

✓ Estrutura de um Servlet

Um Servlet Java é uma classe que geralmente **estende** `HttpServlet`, que está no pacote `javax.servlet.http`.

📦 Pacotes utilizados:

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

🧱 Métodos mais usados:

- `doGet(HttpServletRequest req, HttpServletResponse res)`
- `doPost(HttpServletRequest req, HttpServletResponse res)`

Esses métodos tratam requisições HTTP GET e POST, respectivamente.

↺ Ciclo de Vida de um Servlet

O ciclo de vida de um Servlet é gerenciado pelo **Servlet Container** (como Tomcat). Os principais métodos envolvidos são:

Método	Função
<code>init()</code>	Chamado uma vez , quando o Servlet é carregado pela primeira vez.
<code>service()</code>	Chamado a cada requisição . Decide se executa <code>doGet()</code> , <code>doPost()</code> , etc.
<code>destroy()</code>	Chamado antes do Servlet ser descarregado da memória.

🔍 Dica importante:

Se você sobrescreve `service()`, os métodos `doGet()` e `doPost()` **não são chamados automaticamente**. Você precisa fazer isso manualmente se quiser mantê-los.

📧 Objetos principais passados ao Servlet:

HttpServletRequest

- Representa a **requisição** do cliente
- Permite acessar parâmetros, headers, IP, etc.

HttpServletResponse

- Representa a **resposta** para o cliente
- Permite escrever HTML, JSON, configurar headers, etc.

3.6 Transferência de Requisições (Forward e Include)

Usada para **manter o contexto da requisição** e encaminhar para outro recurso (Servlet, JSP, etc.), sem reiniciar o ciclo de requisição/resposta.

➤ **forward(request, response)**

- Envia a requisição para outro recurso do mesmo servidor.
- O **cliente não sabe** que foi encaminhado.
- A **URL no navegador não muda**.
- A resposta final **vem do recurso de destino**, e o controle não volta mais para o primeiro.

```
RequestDispatcher dispatcher = request.getRequestDispatcher("pagina.jsp");  
dispatcher.forward(request, response);
```

➤ **include(request, response)**

- Insere a saída de outro recurso **dentro da resposta atual**.
- Útil, por exemplo, para incluir cabeçalhos, rodapés ou menus.

```
RequestDispatcher dispatcher = request.getRequestDispatcher("menu.jsp");  
dispatcher.include(request, response);
```

3.7 Redirecionamento de Servlets (**sendRedirect**)

- Gera uma nova requisição **do lado do cliente** (browser).
- A **URL muda no navegador**.
- **Não mantém os dados da requisição original**.
- O redirecionamento **gera uma nova requisição HTTP GET**.

```
response.sendRedirect("cadastro");
```

Esse método é muito usado, por exemplo, **após um POST**, para evitar reenvios ao atualizar a página (padrão POST/Redirect/GET).

Resumo prático:

Recurso	<code>forward()</code>	<code>include()</code>	<code>sendRedirect()</code>
Tipo de ação	Interna (servidor)	Interna (servidor)	Externa (cliente)
Nova requisição?	✗ Não	✗ Não	✓ Sim
Mantém dados da requisição?	✓ Sim	✓ Sim	✗ Não
URL visível no browser	✗ Não	✗ Não	✓ Sim
Uso comum	Fluxo interno	Montagem de página	Navegação, pós-POST

✓ 3.8 – Anotações em Servlets (Servlet 3.0+)

A especificação **Servlet 3.0 (Java EE 6)** trouxe **anotações** para facilitar a configuração, substituindo (ou complementando) o tradicional `web.xml`.

📌 Principais Anotações

@WebServlet

Define um Servlet diretamente na classe.

```
@WebServlet(name="MeuServlet", urlPatterns={"/minha-url"})
```

Atributos úteis:

Atributo	Descrição
<code>name</code>	Nome do Servlet
<code>urlPatterns</code>	Um ou mais padrões de URL atendidos
<code>initParams</code>	Lista de parâmetros de inicialização (<code>@WebInitParam</code>)

@WebInitParam

Define parâmetros de inicialização que podem ser acessados via `getInitParameter`.

```
@WebInitParam(name="usuario", value="admin")
```


Exemplo completo:

```
@WebServlet(  
    name="ServletLogin",  
    urlPatterns={"/login"},  
    initParams = {  
        @WebInitParam(name = "usuario", value = "admin"),  
        @WebInitParam(name = "senha", value = "admin")  
    }  
)
```

