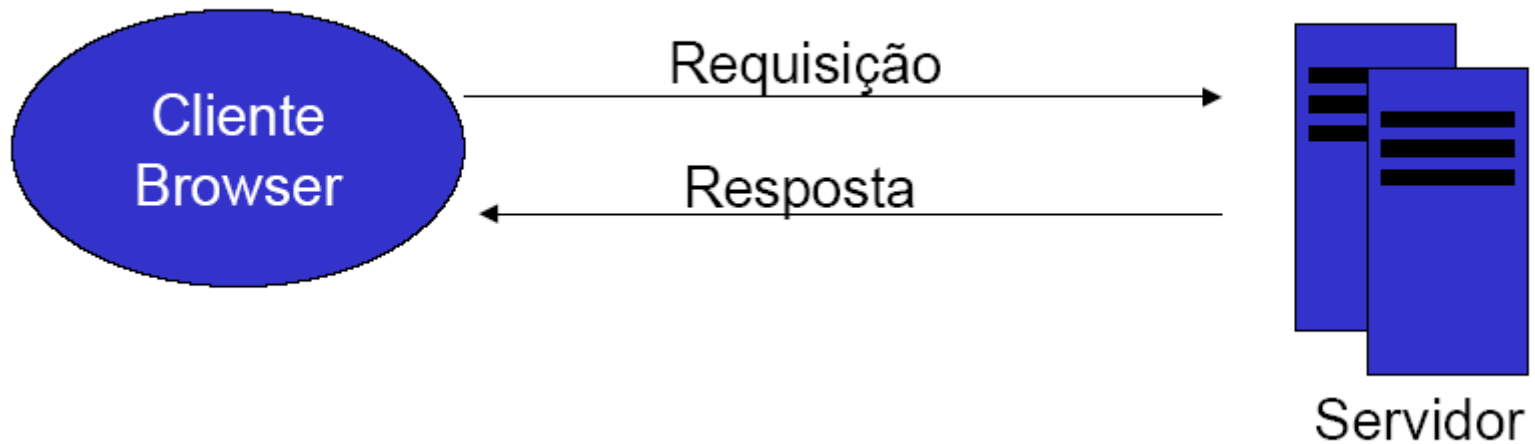
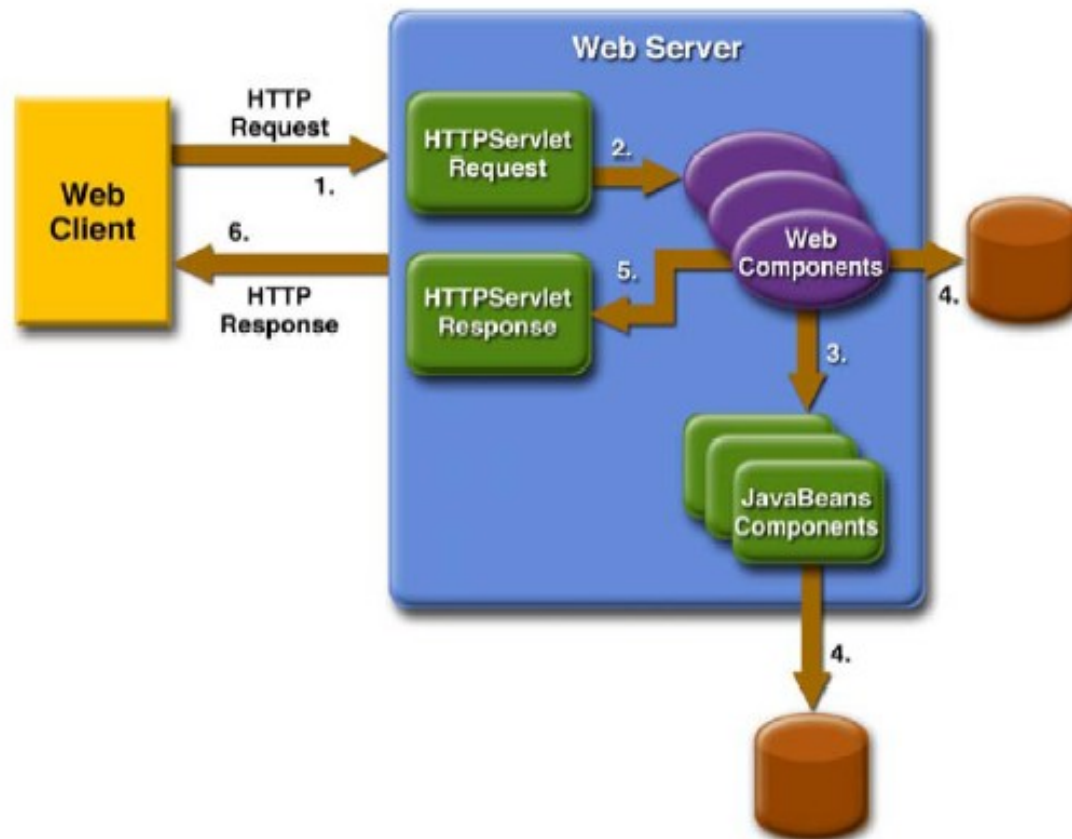


