

**INSTITUTO
FEDERAL**

Paraíba

Campus
Cajazeiras

PROGRAMAÇÃO P/ WEB 1

3.2 Servlets- Gerenciamento de Aplicações (BASEADO NO MATERIAL DO PROF. FABIO GOMES)

PROF. DIEGO PESSOA

✉ DIEGO.PESSOA@IFPB.EDU.BR

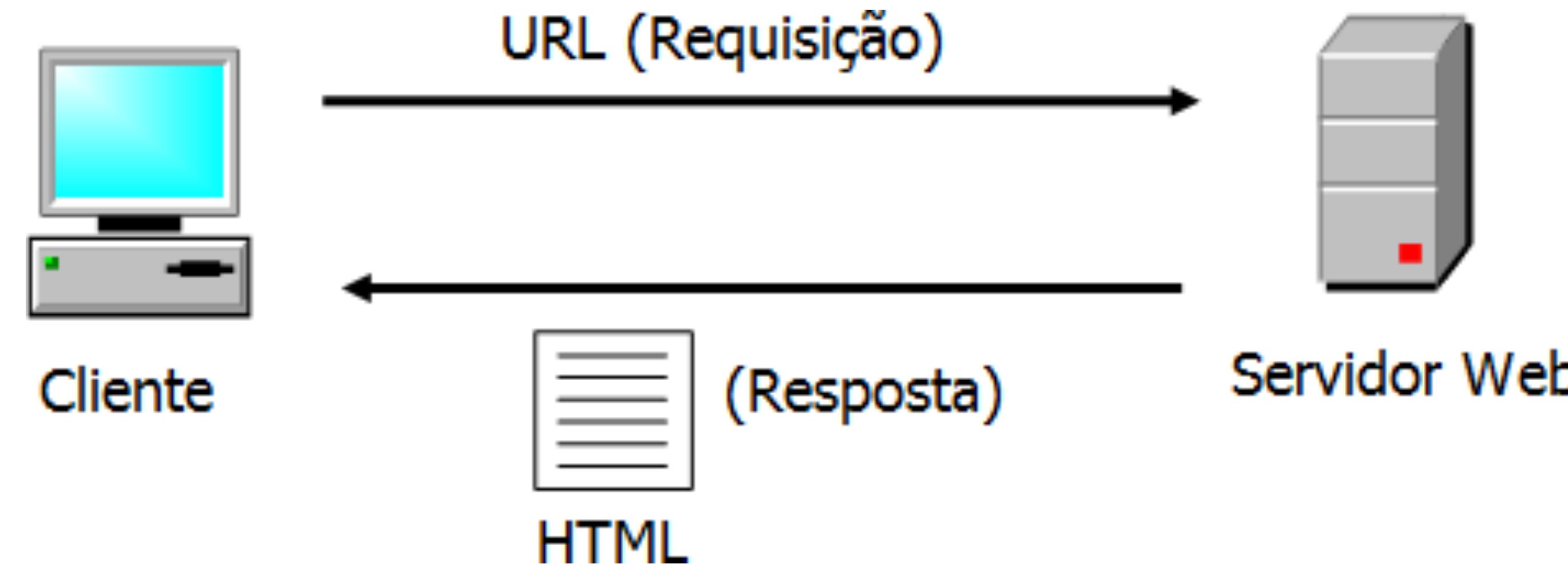
🐱 [@DIEGOEP](https://github.com/DIEGOEP)



CST em Análise e
Desenvolvimento
de Sistemas

Entendendo o HTTP

- Visão geral de uma comunicação HTTP:

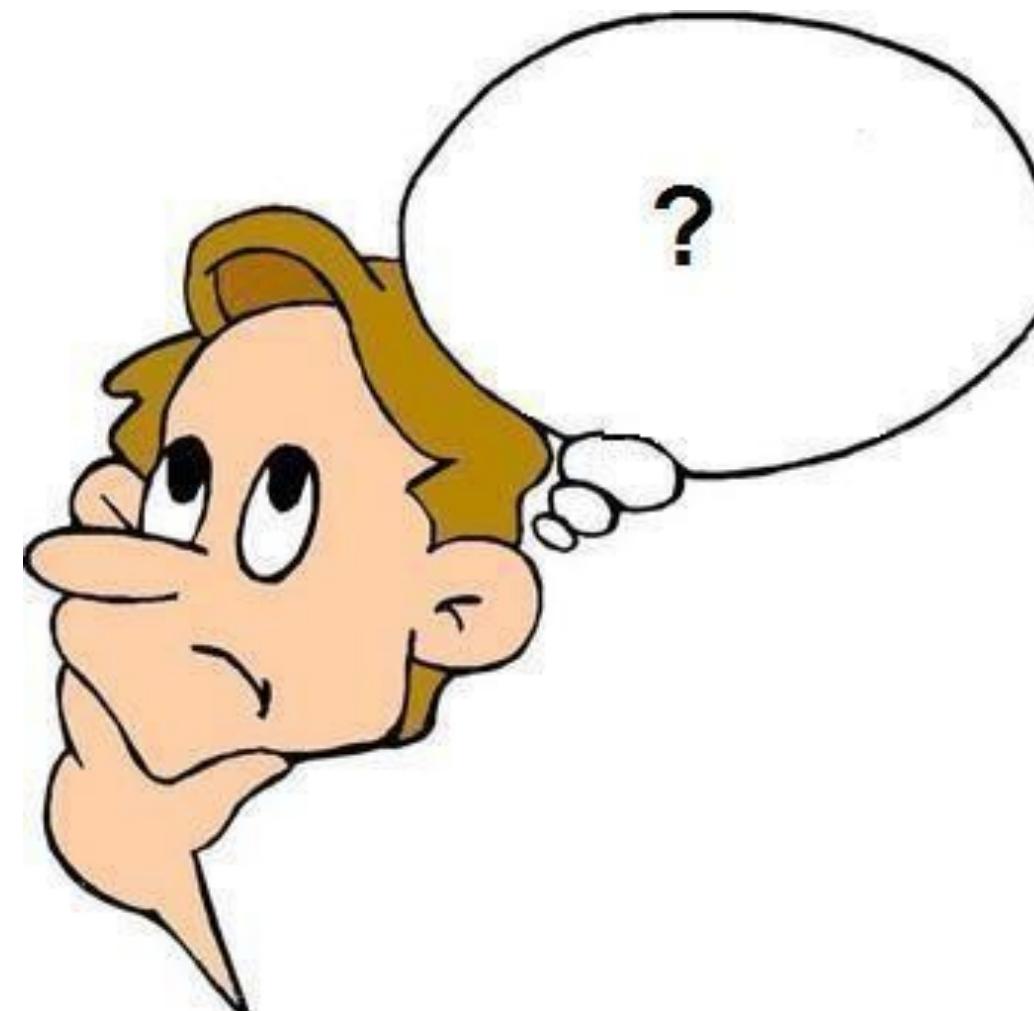


Motivação

- Uma aplicação Web é normalmente composta por vários componentes que trabalham em conjunto;
- Muitas vezes, estes componentes precisam interagir entre si;

Motivação

- Isto implica na necessidade de preparamos o ambiente da aplicação de forma que estas interações possam acontecer de forma conveniente;



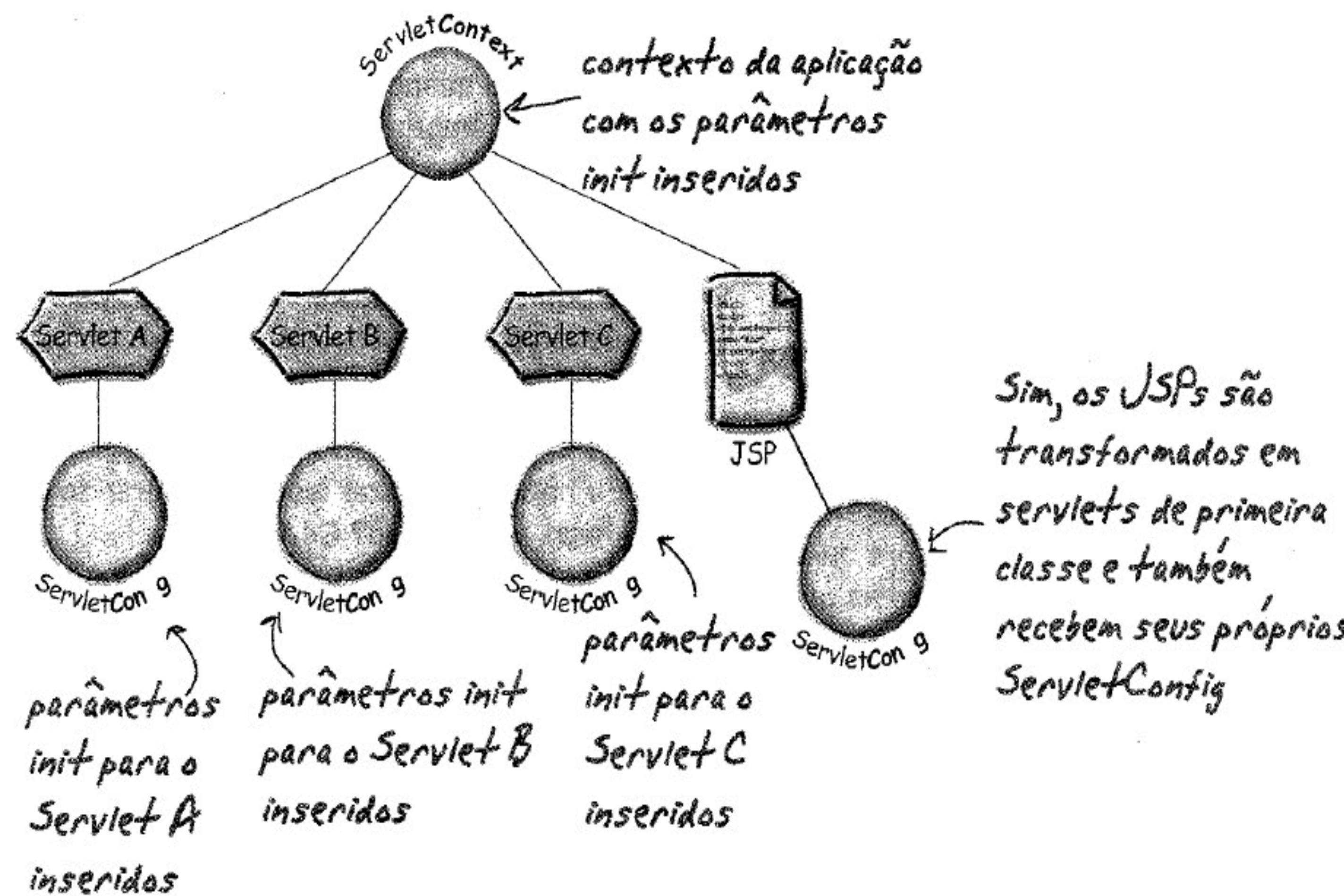
O Contexto da Aplicação

- Um servlet pode estar presente em mais de uma aplicação;
 - Mas cada uma de suas instâncias está associada a apenas uma;
- A aplicação na qual o servlet está sendo executado é chamada de contexto;

O Contexto da Aplicação

- Todo servlet tem acesso a um objeto que contém informações sobre a aplicação na qual ele está sendo executado;
- Este objeto implementa a interface **ServletContext**;

O Contexto da Aplicação



O Container lê o DD e cria um par de String nome/valor para cada <context-param>.

O Container cria uma nova instância do ServletContext.

O Container dá ao ServletContext uma referência a cada par nome/valor dos parâmetros init do contexto.

Cada servlet e JSP distribuído como parte de uma única aplicação tem acesso àquele mesmo ServletContext.

A Interface ServletContext

- Os objetos desta interface representam a aplicação na qual o servlet está sendo executado;
- A implementação destes objetos é oferecida pelo container no qual a aplicação é executada;

A Interface ServletContext

- Através do ServletContext, um servlet pode:
 - Acessar os valores dos parâmetros de inicialização da aplicação;
 - Realizar o log de eventos;
 - Compartilhar informações entre os componentes da aplicação;



A Interface ServletContext

- Através do ServletContext, um servlet pode:
 - Obter referências para outros componentes da aplicação;
 - Interagir com outros componentes da aplicação;



A Interface ServletContext

Obtém os parâmetros init e obtém/especifica atributos

Obtém informações sobre o servidor/container.

Falaremos adiante neste capítulo sobre o RequestDispatcher.

<code><<interface>></code>
<code>ServletContext</code>
<code>getInitParameter(String)</code>
<code>getInitParameterNames()</code>
<code>getAttribute(String)</code>
<code>getAttributeNames()</code>
<code>setAttribute(String)</code>
<code>removeAttribute(String)</code>
<hr/>
<code>getMajorVersion()</code>
<code>getServerInfo()</code>
<hr/>
<code>getRealPath(String)</code>
<code>getResourceAsStream(String)</code>
<code>getRequestDispatcher(String)</code>
<hr/>
<code>log(String)</code>
<code>// mais métodos</code>

Obtém os parâmetros init e obtém/especifica atributos

← Escreve no arquivo de log do servidor (específico por fabricante) ou no System.out.