Equivalente ao `web.xml`

As anotações substituem essa configuração:

```
<servlet>
  <servlet-name>ServletLogin</servlet-name>
  <servlet-class>com.Servlet.ServletLogin</servlet-class>
  <init-param>
    <param-name>usuario</param-name>
    <param-value>admin</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>ServletLogin</servlet-name>
  <url-pattern>/login</url-pattern>
</servlet-mapping>
```

 **Atenção:** Se o `web.xml` e as anotações forem usados juntos, o `web.xml` tem **prioridade**.

Outras Anotações Úteis no pacote `javax.servlet.annotation`

Anotação	Função Principal
<code>@WebFilter</code>	Define um filtro , usado para interceptar requisições/respostas
<code>@WebListener</code>	Registra um listener , como eventos de sessão e contexto
<code>@MultipartConfig</code>	Permite receber arquivos com formulários multipart/form-data

Listeners Registráveis com `@WebListener`


Listener Interface	Descrição
<code>ServletContextListener</code>	Inicialização/finalização da aplicação
<code>ServletContextAttributeListener</code>	Mudanças em atributos do contexto
<code>ServletRequestListener</code>	Criação/destruição de requisições
<code>ServletRequestAttributeListener</code>	Atributos da requisição

<code>HttpSessionListener</code>	Criação/fim de sessões
<code>HttpSessionAttributeListener</code>	Atributos da sessão


3.9 – Mapeamento com Curinga

É possível usar curingas (*) no mapeamento de Servlets:


```
@WebServlet(name = "ServletLogin", urlPatterns = {"*.do"})
```

 Isso permite acessar o servlet com qualquer URL que termine com `.do`:

- `/login.do`
- `/cadastro.do`
- `/programa.do`

 Muito usado em frameworks como **Struts** e **JSF** para criar **controladores centralizados**.

3.10 – ServletConfig


 Usado para obter **parâmetros de inicialização específicos** do Servlet (aqueles definidos via `@WebInitParam` ou dentro do `<servlet>` no `web.xml`).

Exemplo:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String usuario = getServletConfig().getInitParameter("usuario");
    String senha = getServletConfig().getInitParameter("senha");
    PrintWriter out = response.getWriter();
    out.println("Usuário: " + usuario);
    out.println("Senha: " + senha);
}
```

Métodos de ServletConfig:

- `getInitParameter(String name)` – retorna o valor de um parâmetro específico.
- `getInitParameterNames()` – retorna todos os nomes dos parâmetros.

 Dica: você pode carregar os parâmetros no `init()` para usar durante todo o ciclo de vida do Servlet.

✓ 3.11 – ServletContext

 Representa **toda a aplicação** (contexto da web). É útil para:

- Compartilhar dados entre Servlets
- Obter configurações globais

Exemplo de parâmetro de contexto (**web.xml**):

```
<context-param>
  <param-name>aplicacao</param-name>
  <param-value>Contas a pagar</param-value>
</context-param>
```

Acesso no código:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String app = getServletContext().getInitParameter("aplicacao");
    PrintWriter out = response.getWriter();
    out.println("Aplicação: " + app);
}
```


Métodos de **ServletContext**:

Método	Função
<code>getInitParameter(String)</code>	Pega valor de parâmetro global (do <code>context-param</code>)
<code>setAttribute(String, Object)</code>	Armazena objeto visível a todos os Servlets
<code>getAttribute(String)</code>	Recupera objeto compartilhado
<code>removeAttribute(String)</code>	Remove objeto compartilhado

Diferença entre **ServletConfig** e **ServletContext**

Característica	ServletConfig	ServletContext
Escopo	Por Servlet	Por aplicação (global)
Onde configurar	Dentro do Servlet (<code>@WebInitParam</code> ou <code><servlet></code>)	Em <code><context-param></code> no <code>web.xml</code>
Como obter	<code>getServletConfig()</code>	<code>getServletContext()</code>
Compartilhamento	Não compartilha entre Servlets	Compartilhado entre todos os Servlets

4.1 – Definição de Gerenciamento de Sessão

 HTTP é *stateless*

- Cada requisição HTTP é **independente** — ela **não carrega informações** sobre as requisições anteriores.
 - Isso significa que, por padrão, **o servidor não “lembra”** do usuário entre duas requisições diferentes.
-

? Qual o problema disso?