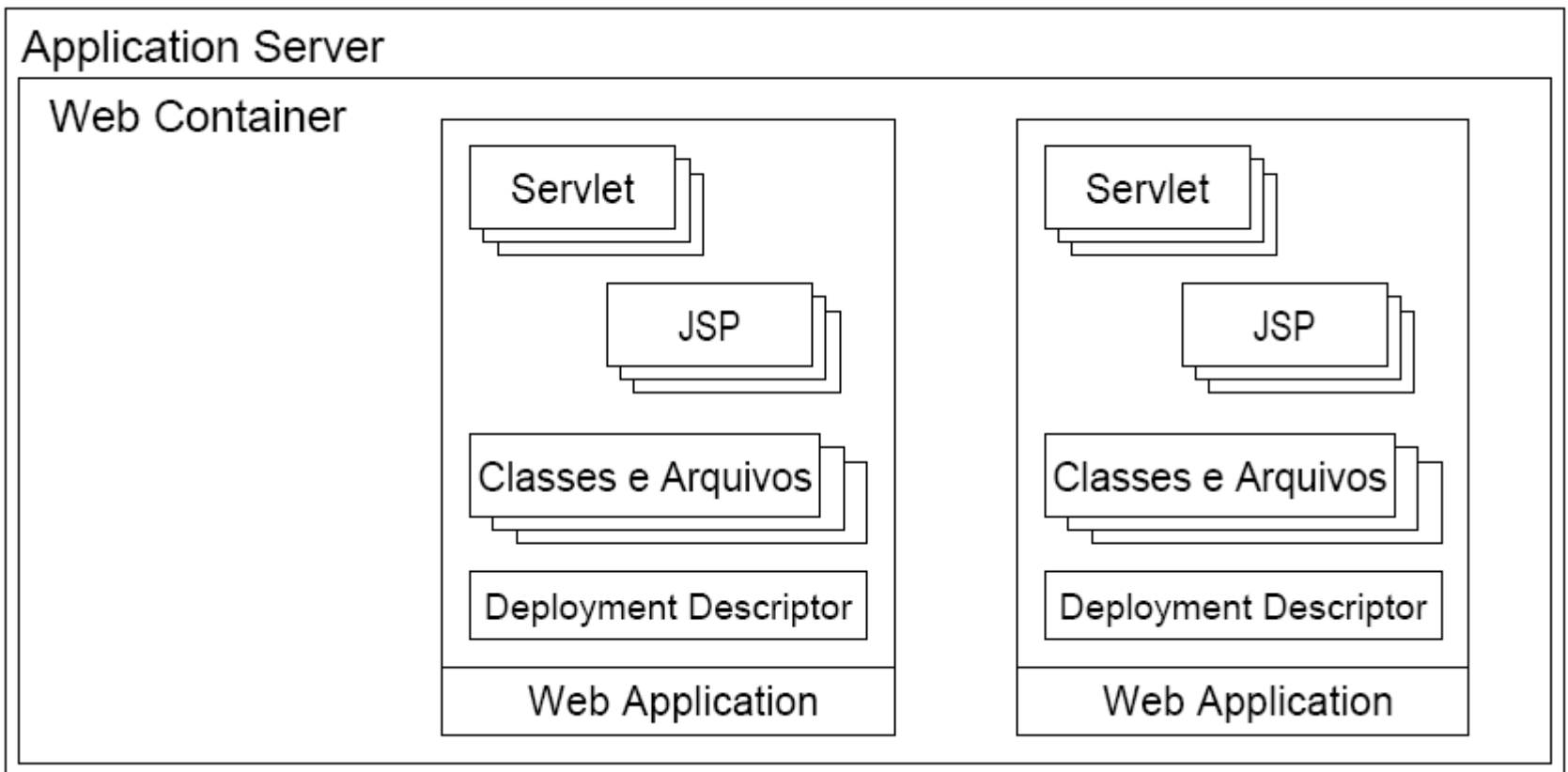
# Arquitetura Web



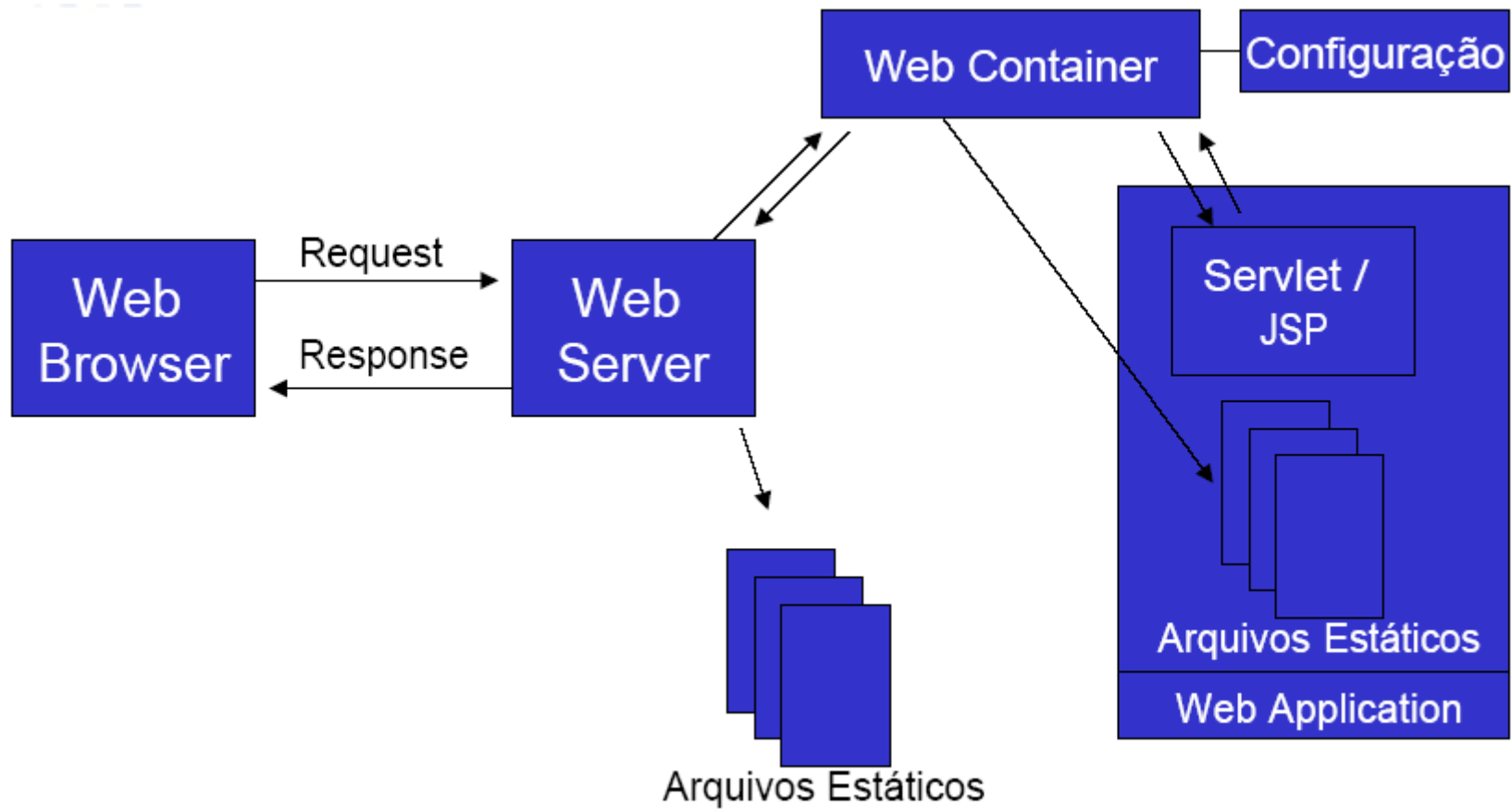
# Container Web



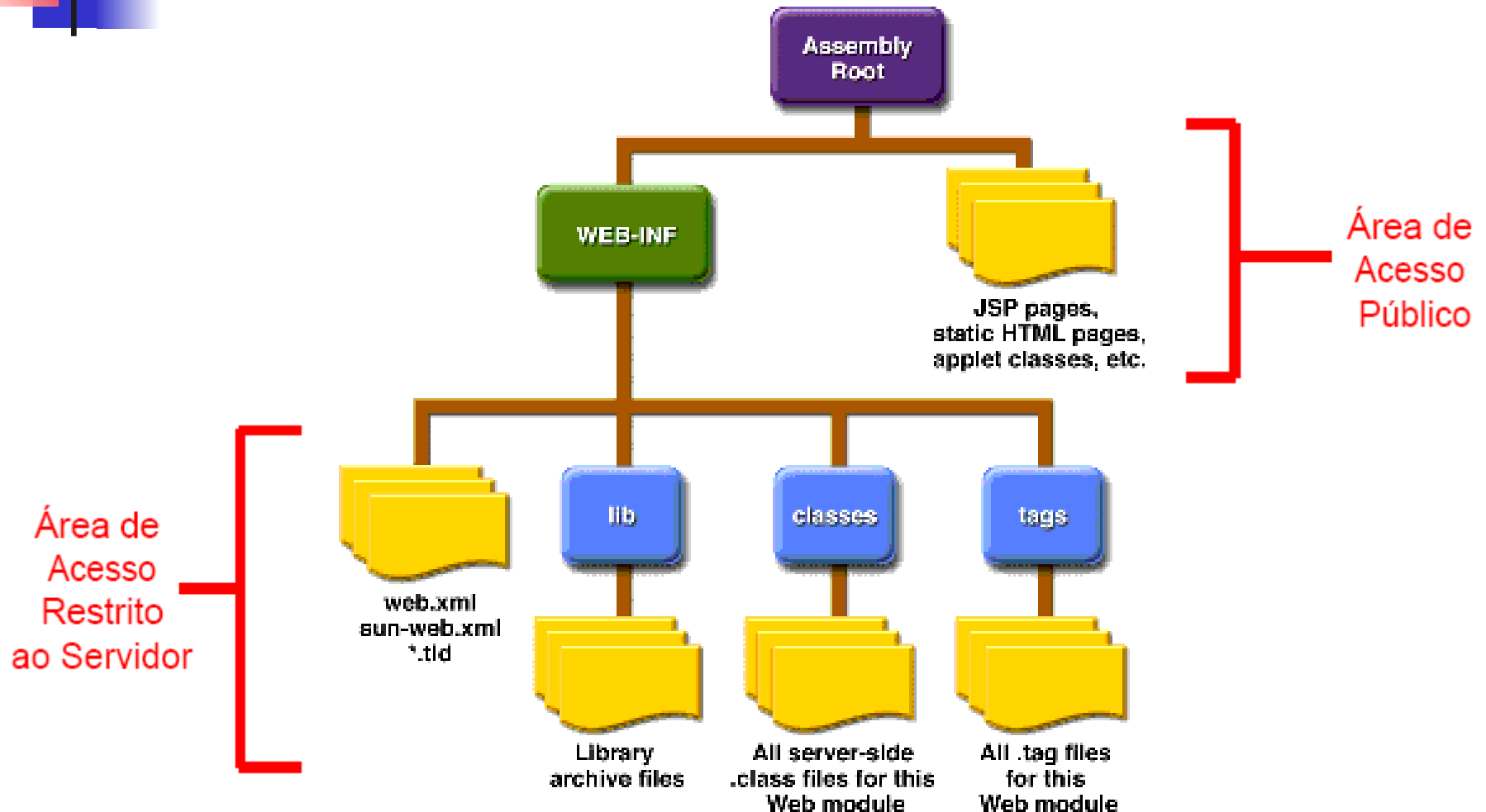
# Web Application



# Web Application, como funciona ?



# Web Application





# Tecnologias do lado do servidor

---

- Estendem as funções básicas do HTTP
  - CGI – Common Gateway Interface
  - API's : Servlet API , Apache API ...
  - Scripts : ASP, JSP, Cold Fusion, PHP



# Tecnologias do lado do servidor

---

- Rodam do lado do servidor, portanto não dependem de suporte dos navegadores
  - Os navegadores fornecem apenas a interface com o usuário
- Interceptam o curso normal da comunicação
  - Recebem dados via requisições HTTP ( GET e POST)
  - Devolvem através de respostas HTTP

- O que são servlets ?
  - Extensão de servidor escrita em Java
  - Podem ser usadas para estender qualquer tipo de aplicação do modelo requisição-resposta
  - Toda Servlet implementa `javax.servlet.Servlet`
- HTTP Servlets
  - Extensões para servidores WEB
  - Estendem `javax.servlet.http.HttpServlet`
  - Lidam com características do HTTP como GET, POST , Cookies , etc.



# Ciclo de vida da Servlet

---

- O Ciclo de vida de uma Servlet é controlado pelo container
- Quando o servidor recebe uma requisição, ela é repassada para o container que delega a uma Servlet
- O container fica responsável por:
  - Carregar a classe em memória
  - Criar uma instância da classe
  - Inicializar a instância chamando o método `init()`