Parâmetros de Inicialização

- Em uma aplicação web, podemos pré-definir um ou mais parâmetros de inicialização;
- Parâmetros podem ser aplicados para toda a aplicação ou apenas para um servlet específico;
- Estes parâmetros são inicializados no momento em que a aplicação é iniciada;

Parâmetros de Inicialização

- Os parâmetros de inicialização podem ser definidos no descriptor da aplicação;
 - Independente do escopo;
- Parâmetros de inicialização para servlets podem ser definidos através de anotações;

Parâmetros de Inicialização

- Configurando parâmetros para um servlet específico:
 - Os parâmetros de inicialização para um servlet são definidos dentro do elemento que o descreve;
 - Cada parâmetro é definido através de um elemento do tipo **init-param**;

Parâmetros de Inicialização

- Configurando parâmetros para um servlet específico:
 - O elemento `init-param` é composto por dois sub-elementos;
 - ✓ `param-name`: o nome do parâmetro;
 - ✓ `param-value`: o valor do parâmetro;

Parâmetros de Inicialização

- Configurando parâmetros para um servlet específico:

```
<servlet>
    <servlet-name>exemplo</servlet-name>
    <servlet-class>ExemploDeInicializacao</servlet-class>
    <init-param>
        <param-name>email</param-name>
        <param-value>fabio@ifpb.edu.br</param-value>
    </init-param>
    <init-param>
        <param-name>disciplina</param-name>
        <param-value>psd</param-value>
    </init-param>
</servlet>
```

Parâmetros de Inicialização

- Parâmetros de inicialização para servlets também podem ser definidos com anotações;
- Cada parâmetro é definido através da anotação `@WebInitParam`;
 - Este elemento é definido dentro da anotação `@WebServlet`, dentro de um atributo chamado `initParams` ;

Parâmetros de Inicialização

- Definindo parâmetros através de anotações:

```
@WebServlet(  
    urlPatterns = {"/parametrosServlet"},  
    initParams = {  
        @WebInitParam (name = "email", value = "fabio@ifpb.edu.br"),  
        @WebInitParam (name = "disciplina", value = "psd")  
    }  
)
```

Parâmetros de Inicialização

- Obtendo os parâmetros de um servlet:
 - As informações de configuração do servlet são encapsuladas em objetos que implementam a interface **ServletConfig**;
 - Esta interface é implementada pela classe **GenericServlet**;
 - O servlet pode obter uma referência para este objeto através do método **getServletConfig**;

Parâmetros de Inicialização

- Métodos da interface ServletConfig:

javax.servlet.ServletConfig

<<interface>>

ServletConfig

getInitParameter(String)

Enumeration getInitParameterNames()

getServletContext()

getServletName()



*A maioria das pessoas
nunca usa este método*

Parâmetros de Inicialização

- Métodos da interface `ServletConfig`:
 - `getInitParameterNames()`:
 - ✓ Retorna uma enumeração contendo o nome de todos os parâmetros de inicialização do servlet;
 - `getInitParameter(String parameterName)`:
 - ✓ Retorna o valor de um determinado parâmetro;
 - ✓ Ou null, caso o parâmetro não exista;

Parâmetros de Inicialização

- Métodos da interface `ServletConfig`:
 - `getServletName()`:
 - ✓ Retorna o nome que identifica o servlet no descriptor da aplicação ou o nome da classe (caso o servlet seja descrito através de uma anotação);
 - `getServletContext()`:
 - ✓ Retorna uma referência para o objeto que contém as informações de configuração da aplicação;

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException{
    PrintWriter out = response.getWriter();
    out.println("<html><head><title> Informações de Configuração do Servlet " +
        "</title></head>");
    out.println("<body>");
    out.println("<p> " +
        "<h1> Informações de Configuração do Servlet<h1> " +
        "</p>");

    out.println("<p><h2><b> Nome do servlet:</b> "+getServletName()+"</h2></p>");

    out.println("<p><h2>Parâmetros de inicialização</h2></p>");

    Enumeration parameterNames = getInitParameterNames();

    while(parameterNames.hasMoreElements()) {
        String parameterName = parameterNames.nextElement().toString();
        String value = getInitParameter(parameterName);
        out.println("<p><h2> <b>" + parameterName+":"+value+"</b></h2></p>");

    }

    out.println("</body></html>");

}
```

Parâmetros de Inicialização

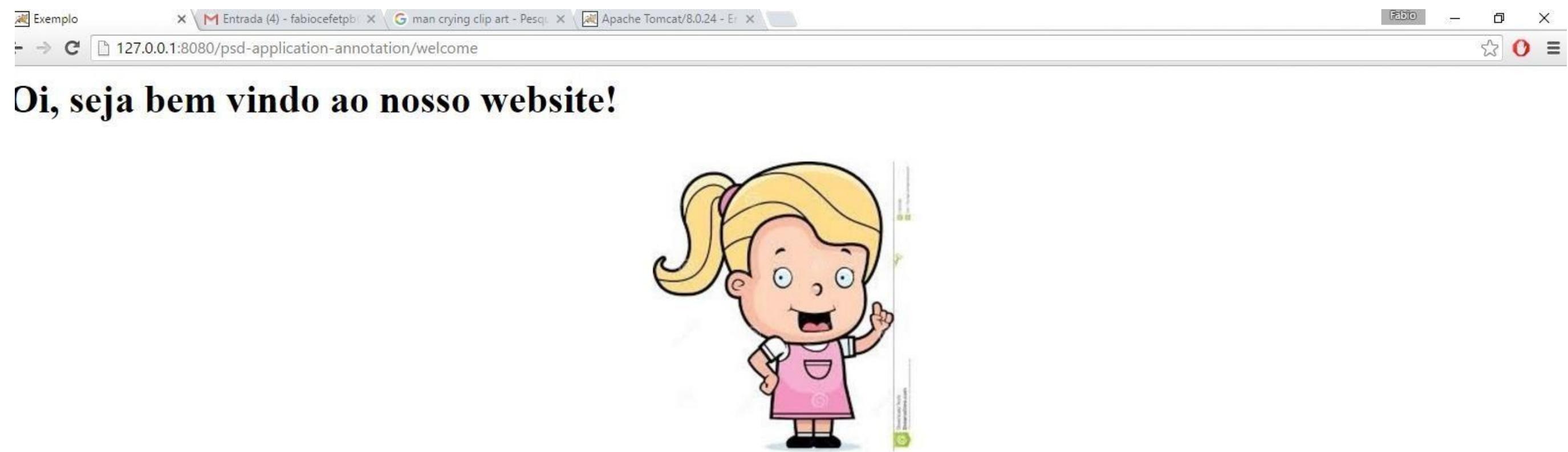
- Vamos agora ver um servlet que dá boas vindas ao usuário;
- O servlet tem uma imagem e uma frase, que são configuradas através de parâmetros de inicialização;

Parâmetros de Inicialização

```
public class MostraParametrosWelcome extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws IOException{  
        doPost(request, response);  
    }  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws IOException{  
        PrintWriter out = response.getWriter();  
        out.println("<html><head><title> Exemplo </title></head>");  
        out.println("<body>");  
        out.println("<p align=\"center\"> " +  
                    "<h1> Oi, seja bem vindo ao nosso website! <h1> </p>");  
        String image = getServletConfig().getInitParameter("image");  
        String tip = getServletConfig().getInitParameter("tip");  
  
        out.println("<p align=\"center\"> <image src=\"images/" +image+ "\">");  
        out.println("<p align=\"center\"> Dica do dia:" +tip+ "</p>");  
  
        out.println("</body></html>");  
    }  
}
```

Parâmetros de Inicialização

- Resultado:



Dica do dia: A vingança nunca é plena. Mata a alma e envenena.

Parâmetros de Inicialização

- Configurando parâmetros para a aplicação:
 - Parâmetros para a aplicação são definidos na raiz do descritor;
 - Cada parâmetro é definido através de um elemento do tipo **context-param**;
 - Este elemento é composto pelos mesmos sub-elementos que compõem o **init-param**;

Parâmetros de Inicialização

- Configurando parâmetros para a aplicação:

```
<context-param>
    <param-name>idioma</param-name>
    <param-value>português</param-value>
</context-param>
<context-param>
    <param-name>sistema</param-name>
    <param-value>windows</param-value>
</context-param>
<context-param>
    <param-name>versao</param-name>
    <param-value>1.0</param-value>
</context-param>
```

Parâmetros de Inicialização

- Obtendo os parâmetros da aplicação:
 - As informações de configuração da aplicação são encapsuladas no objeto que implementa o **ServletContext**;
 - Este objeto pode ser obtido através do método **getServletContext** da interface **ServletConfig**;

Parâmetros de Inicialização

- Métodos da interface ServletContext para a manipulação de parâmetros:

<pre><<interface>> ServletContext</pre>	
Obtém os parâmetros init e obtém/especifica atributos	<pre>getInitParameter(String) getInitParameterNames()</pre>
Obtém informações sobre o servidor/container.	<pre>getAttribute(String) getAttributeNames() setAttribute(String) removeAttribute(String)</pre> <hr/> <pre>getMajorVersion() getServerInfo()</pre> <hr/> <pre>getRealPath(String) getResourceAsStream(String) getRequestDispatcher(String)</pre> <hr/> <pre>log(String) // mais métodos</pre>
Faremos adiante neste capítulo sobre o RequestDispatcher.	<p>← Escreve no arquivo de log do servidor (específico por fabricante) ou no System.out.</p>

Parâmetros de Inicialização

- Métodos da interface ServletContext para a manipulação de parâmetros:
 - `getInitParameterNames()`:
 - ✓ Retorna uma enumeração contendo o nome de todos os parâmetros de inicialização da aplicação;
 - `getInitParameter(String parameterName)`:
 - ✓ Retorna o valor de um determinado parâmetro;
 - ✓ Ou null, caso o parâmetro não exista;

Imprimindo os parâmetros de inicialização da aplicação

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException{
    PrintWriter out = response.getWriter();
    out.println("<html><head><title> Parâmetros de Inicialização da Aplicação " +
               "</title></head>");
    out.println("<body>");
    out.println("<p> " +
               "<h1> Parâmetros de Inicialização da Aplicação<h1> " +
               "</p>");

    out.println("<p><h2><b> Contexto da aplicação:</b> "+getServletContext().getContextPath()
               +"</h2></p>");
    out.println("<p><h2><b> Server Info:</b> "+getServletContext().getServerInfo()+
               "</h2></p>");

    Enumeration parameterNames = getServletContext().getInitParameterNames();
    while(parameterNames.hasMoreElements()) {
        String parameterName = parameterNames.nextElement().toString();
        String value = getServletContext().getInitParameter(parameterName);
        out.println("<p><h2> <b>"+ parameterName+":</b>"+value+"</h2></p>");
    }
    out.println("</body></html>");
}
```

Compartilhando Informações

- Parâmetros de inicialização são muito úteis, mas possuem algumas limitações:
 - Só podem receber valores do tipo String;
 - Os seus valores precisam ser conhecidos no momento em que a aplicação é inicializada;
 - Os seus valores não podem ser alterados em tempo de execução;

Compartilhando Informações

- Existem situações nas quais precisamos inicializar um objeto que será usado pelos componentes da aplicação;
 - Que podem eventualmente alterar o estado destes objetos;
- Nestes casos, parâmetros de inicialização não resolvem o problema;

Compartilhando Informações

- Tampouco, não podemos delegar esta tarefa a um servlet;
 - Como garantir que este *servlet* seria executado antes de todos os outros?

Compartilhando Informações

Precisamos de um objeto à parte que possa:

- Ser notificado quando o contexto é inicializado (a aplicação está sendo distribuída).
 - Conseguir os parâmetros init do contexto através do ServletContext.
 - Usar o nome de lookup do parâmetro init para fazer uma conexão com o banco de dados.
 - Armazenar a conexão com o banco de dados como um atributo, para que todas as partes da aplicação possam acessá-la.
- Ser notificado quando o contexto é destruído (a aplicação é retirada do ar ou cai).
 - Encerrar a conexão com o banco de dados.