Imagine um site de compras onde você:

1. Adiciona um item no carrinho,
2. Vai para outra página,
3. O carrinho desaparece...

Sem um mecanismo para manter o estado da interação do usuário, **isso aconteceria o tempo todo.**

💡 Solução: Sessão HTTP

O container Java (ex: Tomcat) oferece uma solução: **gerenciamento de sessão.**

🔑 **Sessão = espaço temporário de memória no servidor** associado a um usuário.

- Criada automaticamente na **primeira requisição** de um usuário.
 - O container gera um **ID único de sessão** (geralmente via cookie **JSESSIONID**).
 - Cada vez que o usuário faz nova requisição, esse ID é enviado ao servidor.
 - O servidor usa esse ID para **associar a requisição à sessão existente.**
-

📌 Características da sessão

- **Armazena objetos (inclusive sensíveis)** na memória do servidor.

- Permite manter valores durante **toda a visita do usuário** ao site.
 - O tempo de vida da sessão:
 - É **renovado a cada requisição**.
 - Expira após um período de **inatividade**, liberando recursos.
-

Exemplo de uso típico

Durante o login, você pode armazenar informações do usuário assim:

```
HttpSession session = request.getSession();  
session.setAttribute("usuarioLogado", usuario);
```

E depois, em outras páginas:

```
Usuario usuario = (Usuario) session.getAttribute("usuarioLogado");
```

Alternativas para preservar estado sem sessão

- Campos ocultos (`<input type="hidden">`) em formulários.
 - URLs com parâmetros (`?id=123`).
 - Cookies persistentes, mas estes ficam do lado do cliente.
-

Conclusão

O **gerenciamento de sessão** é essencial para criar aplicações web que oferecem:

- Navegação contínua,
- Segurança (sem trafegar dados sensíveis em URLs),
- Experiência personalizada para cada usuário.

4.2 – Modelo do Controle de Sessão

Geração da Sessão no Servidor

- O **web container** (como Tomcat) cria uma **sessão (HttpSession)** para cada cliente.

- Um **ID único da sessão** é gerado no servidor.
 - Esse ID é transferido para o **cliente**, geralmente por meio de:
 - **Cookies (padrão)**,
 - **Campo oculto de formulário** (`<input type="hidden">`),
 - **Reescrita de URL** (`/pagina.jsp;jsessionid=XYZ`).
-

Funcionamento do mecanismo

1. Cliente faz a **primeira requisição**.
 2. Servidor cria um **HttpSession** e gera um **ID**.
 3. Esse ID é **enviado para o cliente**.
 4. Cliente faz nova requisição → **ID é enviado de volta**.
 5. Servidor reconhece o usuário e **acessa a sessão existente**.
-

4.3 – A classe HttpSession

A sessão é representada pela interface `javax.servlet.http.HttpSession`.

Características:

- Criada **automaticamente** na primeira requisição.
 - Associada a **um único usuário ativo**.
 - Expira após um tempo de inatividade (timeout configurável).
 - Permite armazenar **atributos da sessão** com escopo de usuário.
-

Principais métodos do HttpSession

Método	Descrição
<code>setAttribute(String, Object)</code>	Armazena um valor na sessão
<code>getAttribute(String)</code>	Recupera um valor da sessão
<code>removeAttribute(String)</code>	Remove um atributo da sessão
<code>invalidate()</code>	Invalida a sessão atual
<code>getId()</code>	Retorna o ID da sessão
<code>getCreationTime()</code>	Retorna o horário de criação
<code>getLastAccessedTime()</code>	Retorna o horário do último acesso
<code>setMaxInactiveInterval(int)</code>	Define o tempo de inatividade (em segundos)

Escopos (Figura 35)

- **ServletContext**: escopo da **aplicação inteira**.
 - **HttpSession**: escopo de **um usuário específico**.
 - **Request**: escopo de **uma única requisição**.
-

4.4 – Campos Ocultos

Campos ocultos são incluídos em formulários HTML para **manter informações entre requisições**.

Exemplo:

```
<input type="hidden" name="cmpHidden" value="true">
```

Vantagens:

- Simples de usar em formulários.

- Funciona mesmo sem cookies.

Desvantagens:

- Só funciona enquanto o fluxo de formulários está intacto.
- Não mantém dados em navegação fora de formulários.
- Dados visíveis ao cliente (apesar de ocultos no HTML).

Reescrita de URL (fallback para cookies desabilitados)

Se cookies estiverem desativados, o servidor pode **anexar o ID da sessão na URL**, assim:

`http://meusite.com/produtos.jsp;jsessionid=ABC123XYZ`

Esse comportamento é automático, mas você pode forçar via:

```
response.encodeURL("produtos.jsp");
```