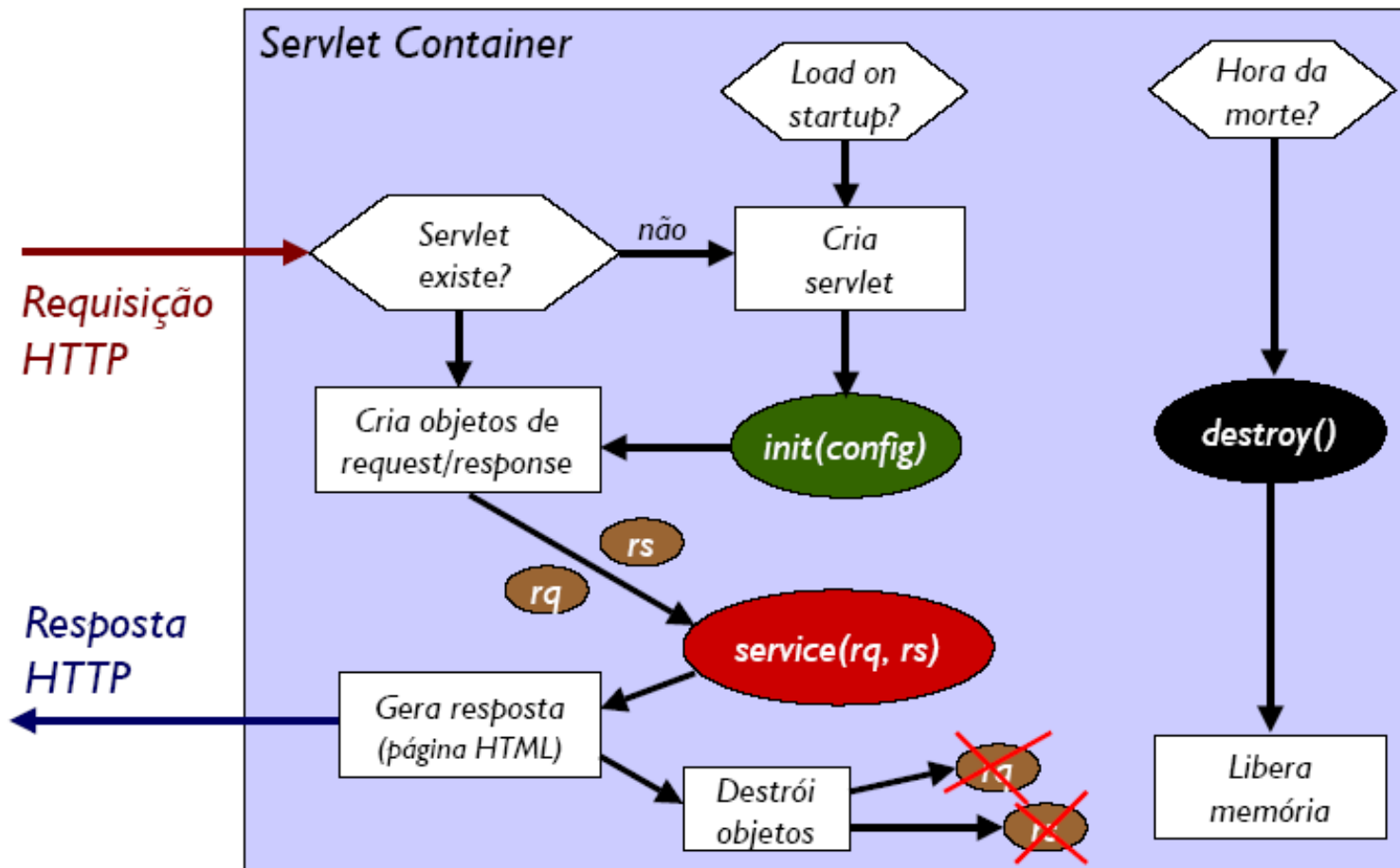


# Ciclo de vida da Servlet

---

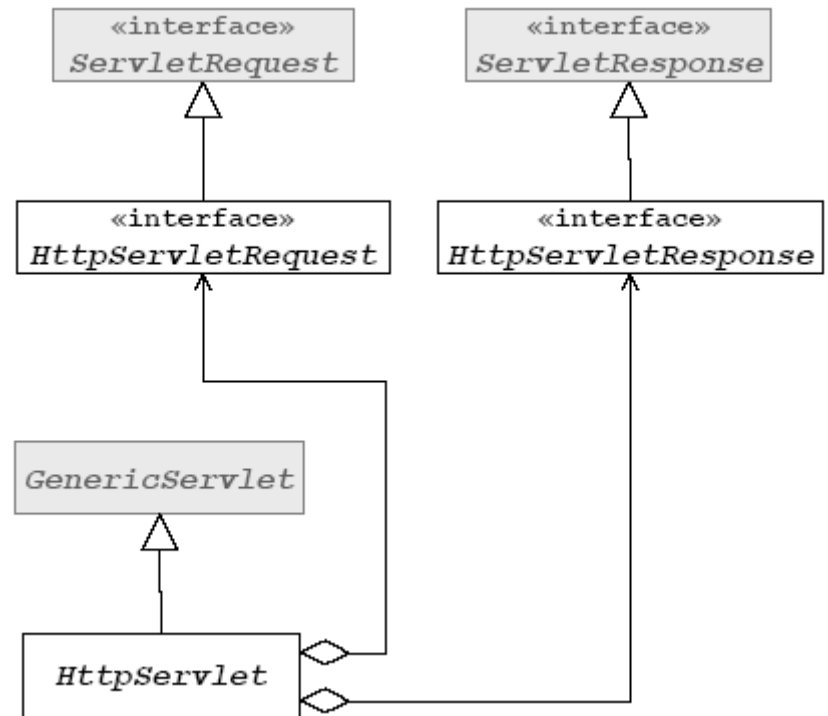
- Depois que a servlet foi inicializada, cada requisição é executada em um método **service()**
  - O Container cria um ServletRequest, que representa um objeto de requisição e um ServletResponse, que representa um objeto de resposta, depois chama o **service()** passando-os como parâmetros
- Quando o container decidir remover a servlet da memória ele a finaliza chamando o método **destroy()**