Compartilhando Informações

- Neste caso, precisamos de uma solução que nos permita:
 - Inicializar um objeto para a aplicação;
 - Disponibilizar este objeto para os componentes da aplicação;
 - Destruir o objeto quando a aplicação for finalizada;

Inicializando Objetos

- O **ServletContextListener**:
 - Os objetos que implementam esta interface são notificados sobre eventos ocorridos na aplicação;
 - Usamos este tipo de objeto para programar ações que devem ser executadas no momento de inicialização ou finalização da aplicação;

Inicializando Objetos

- Para ser notificado sobre os eventos, o listener precisa ser adicionado à aplicação;
 - Uma aplicação pode ter vários listeners;
- O container é responsável por instanciar cada listener;
 - E por notificá-los quando algum dos eventos esperados por eles acontece;

Inicializando Objetos

- Listeners podem ser cadastrados na aplicação de duas formas:
 - Através do descritor da aplicação;
 - Através de uma anotação;

Inicializando Objetos

- Os listeners podem ser cadastrados através do descritor da aplicação;
- Para isso, usamos um elemento chamado **listener**;
- Cada **listener** tem um sub-elemento **listener-class**, no qual definimos a classe que o implementa;

Inicializando Objetos

- Cadastrando um listener no descriptor da aplicação:

```
<listener>
    <listener-class>servlets.ApplicationLogger</listener-class>
</listener>
```

Inicializando Objetos

- Listeners também podem ser adicionados à aplicação através de uma anotação;
- Para isso, usamos a anotação `@WebListener`;

Inicializando Objetos

- Uma classe anotada com a anotação `@WebListener` precisa implementar uma das seguintes interfaces:
 - `javax.servlet.ServletContextListener`
 - `javax.servlet.ServletContextAttributeListener`
 - `javax.servlet.ServletRequestListener`

Inicializando Objetos

- Uma classe anotada com a anotação `@WebListener` precisa implementar uma das seguintes interfaces:
 - `javax.servlet.ServletRequestAttributeListener`
 - `javax.servlet.http.HttpSessionListener`
 - `javax.servlet.http.HttpSessionAttributeListener`

Inicializando Objetos

- Métodos do ServletContextListener:
 - `contextInitialized(ServletContextEvent event);`
 - ✓ Neste método, definimos o que deve ser feito quando a aplicação for inicializada;
 - `contextDestroyed(ServletContextEvent event);`
 - ✓ Neste método, definimos o que deve ser feito quando a aplicação for encerrada;

Inicializando Objetos

- O **ServletContextListener**:
 - Ambos os métodos da interface recebem um objeto da classe **ServletContextEvent**;
 - Neste objeto, podemos usar o método **getServletContext()** para obter uma referência para o contexto da aplicação;
 - ✓ E acessar as informações sobre a mesma;

Inicializando Objetos

- O ServletContextListener:

Uma classe ServletContextListener:

```
import javax.servlet.*;
public class MyServletContextListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent event) {
        //código para inicializar a conexão com o banco de dados
        //e armazená-la como um atributo do contexto
    }
    public void contextDestroyed(ServletContextEvent event) {
        //código para encerrar a conexão com o banco de dados
    }
}
```

O ServletContextListener
está no pacote javax.servlet

O listener do contexto
é simples: implementa o
ServletContextListener

Estas são as duas notificações
que você recebe. Ambas exibem
um ServletContextEvent

```
package com.example;
```

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

```
public class MyServletContextListener implements ServletContextListener {
```

```
    public void contextInitialized(ServletContextEvent event) {
```

```
        ServletContext sc = event.getServletContext(); ← Solicita o ServletContext  
        ao evento.
```

```
        String dogBreed = sc.getInitParameter("breed");
```

```
        Dog d = new Dog(dogBreed); ← Cria um  
        novo Dog
```

```
        sc.setAttribute("dog", d); ← Usa o contexto para obter o  
        parâmetro init
```

```
}
```

Uso o contexto para especificar um atributo
(um par nome/objeto) Dog. Agora, outros
trechos da aplicação serão capazes de receber
o valor do atributo (o Dog)

```
    public void contextDestroyed(ServletContextEvent event) {
```

```
        // nada a fazer aqui
```

```
}
```



Não precisamos de nada aqui. O Dog não precisa
ser limpo... quando o contexto termina, significa
que toda a aplicação vai finalizar, incluindo o Dog

Inicializando Objetos

```
package com.example;

public class Dog {
    private String breed;

    public Dog(String breed) {
        this.breed = breed;
    }

    public String getBreed() {
        return breed;
    }
}
```

Nada de especial aqui. Só uma classe Java simples

(Usaremos os parâmetros init do contexto como argumento para o construtor Dog.)

Nosso servlet receberá o Dog do contexto (o Dog, que o listener configura como atributo), chamará o método `getBreed()` do Dog e copiará a raça na resposta, para que possamos vê-la no browser

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

public class ListenerTester extends HttpServlet {

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("test context attributes set by listener<br>");

        out.println("<br>");

        Dog dog = (Dog) getServletContext().getAttribute("dog");
    }
}
```

↑
Se algo não funcionar, é AQUI que
descobriremos... vamos receber um enorme
NullPointerException se tentarmos chamar
o getBreed() e não houver nenhum Dog

Nada de especial até aqui... apenas um
servlet comum

Aqui obtemos o
Dog através do
ServletContext. Se
o listener funcionou,
o Dog vai estar ali
ANTES que o método
service seja chamado
pela primeira vez

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <servlet>
        <servlet-name>ListenerTester</servlet-name>
        <servlet-class>com.example.ListenerTester</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ListenerTester</servlet-name>
        <url-pattern>/ListenTest.do</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>breed</param-name>
        <param-value>Great Dane</param-value>
    </context-param>

    <listener>
        <listener-class>
            com.example.MyServletContextListener
        </listener-class>
    </listener>
</web-app>
```

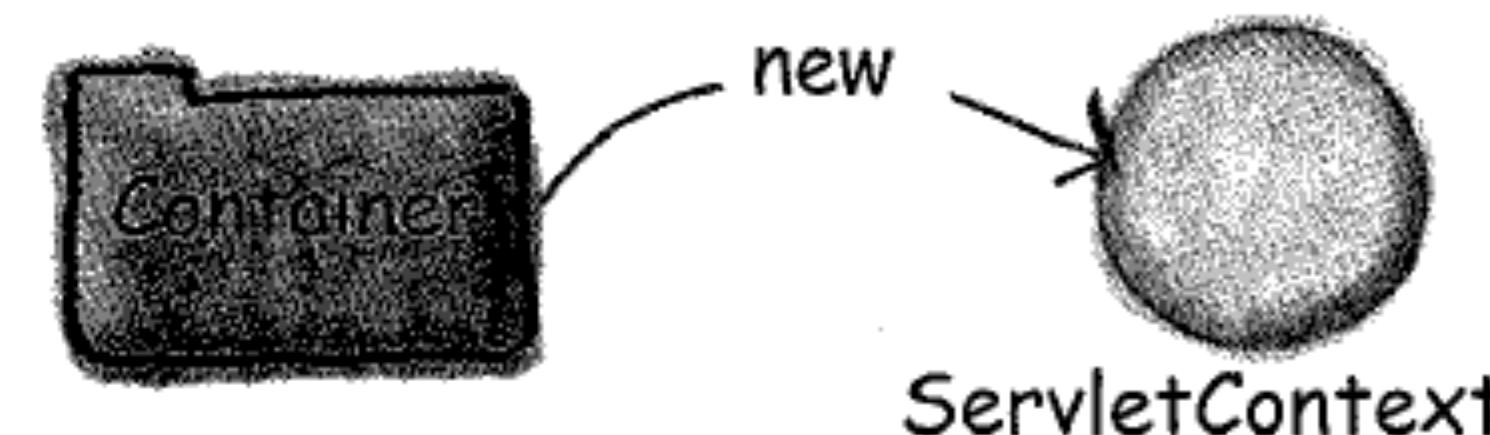
Precisamos de um parâmetro init
do contexto para a aplicação.
O listener precisa dele para
construir o Dog

Registre esta classe como um listener. IMPORTANTE:
elemento <listener> NÃO vai dentro de um elemento <servlet>. Isto não funcionaria, pois o listener do contexto é para um evento ServletContext (que significa para toda a aplicação). A questão toda é inicializar a aplicação ANTES que qualquer servlet seja inicializado.

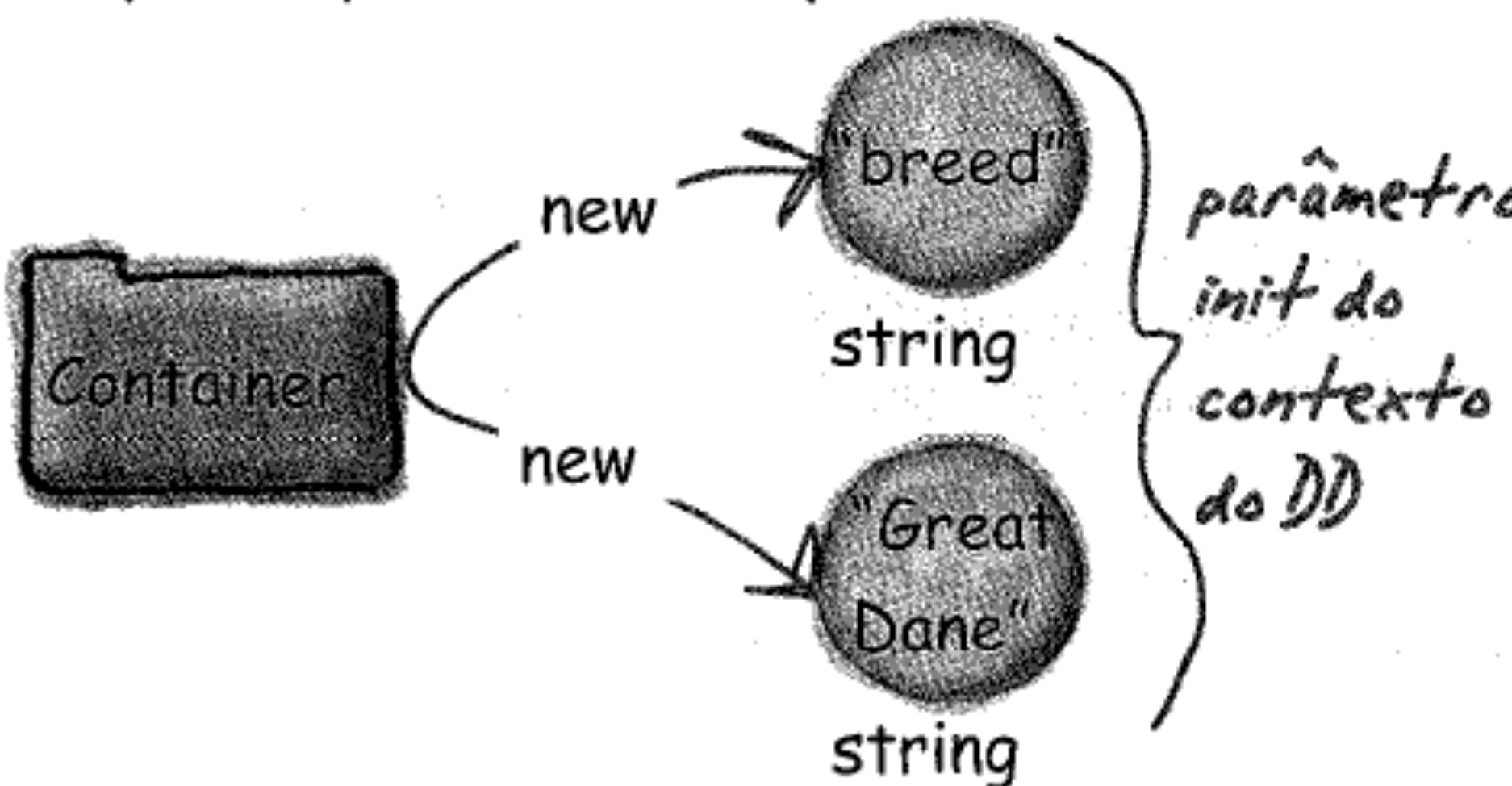
- 1** O Container lê o Deployment Descriptor para esta aplicação, inclusive os elementos <listener> e <context-param>.



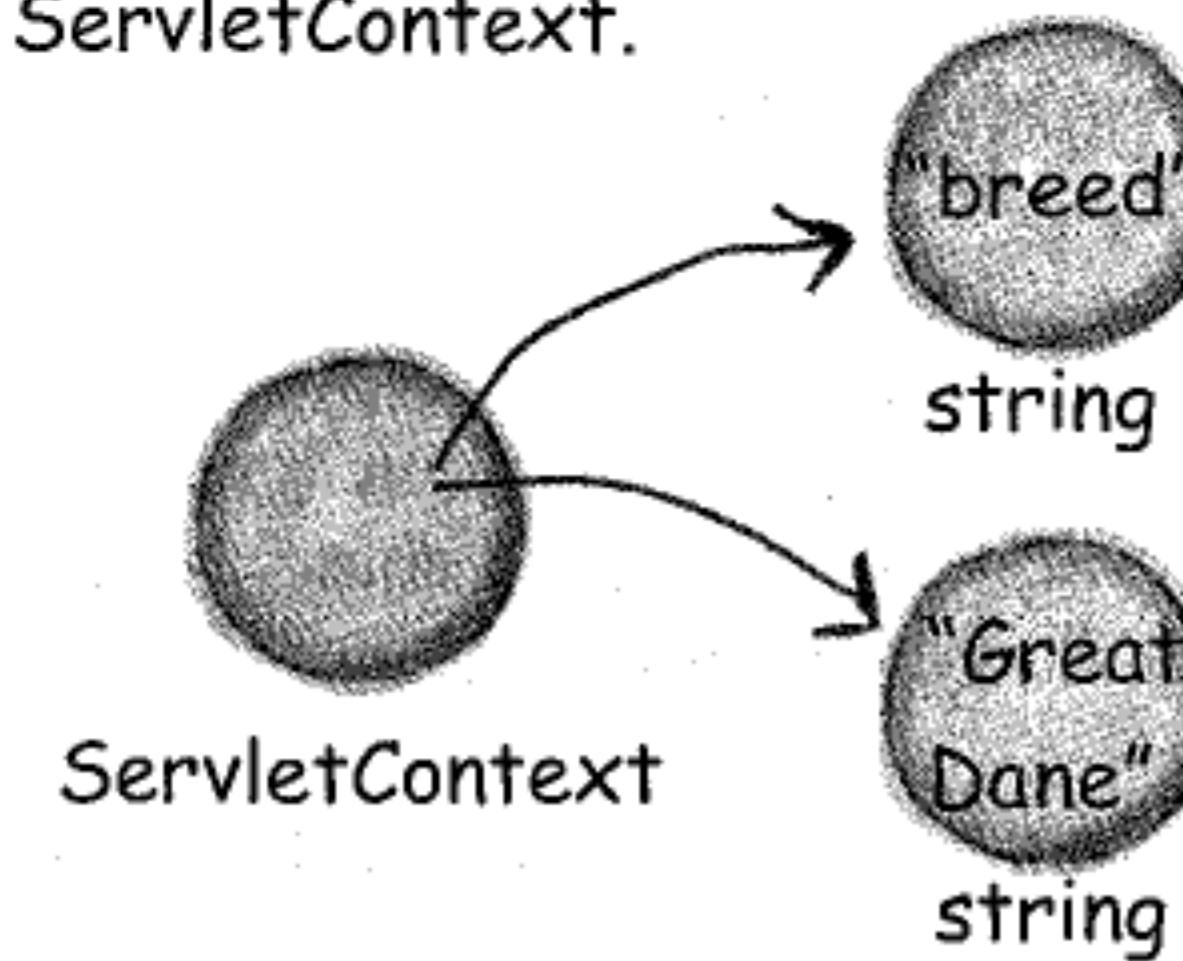
- 2** O Container cria um novo ServletContext, que todas as partes da aplicação compartilharão.