# Ciclo de vida da servlet



# API: Servlets HTTP

- Interfaces
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
- Classes Abstratas
  - HttpServlet
- Classes Concretas
  - Cookie





# Como escrever um servlet HTTP

---

- Para escrever uma Servlet HTTP, deve-se estender `HttpServlet` e implementar um ou mais métodos de serviço, normalmente o `doPost()` e/ou `doGet()`

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

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

        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<h1>Hello, World!</h1>");
        out.close();
    }
}
```



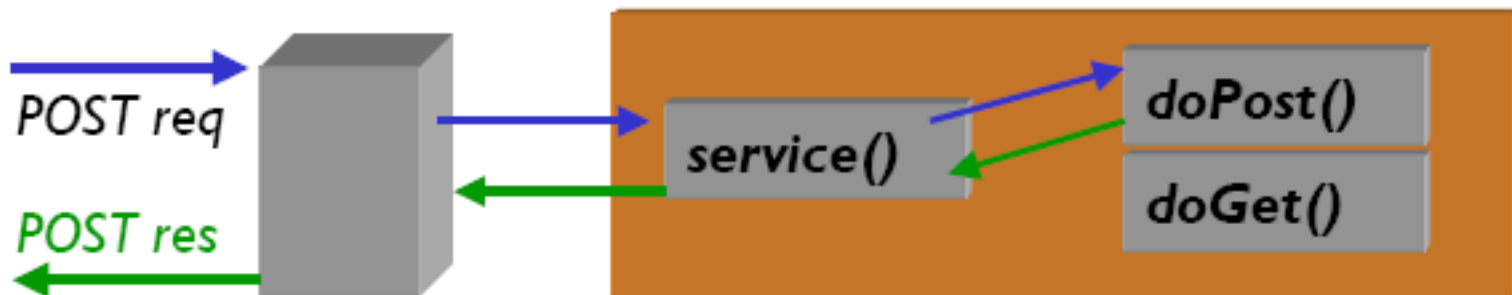
# Métodos de serviço HTTP

---

- A Classe `HttpServlet` redireciona os pedidos encaminhados para `service()` para os métodos que refletem os métodos HTTP (GET , POST ,etc ...)
  - `public void doGet(HttpServletRequest,HttpServletResponse)`
  - `public void doPost(HttpServletRequest,HttpServletResponse)`
  - Outros como `doDelete()`,`doTrace()`,`doPut()`,`doOptions()` ( o método HEAD é implementado pelo `doGet` )

# Métodos de serviço HTTP

- Uma Servlet HTTP deverá estender `HTTPServlet` e implementar pelo menos um dos métodos `doGet()` ou `doPost()`





# Inicialização

---

- A inicialização de uma HttpServlet pode ( e deve ) ser feita através do método **init()**

```
public void init() throws ServletException {  
    String dirImagens =  
        getInitParameter("imagens");  
    if (dirImagens == null) {  
        throw new UnavailableException  
            ("Configuração incorreta!");  
    }  
}
```



# Obtenção de dados de requisições

---

- Alguns métodos de `HttpServletRequest`
  - Enumeration `getHeaderNames()` – obtém nomes dos cabeçalhos
  - `String` `getHeader("nome")` – obtém o valor do cabeçalho
  - `String` `getParameter("nome")` – obtém o valor do parâmetro HTTP
  - `String[]` `getParameterValues("nome")` – obtém os valores do parâmetro HTTP
  - Enumeration `getParameterNames()` – obtém os nomes dos parâmetros



# Obtenção de dados de requisições

---

- Alguns métodos de `HttpServletRequest`
  - `HttpSession getSession()` – retorna a sessão
  - `setAttribute("nome",obj)` – define um atributo `obj` chamado `nome`
  - `Object getAttribute("nome")` – recupera o atributo `nome`



# Preenchimento de uma resposta

---

- Alguns métodos de HttpServletResponse
  - `addHeader(String nome, String valor)` – adiciona cabeçalho HTTP
  - `setContentType(tipo MIME)` – define o tipo MIME para gerar a saída (`text/html` , `image/gif` , etc ...)
  - `sendRedirect(String url)` – redireciona o cliente para a url
  - `Writer getWriter()` – obtém um `Writer` para gerar a saída ( ideal para saídas de texto)
  - `OutputStream getOutputStream()` – obtém um `OutputStream`, ideal para gerar saídas não texto (imagens , etc ...)
  - `reset()` – limpa toda a saída
  - `resetBuffer()` – limpa toda a saída, exceto os cabeçalhos



# Como implementar doGet e doPost

---

- Use **doGet()** para receber requisições GET
  - Links clicados ou URL digitadas no navegador
  - Alguns formulários que usam GET
- Use **doPost()** para receber dados de formulários POST
- Se quiser usar ambos, não sobreponha o **service()**, e sim implemente os 2 métodos



# Parâmetros de requisição

---

- Parâmetros são pares nome/valor que são enviados pelo cliente concatenados em Strings e separados por &
  - nome=**joao**&sobrenome=**paulo**&id=**32**
- Se o método for GET, os parâmetros são passados em uma única linha no query String, que estende a URL após um "?"
  - /servlet/Teste?nome=**joao**&sobrenome=**paulo**&id=**32**
- Se o método for POST, os parâmetros são passados como uma stream no corpo da mensagem



# Como ler parâmetros de requisição

---

- Seja o método POST ou GET, os valores dos parâmetros podem ser recuperados pelo método `getParameter()`
  - `String parametro = request.getParameter("id");`
- Parâmetros de mesmo nome podem ser repetidos, nesse caso o `getParameter()` retornará a primeira ocorrência, para obter todas use :
  - `String[] parans = request.getParameterValues("alunos");`



# Como gerar uma resposta

---

- Para gerar uma resposta, primeiro deve-se obter um fluxo de saída de caracteres (Writer) ou de Bytes ( OutputStream)
  - `Writer out = response.getWriter();`
  - `OutputStream out = response.getOutputStream();`
- Deve-se definir o tipo de dados a ser gerado, isto é importante para que o navegador saiba exibir as informações
  - `response.setContentType("text/html");`
- Deve-se imprimir os dados no objeto de saída
  - `out.println("<h1>Olá</h1>");`



# Criando uma Servlet

---

- São necessárias quatro etapas para construir e usar uma servlet
  - Codificar
  - Compilar
  - Implantar (deploy)
  - Executar



# Compilação e implantação

---

- Para compilar, use qualquer distribuição da API
  - O `servlet.jar` distribuído pelo Tomcat em `common/lib`
  - O `j2ee.jar` distribuído no pacote J2EE da Sun
  - O `javax.servlet.jar` do JBoss em `server/default/lib`
- Inclua o JAR no seu CLASSPATH ao compilar, ex:
  - `> javac -classpath ../servlet.jar;. MeuServlet.java`
- Para implantar, copie as classes compiladas para um contexto existente no servidor
  - Jboss : `/server/default/deploy`

- Se você instalou as servlets em um contexto raiz, execute-as através da URL:
  - <http://localhost:8080/contexto/nome.do.servlet>
- Para passar parâmetros :
  - Escreva um formulário HTML
  - Passe-os via URL
    - <http://localhost:8080/contexto/nome.do.servlet?id=32>



# Exercícios

---

- Crie uma servlet (treinamento.exercicios.Lista) , que imprima em uma tabela todos os nomes de parâmetros enviados e seus valores
  - A servlet deve suportar tanto GET quanto POST