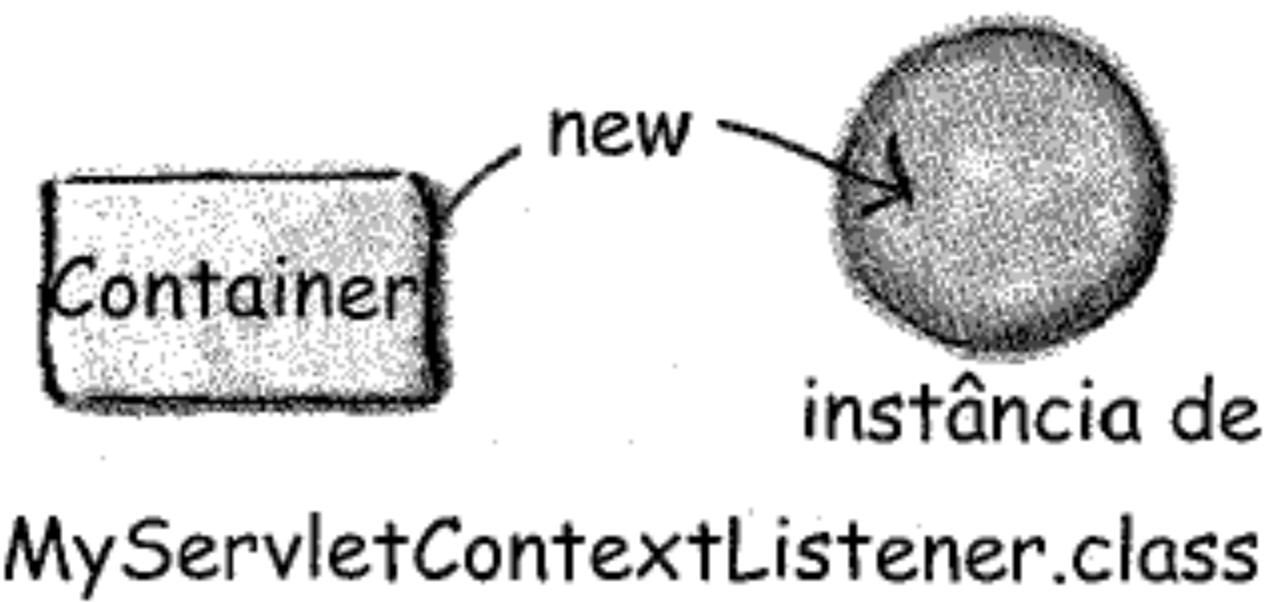
- 3** O Container cria um par de Strings nome/valor para cada parâmetro init do contexto. Suponha que tenhamos apenas um.



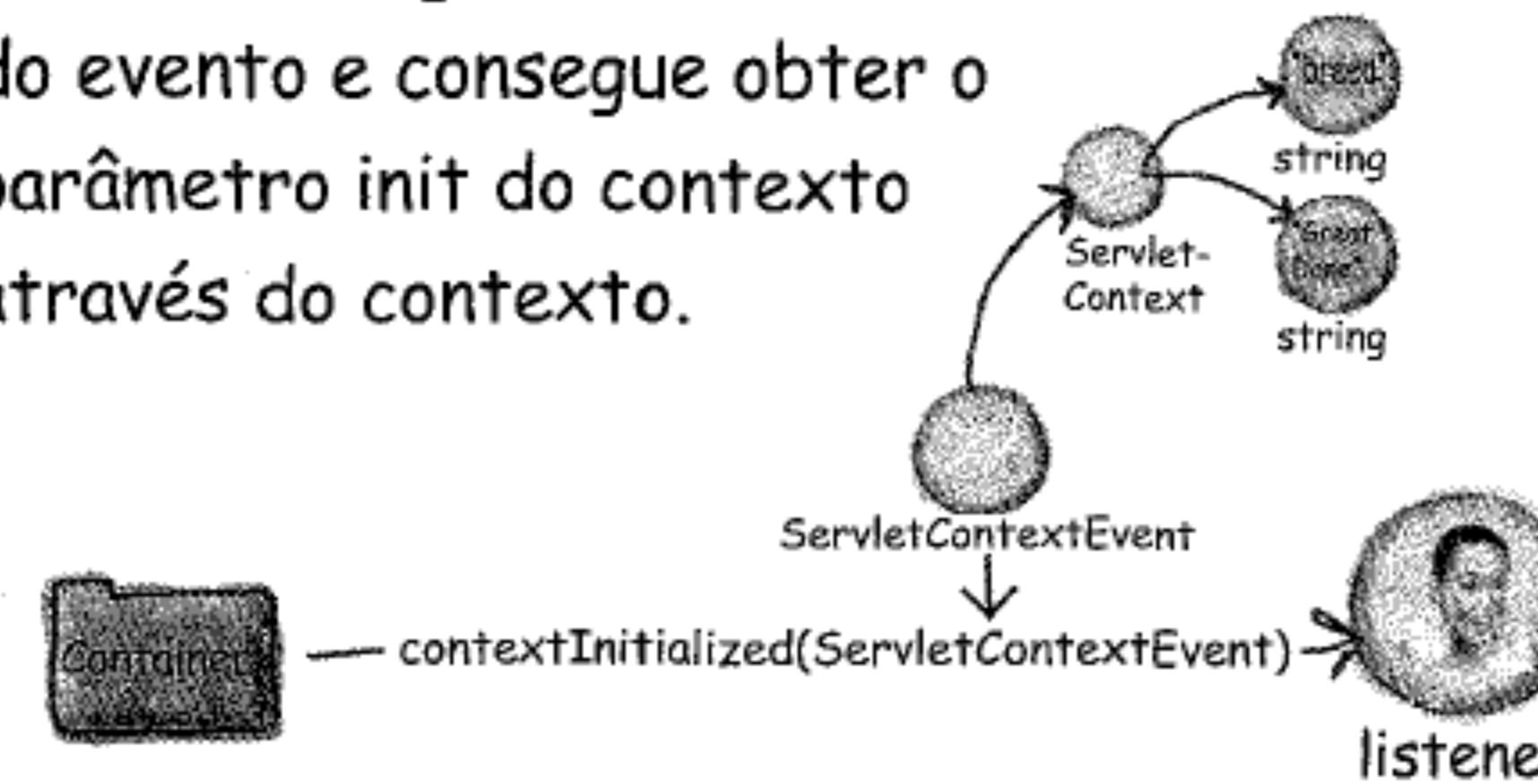
- 4** O Container dá aos parâmetros nome/valor as referências do ServletContext.



- 5 O Container cria uma nova instância da classe MyServletContextListener.



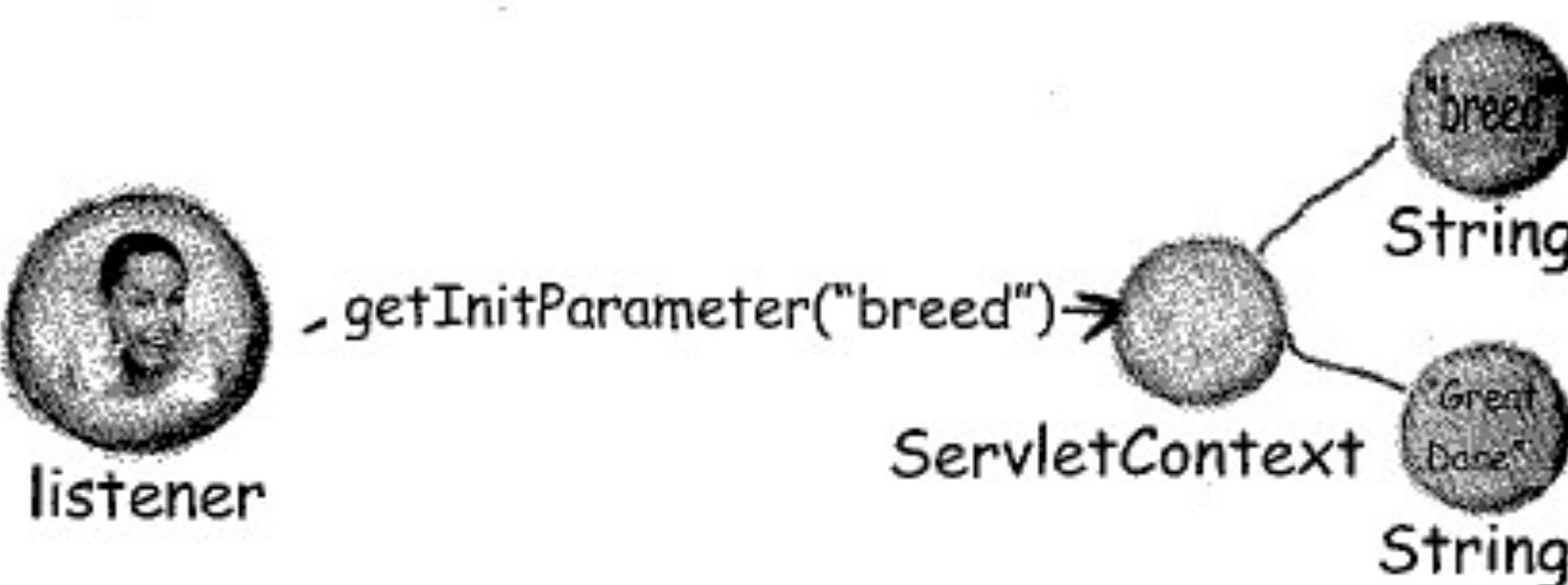
- 6 O Container chama o método contextInitialized() do listener, passando em um novo ServletContextEvent. O objeto de evento tem uma referência para o ServletContext, então o código que trata o evento consegue obter o contexto através do evento e consegue obter o parâmetro init do contexto através do contexto.



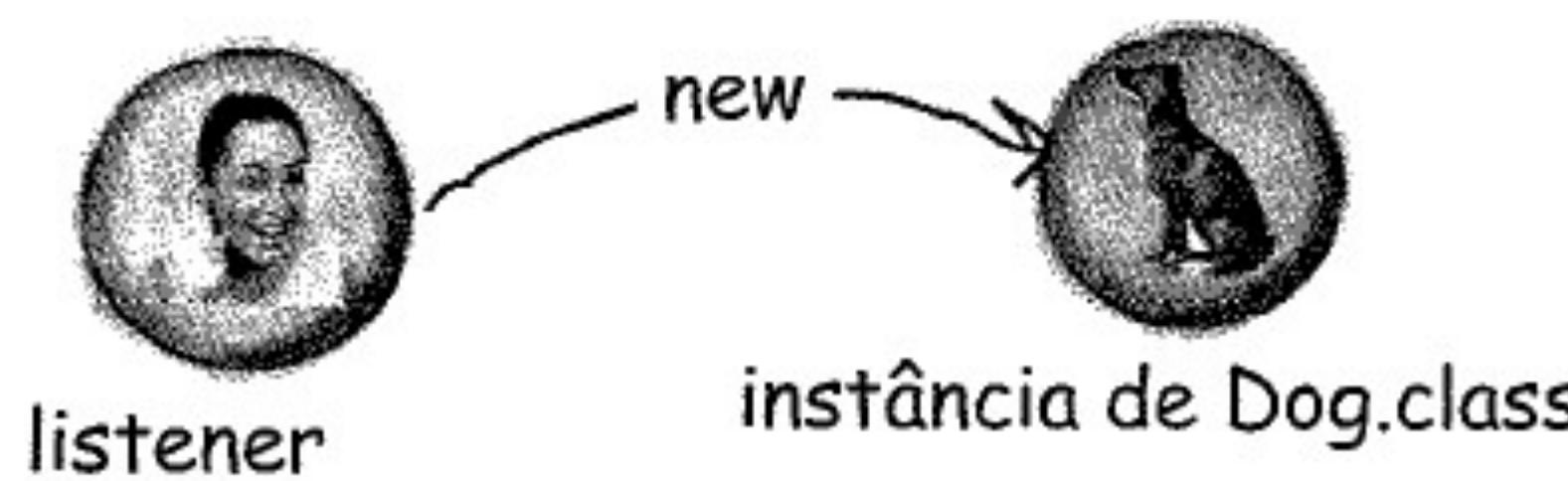
- 7** O listener solicita ao ServletContextEvent uma referência para o ServletContext.



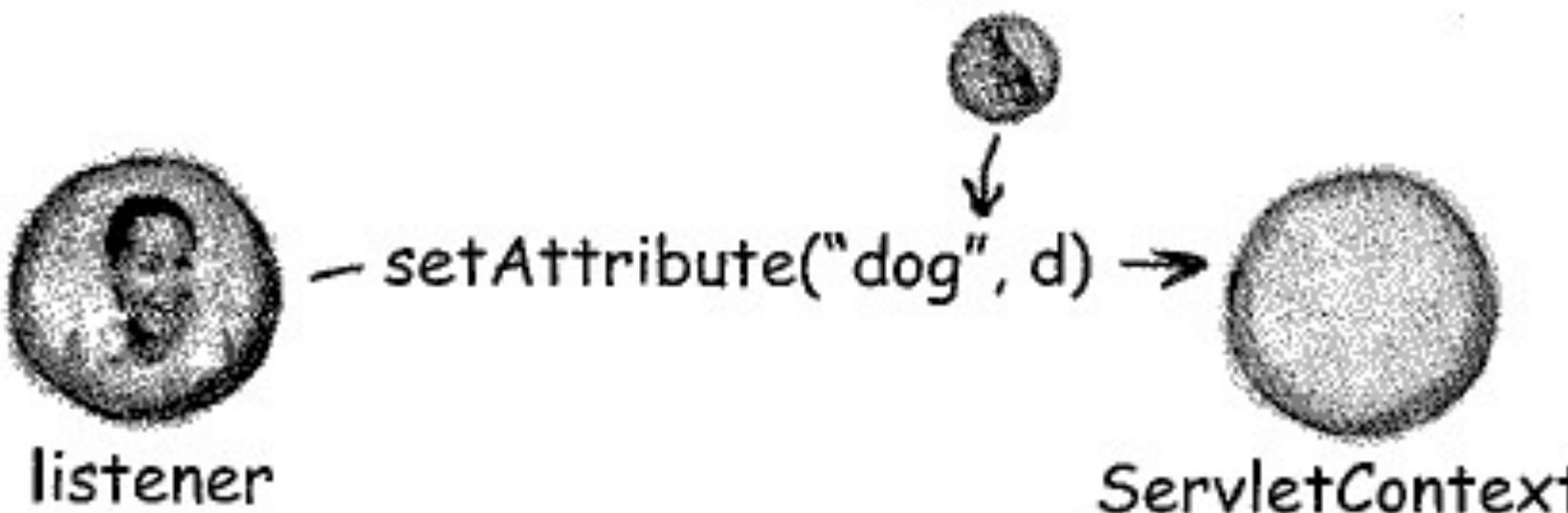
- 8** O listener solicita ao ServletContext o parâmetro init do contexto "breed".



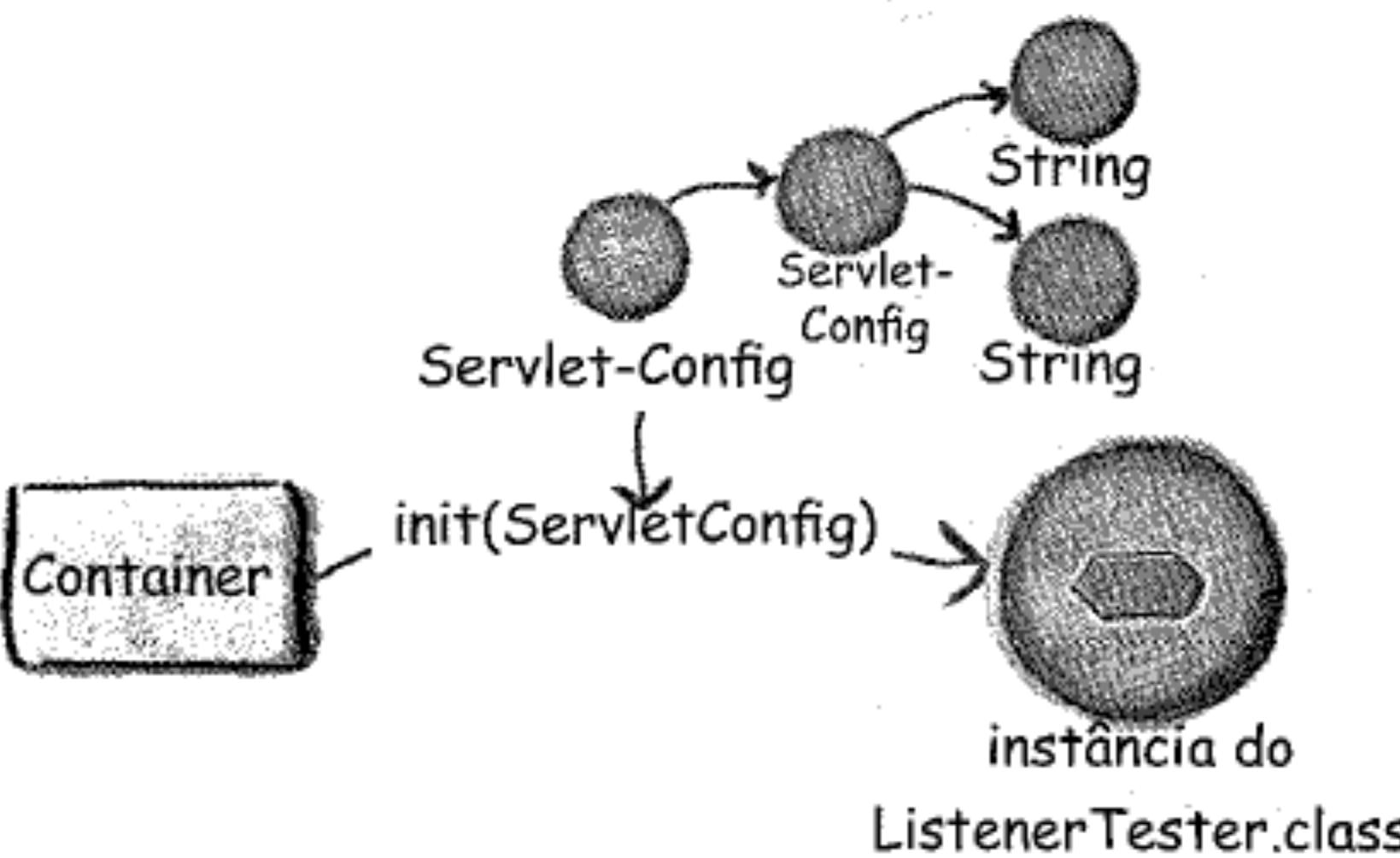
- 9** O listener usa o parâmetro init para construir um novo objeto Dog.



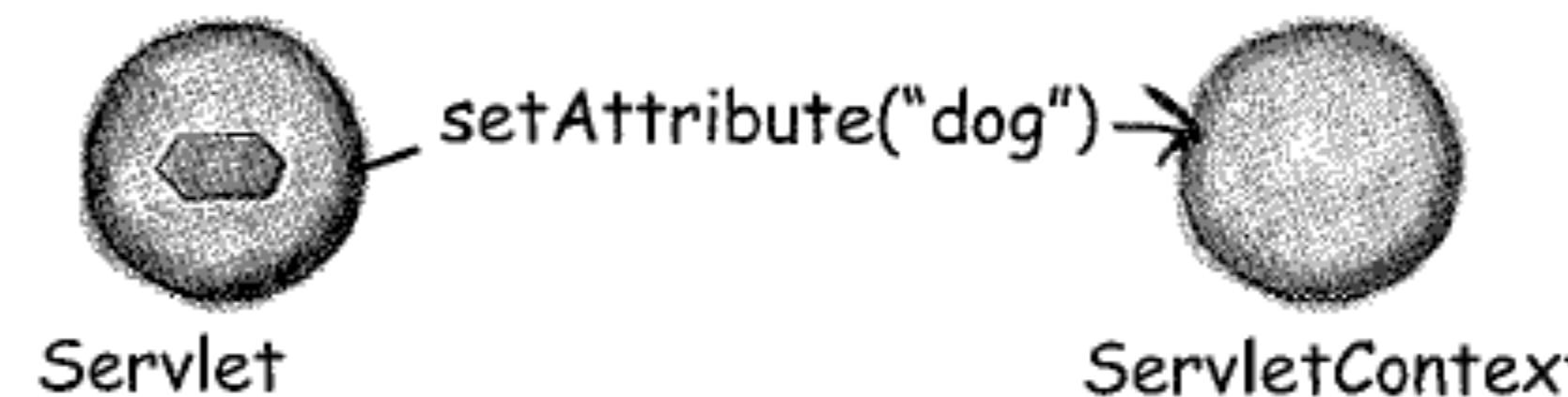
- 10** O listener configura o Dog como um atributo no ServletContext.



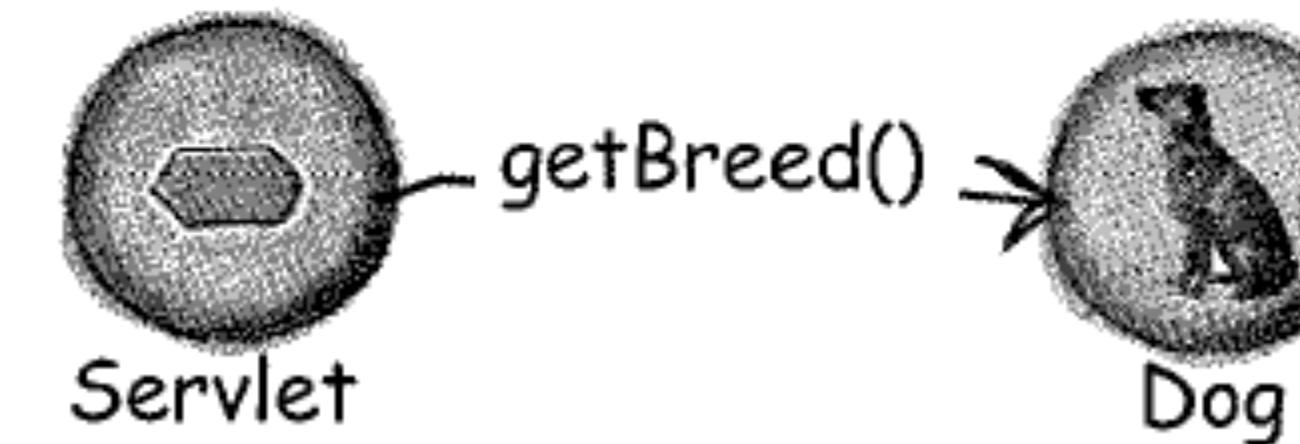
11 O Container cria um novo Servlet (isto é, cria um novo ServletConfig com parâmetros init, dá ao ServletConfig uma referência para o ServletContext e chama o método init() do Servlet).



12 O servlet recebe uma solicitação e pede ao ServletContext o atributo "dog".



13 O servlet chama o getBreed() no Dog (e exibe-o no HttpResponse).



Inicializando Objetos

- Outros tipos de listeners oferecidos pela API de Servlets:
 - `AttributeListener`;
 - `HttpSessionListener`;
 - `ServletRequestListener`;
 - `ServletRequestAttributeListener`;

Inicializando Objetos

- Outros tipos de listeners oferecidos pela API de Servlets:
 - `HttpSessionBindingListener`;
 - `HttpSessionAttributeListener`;
 - `HttpSessionActivationListener`;

Compartilhando Informações

- Podemos compartilhar informações entre os componentes da aplicação através de atributos;
- Estes atributos são cadastrados no contexto da aplicação;
- Qualquer componente da aplicação pode recuperar, alterar ou remover este atributo;

Compartilhando Informações



*Quem pode ver este
quadro de avisos?*

*Quem pode receber
e configurar os
atributos?*

Um atributo é como um objeto preso em um quadro de avisos. Alguém o afixou no quadro para que outros possam pegá-lo.

As grandes perguntas são: quem tem acesso ao quadro de avisos e quanto tempo ele permanece lá? Em outras palavras, qual é o escopo do atributo?

Compartilhando Informações

- Todo atributo é identificado através de um nome;
- Todo atributo é armazenado na forma de um objeto;

Compartilhando Informações

- Atributos oferecem as seguintes vantagens:
 - Podem ser criados por qualquer componente da aplicação e em qualquer momento;
 - Podem ser acessados, modificados ou excluídos por outros componentes da aplicação;
 - Podem armazenar qualquer tipo de objeto;
 - ✓ Diferente dos parâmetros de inicializam, em que o valor tem que ser String;

		Atributos	Parâmetros
Tipos	<p>Application/context Request Session</p> <p><i>Não há nenhum atributo específico para o servlet (basta usar uma variável da instância)</i></p>		<p>Application/parâmetros init do contexto Parâmetros da solicitação Parâmetros init do <u>servlet</u></p> <p><i>Não existem parâmetros da sessão!</i></p>
Método para configuração	setAttribute(nome da String, valor do Objeto)		Você NÃO PODE configurar os parâmetros init da Application e do Servlet – eles são configurados no DD, lembra? (Com os parâmetros da Solicitação, você pode ajustar a query String, mas é diferente.)
Tipo de retorno	Objeto		<p>String</p> 
Método para obtenção	<p>ServletContext.getAttribute (String name)</p> <p><i>Não se esqueça de que os atributos devem ser convertidos, já que o tipo de retorno é Objeto</i></p>		ServletContext.getAttribute(String name)

Compartilhando Informações

- Podemos criar atributos com três diferentes escopos:
 - Requisição, sessão e aplicação;
- O escopo do atributo depende do objeto no qual o mesmo é adicionado;

Compartilhando Informações

- Atributos de requisição:
 - São visíveis apenas para o objeto que manipula a requisição;
 - O atributo deixa de existir no momento em que a requisição é processada;
 - São embutidos na classe `HttpRequest`;

Compartilhando Informações

- Atributos de requisição:



Acessível apenas para aqueles com acesso a um *ServletRequest* específico

Compartilhando Informações

- Atributos de sessão:
 - São visíveis para todos os componentes que compõem a sessão com o usuário;
 - O atributo deixa de existir no momento em que o usuário encerra a sua sessão com a aplicação;
 - São embutidos na classe **HttpSession**;

Compartilhando Informações

- Atributos de sessão:

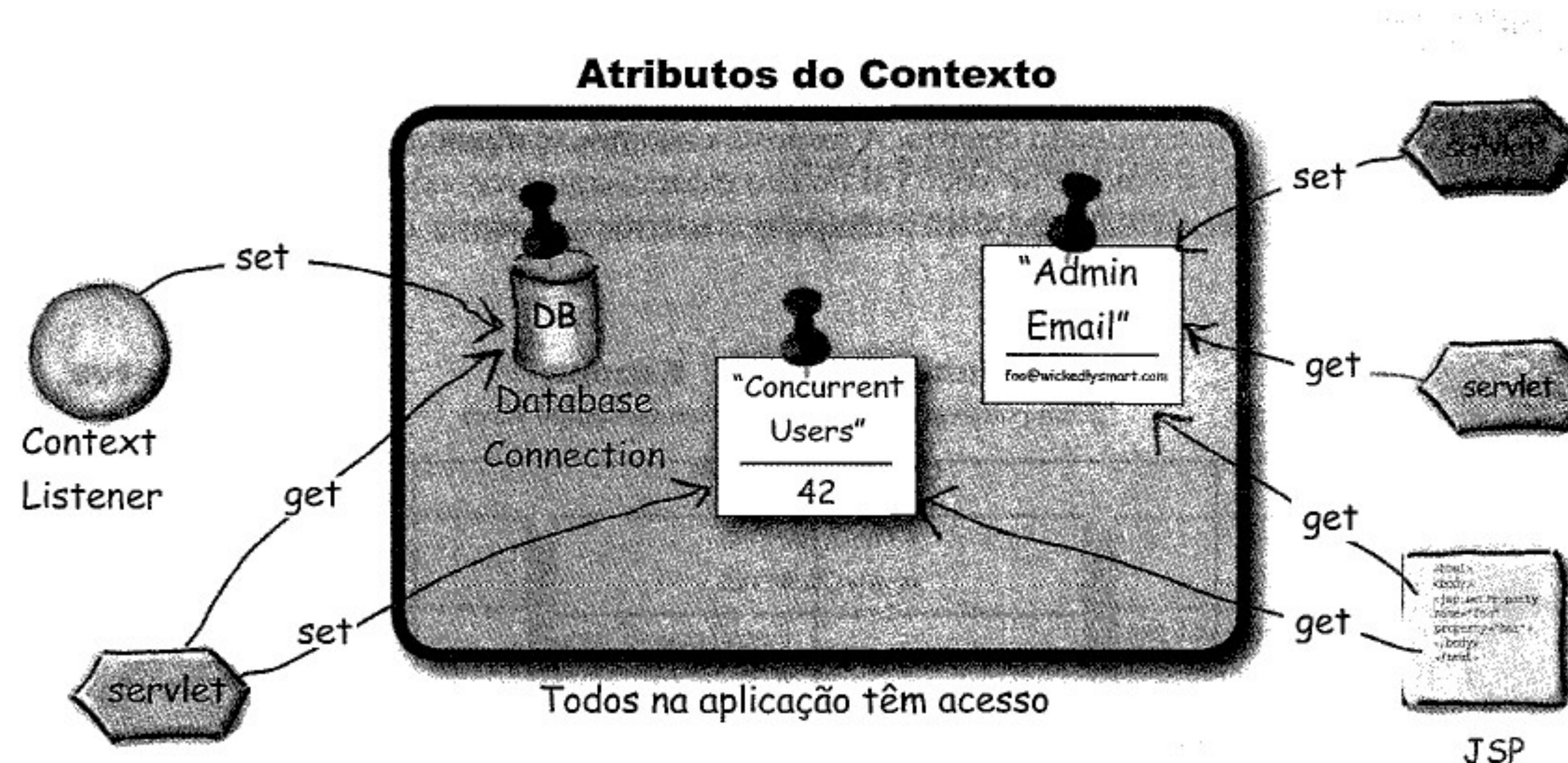


Compartilhando Informações

- Atributos de aplicação:
 - São visíveis para todos os componentes que compõem uma mesma aplicação;
 - ✓ Mesmo que relacionados a sessões com usuários diferentes;
 - O atributo deixa de existir no momento em que a aplicação é encerrada;
 - São embutidos na classe **ServletContext**;

Compartilhando Informações

- Atributos de aplicação:



Compartilhando Informações

- Métodos usados para a manipulação de atributos:
 - Alguns métodos podem ser usados para a criação e manipulação de atributos;
 - Estes métodos são idênticos para os três tipos de objetos que podem encapsular atributos;

Compartilhando Informações

- Métodos usados para a manipulação de atributos:
 - `getAttributeNames()`:
 - ✓ Retorna uma enumeração contendo os nomes de todos os atributos definidos para o objeto;
 - `getAttribute(String attributeName)`:
 - ✓ Retorna o valor associado a um determinado atributo, ou null, caso não exista nenhum atributo com o nome especificado;

Compartilhando Informações

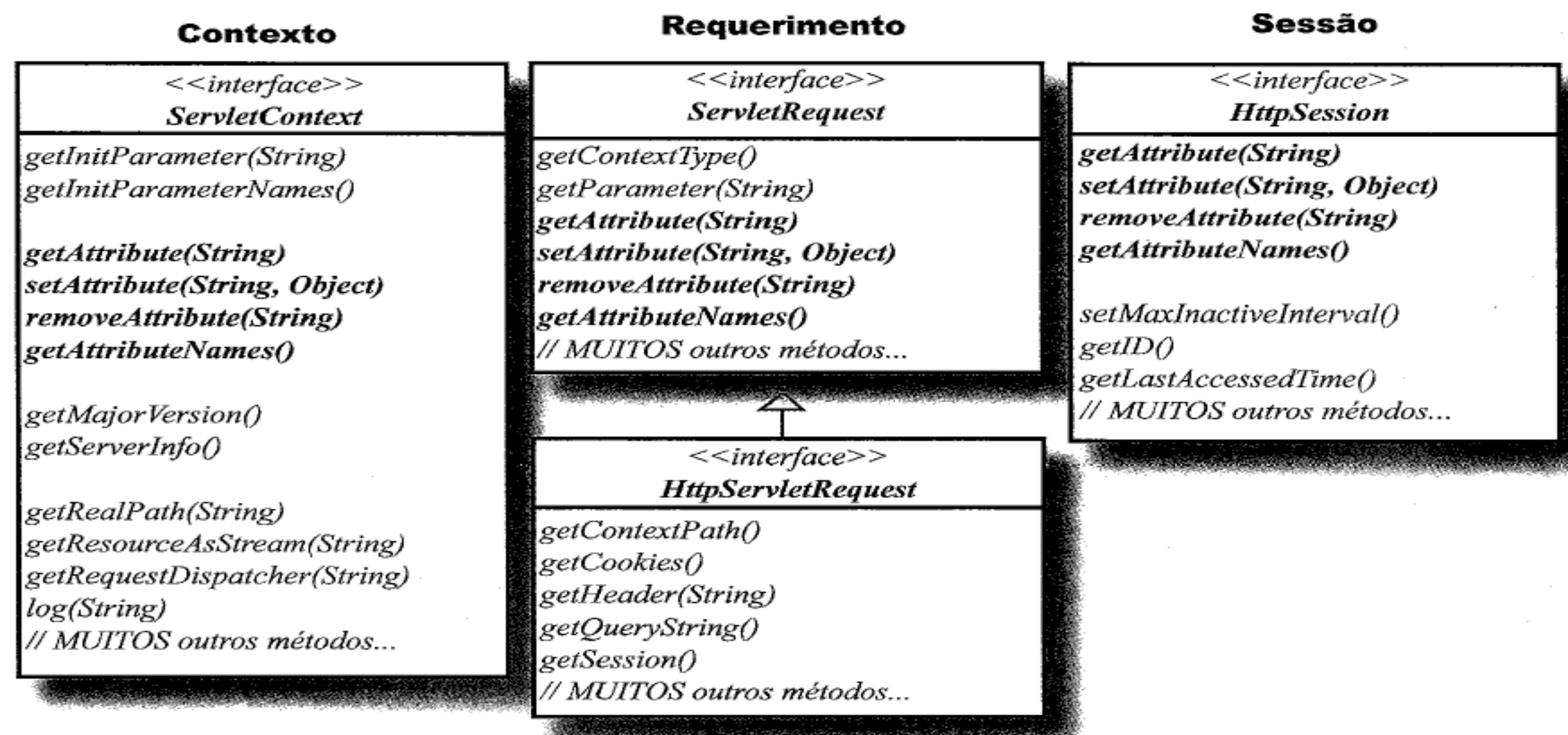
- Métodos usados para a manipulação de atributos:
 - `setAttribute(String attributteName, Object value):`
 - ✓ Modifica o valor de um atributo já existente;
 - ✓ Caso não exista nenhum atributo com o nome informado, o mesmo é criado e associado a este objeto;

Compartilhando Informações

- Métodos usados para a manipulação de atributos:
 - `removeAttribute(String attributteName):`
 - ✓ Remove um um atributo já existente;
 - ✓ Após esta remoção, futuras chamadas para obter o valor deste atributo retornarão o valor null;

Compartilhando Informações

- Métodos usados para a manipulação de atributos:



Listener que inicializa um atributo

```
@WebListener
public class InicializaObjetoListener implements ServletContextListener{

    @Override
    public void contextInitialized(ServletContextEvent event) {
        //Instanciando o objeto
        Calendar rightNow = Calendar.getInstance();

        //Compartilhando o objeto com os demais componentes da aplicação
        event.getServletContext().setAttribute("startTime", rightNow);
    }

    @Override
    public void contextDestroyed(ServletContextEvent event) {
        event.getServletContext().removeAttribute("startTime");
    }
}
```

Servlet que recupera e imprime o valor do atributo

```
@WebServlet("/mostraHora")
public class MostraHoraInicializacaoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Exemplo </title></head>");
        out.println("<body>");
        //Recuperando o valor do atributo
        Calendar startTime = (Calendar) getServletContext().getAttribute("startTime");
        //Imprimindo a hora de inicialização
        out.println("<p> Horário de inicialização da aplicação: "+startTime.getTime()+"</p>");

        out.println("</body></html>");
    }

}
```

Compartilhando Informações

- Cuidado: atributos de contexto podem ser utilizados simultaneamente pela aplicação
- Problemas de sincronização e consistência podem acontecer;



Compartilhando Informações

- Vamos supor a seguinte situação:
 - Uma thread A cria no contexto o atributo “numero”, e atribui a ele o valor 25;
 - Uma thread B, da mesma aplicação, cria o mesmo atributo no contexto e atribui a ele o valor 50;
 - Mais tarde, a thread A recupera o valor do atributo número, obtendo o valor 50;
 - **Resultado: O valor encontrado é inconsistente;**

Compartilhando Informações

- Neste tipo de situação, para garantir a consistência dos dados, precisamos de sincronização;



Compartilhando Informações

- No entanto, temos que ter cuidado para não acabar com o desempenho da aplicação;
 - Blocos sincronizados impõem um grande overhead à aplicação;
 - Os recursos envolvidos no bloco são bloqueados para garantir a serialização do bloco;



Compartilhando Informações

- Uma solução possível seria sincronizar o acesso aos métodos `doGet` e `doPost`, o que seria péssimo;
 - Usando esta solução, o container não criará threads simultâneas do mesmo servlet;
 - Isto acaba com a escalabilidade da aplicação;

Compartilhando Informações

- A ideia é sincronizar (que é inevitável), mas de forma que o bloco sincronizado seja o menor possível;
 - Devemos sincronizar apenas o que for realmente necessário, para minimizar ao máximo o impacto da sincronização no desempenho da aplicação;

Compartilhando Informações

- Assim, sincronizamos apenas os trechos do método que envolvem os recursos que podem gerar a inconsistência:
 - No nosso caso, os comandos que acessam e alteram o valor dos atributos;
 - Para isso, bloqueamos o contexto, que é o recurso compartilhado, durante o processo de atualização;

```
public class SynchronizedServlet extends HttpServlet{

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        ServletContext context = this.getServletContext();
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Meu primeiro Servlet </title></head>");
        out.println("<body>");
        synchronized (context){          Sincronizando o acesso a atributos do contexto
            context.setAttribute("numero", "15");
            context.setAttribute("cliente", "John");
            out.println("<p> <b>Numero:</b> "+context.getAttribute("numero")+"</p>");
            out.println("<p> <b>Cliente:</b> "+context.getAttribute("cliente")+"</p>");
        }
        out.println("</body>");
        out.println("</html>");
    }

}
```

Compartilhando Informações

- Atributos com escopo de sessão podem ter o mesmo problema;
 - Isto pode ser evitado da mesma forma, bloqueando o objeto que representa a sessão;
- Atributos que são protegidos do acesso de outras threads por sincronização são chamados **thread-safe**;

Despachando Requisições

- Um servlet pode transferir o controle da requisição recebida para outros componentes da aplicação;
- Esta transferência pode ser feita de forma temporária ou definitiva;
- Requisições são transferidas através do **RequestDispatcher**;

Despachando Requisições

- A transferência de controle é um recurso importante, pois nos permite escrever servlets menores:
 - Cada servlet desempenha apenas uma tarefa específica e bem definida;
 - Isto aumenta a possibilidade de reuso do servlet por parte dos outros componentes da aplicação;

Despachando Requisições

- O RequestDispatcher:
 - Os objetos que implementam esta interface encaminham a requisição do usuário para outros componentes da aplicação;
 - Estes objetos podem ser obtidos através do **ServletContext** ou do **ServletRequest**;

Despachando Requisições

- Obtendo um RequestDispatcher:
 - `getRequestDispatcher(String resourcePath);`
 - ✓ Usamos este método para obter um despachante para o recurso informado como parâmetro;
 - ✓ O recurso passado como parâmetro é o componente que vai processar a requisição;
 - ✓ O recurso informado pode ser um ser um servlet, uma página JSP, uma página HTML, etc;

Despachando Requisições

- Obtendo um RequestDispatcher:
 - `getRequestDispatcher(String resourcePath);`
 - ✓ Caso o recurso seja um servlet, o caminho corresponde ao padrão de URL definido para o mesmo;
 - ✓ Caso o recurso seja uma página, devemos informar a URL da mesma;
 - ✓ Esta URL pode ser completa ou relativa;

Despachando Requisições

- Obtendo um RequestDispatcher:
 - `getRequestDispatcher(String resourcePath);`
 - ✓ Exemplo: `getRequestDispatcher("/query.jsp");`
 - ✓ Se o método for chamado no `ServletContext`, a página deve estar no diretório raiz da aplicação;
 - ✓ Se o método for chamado no `ServletRequest`, a página deve estar no mesmo diretório da página que fez a requisição;

Despachando Requisições

- Obtendo um RequestDispatcher:
 - `getRequestDispatcher(String resourcePath);`
 - ✓ No caso de URLs relativas, o caminho inicial muda dependendo do tipo de objeto pelo qual o despachante está sendo obtido;
 - ✓ No caso do `ServletContext`, o caminho raiz é o diretório raiz da aplicação;
 - ✓ No caso do `ServletRequest`, o caminho raiz é o caminho da página que fez a requisição;

Despachando Requisições

- Obtendo um RequestDispatcher:
 - `getRequestDispatcher(String resourcePath);`
 - ✓ Caso o recurso informado não seja encontrado, o método retorna um valor nulo;

Despachando Requisições

- Obtendo um RequestDispatcher:
 - `getNamedDispatcher(String resourcePath);`
 - ✓ Retorna um despachante para um servlet cadastrado na aplicação;
 - ✓ Um valor null é retornado caso o servlet não seja encontrado;

Despachando Requisições

- O RequestDispatcher oferece dois métodos para encaminhar a requisição;
 - `include(ServletRequest, ServletResponse);`
 - `forward(ServletRequest, ServletResponse);`

Despachando Requisições

- O método include:
 - Este método concede ao componente que recebe a requisição apenas o controle temporário da mesma;
 - Depois que este componente encerra sua tarefa, a requisição retorna para o servlet que fez o despacho;

Despachando Requisições

- O método include:
 - O componente para o qual a requisição foi despachada tem algumas restrições:
 - ✓ Ele pode editar o objeto que representa a resposta;
 - ✓ Ele não pode modificar qualquer informação de cabeçalho da requisição;
 - ✓ Ele pode acessar o objeto que representa a sessão, mas não pode gerar cookies;

Despachando Requisições

- Vamos ver um servlet que gera a página inicial do website de uma loja;
- Vamos supor que o conteúdo da página é dividido em três partes: um banner, o conteúdo principal e os copyrights;

Despachando Requisições

- O banner e os copyrights são fixos e devem aparecer em todas as páginas da aplicação;
- O conteúdo principal varia para cada página da aplicação;

Despachando Requisições

- Neste caso, criar um único servlet para gerar todo o conteúdo de cada página não é uma boa solução;
 - Imagine o que vai acontecer no dia em que a loja decidir alterar o banner e/ou os copyrights;



Despachando Requisições

- Neste caso, vamos gerar a página inicial através de três servlets:
 - Um para gerar apenas o banner;
 - Um para gerar apenas os copyrights;
 - Um para gerar o conteúdo principal e controlar a geração da página;

Despachando Requisições

- O servlet que gera o banner:

```
@WebServlet("/banner")
public class BannerServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        out.print("<p align = \"center\"> Lojas Santa Maria. "
            + "Aqui o cliente sempre tem prioridade. </p>");
    }

}
```

Despachando Requisições

- O servlet que gera o copyright:

```
@WebServlet("/copyright")
public class CopyrightServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        out.print("<p align=\"right\"><h2> Copyritghs: Lojas Santa Maria Ltda</h2></p>");
    }
}
```

O servlet que gera a página principal, com a ajuda dos outros servlets

```
@WebServlet("/inicial")
public class SantaMariaServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Lojas Santa Maria </title></head>");
        out.println("<body>");

        //Obtendo um despachante para o servlet que gera o banner e despachando para o mesmo
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner");
        dispatcher.include(request, response);

        out.println("<p align=\"center\"> <h1> Esta é a página inicial das lojas Santa Maria. "
            + "</h1></p>");

        //Despachando para o servlet que gera o copyright
        dispatcher = getServletContext().getRequestDispatcher("/copyright");
        dispatcher.include(request, response);

        out.println("</body></html>");
    }
}
```

Despachando Requisições

- Resultado:



Despachando Requisições

- O método include:
 - Quando o include é usado para despachar uma requisição para uma página HTML ou JSP, o conteúdo da página é incluído na resposta ao cliente;
 - Esta característica é particularmente útil quando o conteúdo a ser adicionado é estático;

Despachando Requisições

- Vamos refazer o exemplo da página inicial da loja;
- Agora, os conteúdos do banner e do copyright serão definidos em arquivos HTML;

Despachando Requisições

- O arquivo banner.html:

```
<p align = "center">  
    Lojas Santa Maria. Aqui o cliente sempre tem prioridade.  
</p>
```

- O arquivo copyright.html:

```
<p align="right\">  
    <h2> Copyritghs: Lojas Santa Maria Ltda</h2>  
</p>
```

O servlet que gera a página principal, com a ajuda dos outros servlets

```
@WebServlet("/ inicialHtml")
public class SantaMariaServletHtml extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Lojas Santa Maria </title></head>");
        out.println("<body>");

        //Obtendo um despachante para o arquivo HTML que contém o banner
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner.html");
        dispatcher.include(request, response);

        out.println("<p align=\"center\"> <h1> Esta é a página inicial das lojas Santa Maria."
            + "</h1></p>");

        //Obtendo um despachante para o arquivo HTML que contém o copyright
        dispatcher = getServletContext().getRequestDispatcher("/copyright.html");
        dispatcher.include(request, response);

        out.println("</body></html>");
    }
}
```

Despachando Requisições

- O método **forward**:
 - Este método concede ao componente que recebe a requisição o controle definitivo da mesma;
 - Depois que o método é chamado a requisição não retorna mais para o servlet que a despachou;
 - O componente que recebe a requisição é responsável por finalizar seu processamento ou encaminhá-la para outro componente da aplicação;

Despachando Requisições

- O método forward:
 - Vamos supor que estamos implementando uma aplicação web para uma loja de livros e que precisamos implementar uma funcionalidade que permita que o usuário realize uma consulta por uma palavra-chave;
 - Vamos supor também que a aplicação tem outros tipos de consulta e que todos os tipos de consulta mostram o resultado da mesma forma;

Despachando Requisições

- O método forward:
 - Neste cenário, implementar a busca e a exibição do resultado em um único servlet não é uma boa solução;
 - Mudanças no layout da página que mostra os resultados de uma consulta vão exigir a mudança de todos os servlets de consulta;

Despachando Requisições

- O método forward:
 - Vamos implementar a funcionalidade através de dois servlets;
 - O primeiro apenas recebe a palavra-chave indicada pelo usuário e realiza a consulta no banco de dados;
 - O segundo apenas apresenta os resultados da consulta para o usuário;

```
@WebServlet("/busca")
public class RealizaBuscaServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

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

        //Realizando a consulta
        String key = request.getParameter("key");
        List<Book> bookList = searchByKey(key);

        //Colocando o resultado como atributo da requisição
        request.setAttribute("result", bookList);

        //Despachando para o servlet que vai mostrar o resultado
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/mostraResultado");
        dispatcher.forward(request, response);
    }

    private List<Book> searchByKey(String key) {...19 linhas }

}
```

```
public class MostraResultadoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

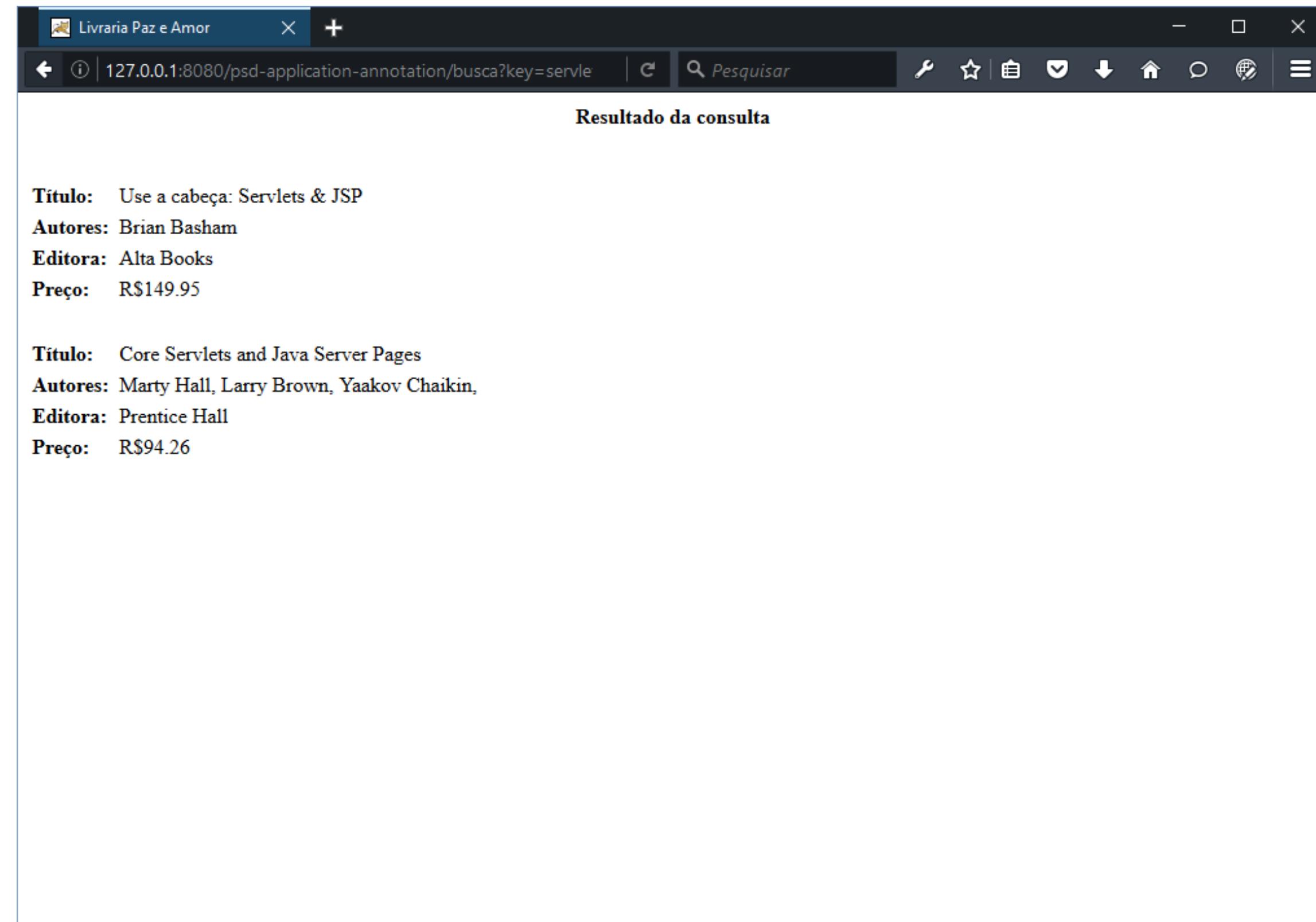
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Livraria Paz e Amor </title></head>");
        out.println("<body> <p align=\"center\"><b>Resultado da consulta </b></p>");
        List<Book> bookList = (List<Book>) request.getAttribute("result");
        out.println("<table>");

        for(Book book : bookList){
            out.println("<tr>");
            out.println("<tr><td><b> Título: </b><td/><td>" +book.getTitle()+"</td></tr>");
            out.println("<tr><td><b> Autores: </b><td/><td>" +book.getAuthors()+"</td></tr>");
            out.println("<tr><td><b> Editora: </b><td/><td>" +book.getPublisher()+"</td></tr>");
            out.println("<tr><td><b> Preço: </b><td/><td>R$" +book.getPrice()+"</td></tr>");
            out.println("<tr><td> &ampnbsp<td>&ampnbsp</td><td>&ampnbsp</td></tr>");
        }
        out.println("</table>");
        out.println("</body></html>");
    }

}
```

Despachando Requisições

- Resultado final:



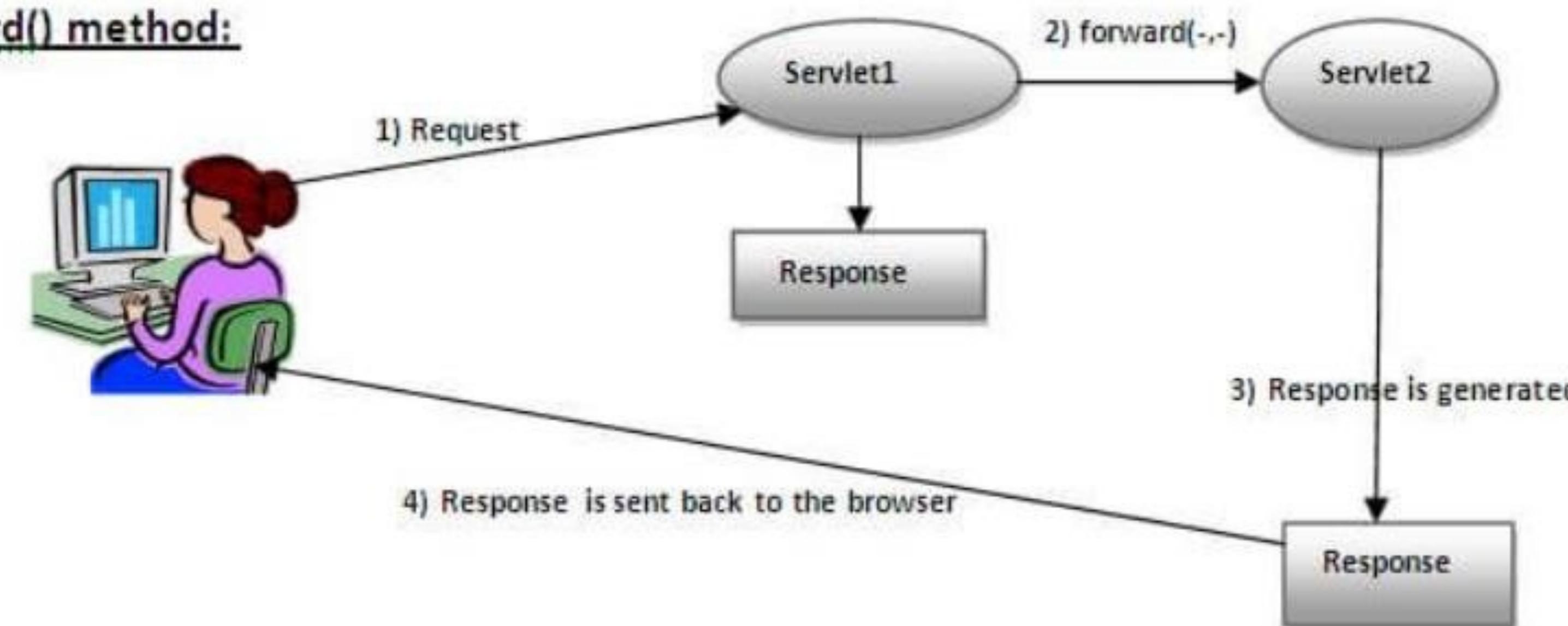
Revisão

- Métodos da interface RequestDispatcher interface
 - **public void forward(ServletRequest request,ServletResponse response)**
 - Encaminha uma solicitação de um servlet para outro recurso (servlet, arquivo JSPou arquivo HTML) no servidor.
 - **public void include(ServletRequest request,ServletResponse response)**
 - Inclui o conteúdo de um recurso (servlet, página JSPou arquivo HTML) na resposta.

Revisão

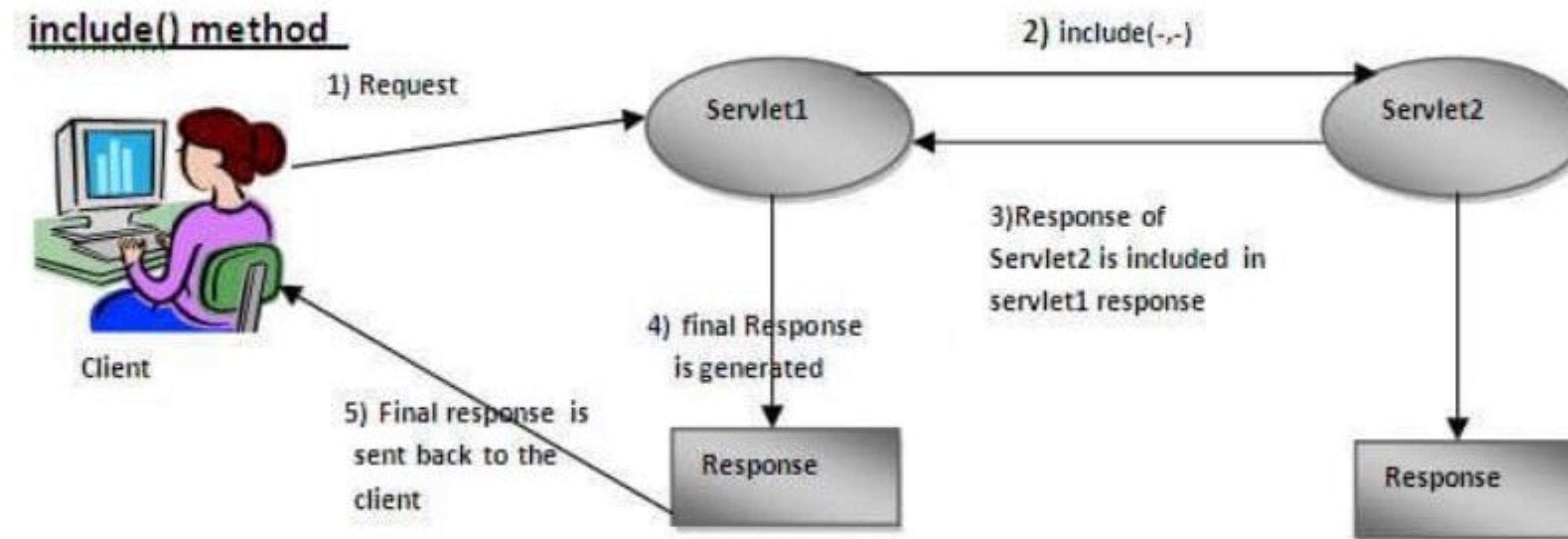
- **public void forward(ServletRequest request, ServletResponse response)**

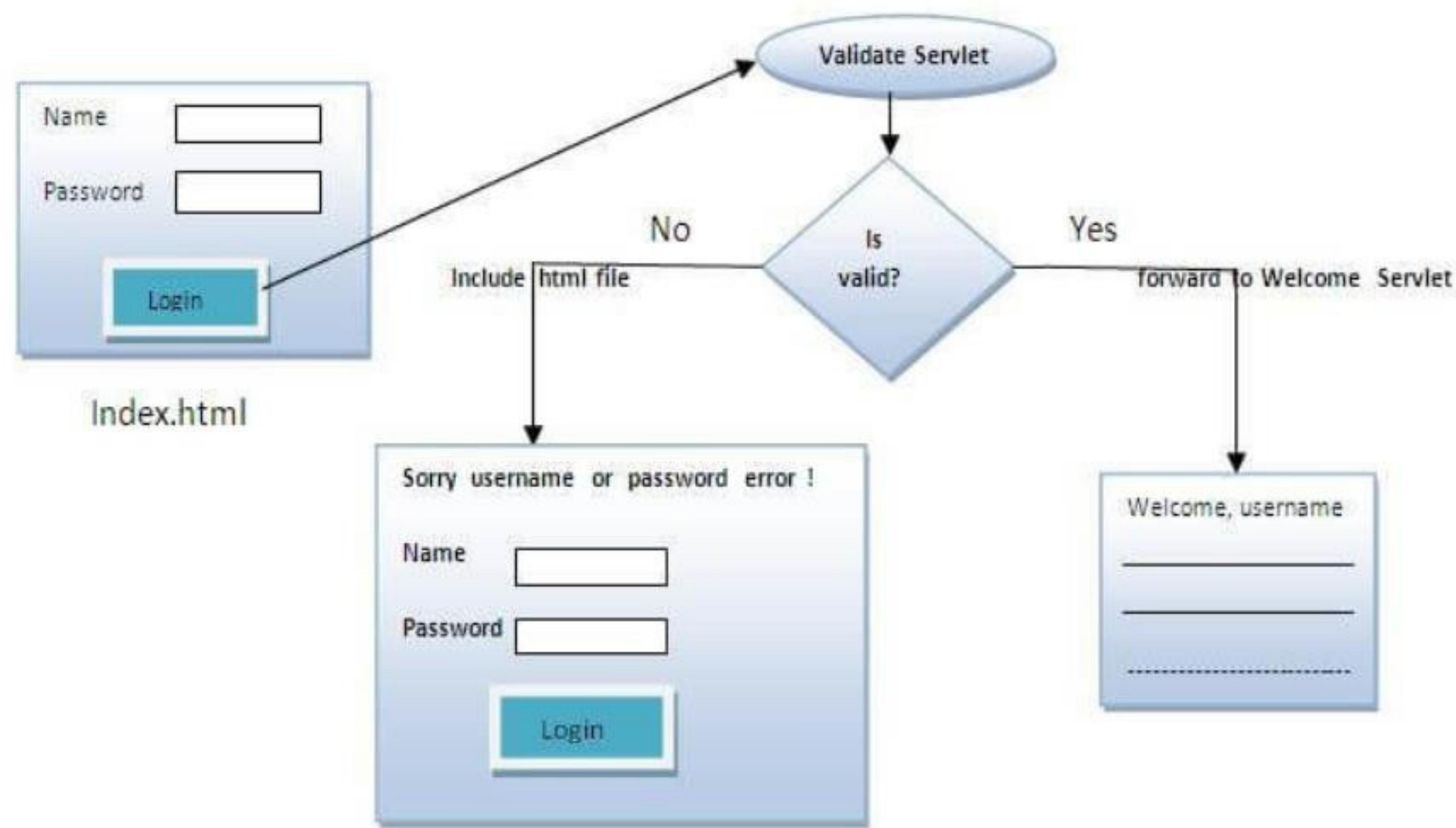
forward() method:



Revisão

- **public void include(ServletRequest request, ServletResponse response)**





Redirecionando Requisições

- A classe `HttpServletResponse` oferece métodos que nos permitem redirecionar o cliente da aplicação;
- Estes métodos são:
 - `sendRedirect(String url);`
 - `sendError(int code);`

Redirecionando Requisições

- O método `sendRedirect`:
 - Permite redirecionar o cliente para outra localização;
 - A URL pode ser para um recurso da mesma aplicação ou para um recurso externo;
 - ✓ Podemos usar URL relativas, caso o redirecionamento seja dentro da aplicação;
 - Após o redirecionamento, o servlet é encerrado;

Redirecionando Requisições

- O método `sendRedirect`:
 - Quando redirecionamos o cliente para outro local, uma nova requisição é gerada;
 - ✓ E, consequentemente, uma nova resposta;
 - Todas as impressões realizadas no buffer de saída da resposta antes do redirecionamento são descartadas;

Redirecionando Requisições

- O método `sendRedirect`:
 - Vamos ver um servlet que recebe uma opção de interesse do usuário e o redireciona para um site de conteúdo da sua preferência;
 - Caso contrário, o servlet redireciona o cliente para outro servlet da aplicação;

Servlet que redireciona o cliente para outros locais

```
@WebServlet("/redirecionamento")
public class RedirecionamentoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

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

        PrintWriter out = response.getWriter();
        //Obtendo o parâmetro informado pelo usuário
        String option = request.getParameter("option");
        //Redirecionando o cliente para sites externos
        if(option.equals("sports"))
            response.sendRedirect("http://www.globoesporte.com");
        if(option.equals("news"))
            response.sendRedirect("http://www.globo.com");
        if(option.equals("movies"))
            response.sendRedirect("http://netflix.com");

        //Redirecionando o cliente para um servlet da aplicação local
        response.sendRedirect("servletLocal");
    }

}
```

Redirecionando Requisições

- O método `sendError`:
 - Este método que nos permite redirecionar o cliente para uma página de erro;
 - O método é interessante pois no permite apresentar, no caso de um erro ou exceção, uma página mais amigável para o usuário;
 - ✓ Diferente da pilha de execução do java;

Redirecionando Requisições

- O método `sendError`:
 - O método `sendError` recebe como entrada um número inteiro, que representa o código do erro gerado pela aplicação;
 - Este código pode ser um código padrão HTTP ou um código definido pela aplicação;

Redirecionando Requisições

- O método `sendError`:
 - As páginas de erro usadas pelos servlets precisam ser definidas no descriptor da aplicação;
 - Cada página de erro pode estar associada a um tipo de código ou a uma classe que representa uma exceção java;
 - Páginas de erro são definidas através do elemento `error-page`;

Redirecionando Requisições

- O método `sendError`:
 - Para associar a página a um código, o componente `errorPage` é composto pelos seguintes sub-elementos:
 - ✓ `error-code`: código do erro associado à pagina;
 - ✓ `description(opcional)`: descrição do erro;
 - ✓ `location`: URL da página para qual o cliente deve ser redirecionado;

Redirecionando Requisições

- O método `sendError`:
 - Associando uma página de erro a um código;

```
<error-page>
    <description>Erro de Login</description>
    <error-code>999</error-code>
    <location>/loginErro.htm</location>
</error-page>
```

Redirecionando Requisições

- O método `sendError(int code)`:
 - Para associar a página a um tipo de exceção, o sub-elemento `exception-type` é usado no lugar do sub-elemento `error-code`;
 - Neste elemento, definimos a classe que representa o tipo de exceção a qual a página é associada;

Redirecionando Requisições

- O método `sendError`:
 - Associando uma página de erro a um tipo específico de exceção;

```
<error-page>
    <description>Erro de indice no vetor</description>
    <exception-type>java.lang.ArrayIndexOutOfBoundsException</exception-type>
    <location>/erro.htm</location>
</error-page>
```

Redirecionando Requisições

- O método `sendError`:
 - Vamos ver um servlet que realiza a autenticação de um usuário;
 - Se as credenciais estiverem corretas, o usuário é redirecionado para a página de acesso ao sistema;
 - Caso contrário, ele é redirecionado para uma página de erro;

Servlet que realiza o redirecionamento de erro

```
@WebServlet("/autentica")
public class AutenticacaoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

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

        //Obtendo os dados para autenticação
        String user = request.getParameter("user");
        String password = request.getParameter("password");

        boolean result = check(user,password);

        if(result)
            response.sendRedirect("acesso.html");
        else response.sendError(999);

    }

    private boolean check(String user, String password) {...5 linhas }

}
```