

# Servlets em detalhes

Ciclo de vida  
Requisição e Resposta  
API



# Servlets

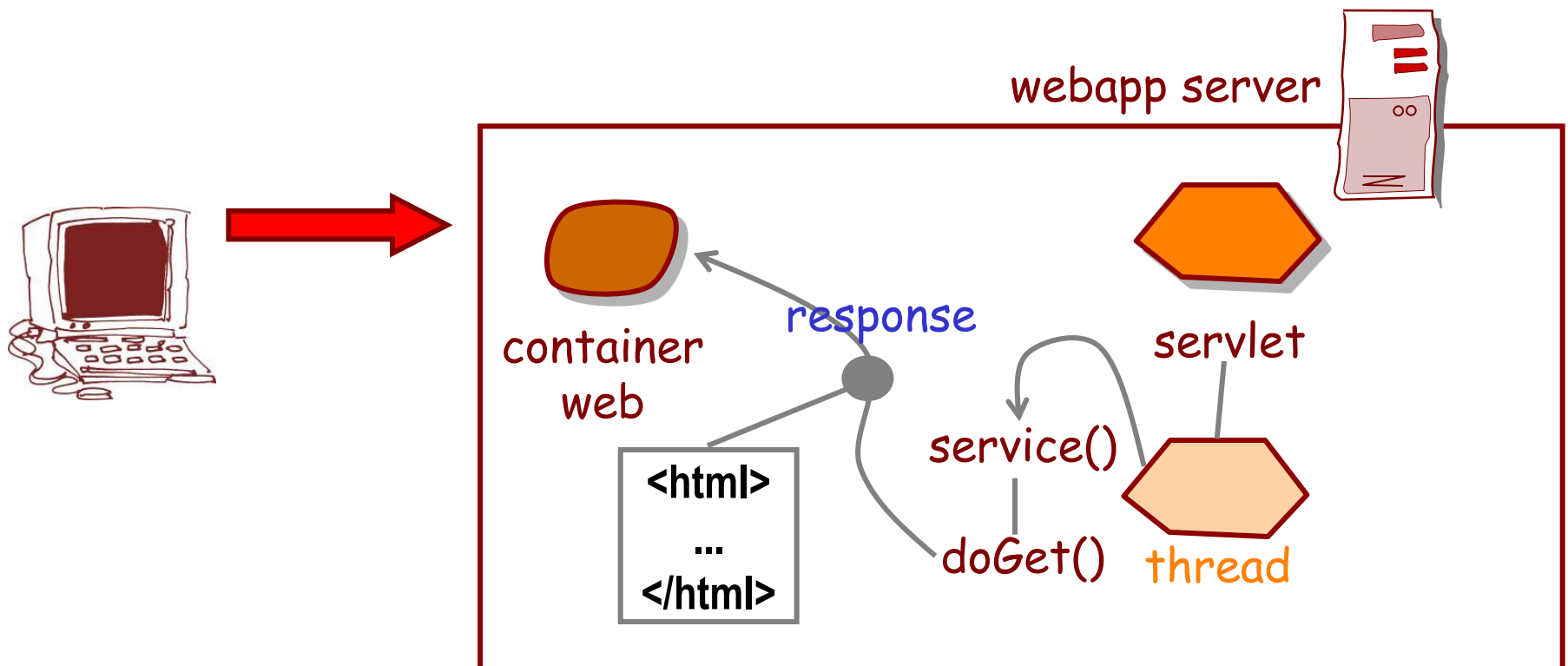
- O objetivo de um servlet é:

Receber uma *requisição* do usuário e  
produzir uma *resposta*

- A resposta pode ser...
  - Produzida pelo próprio servlet
  - Encaminhada para outro componente web (JSP, HTML,...)

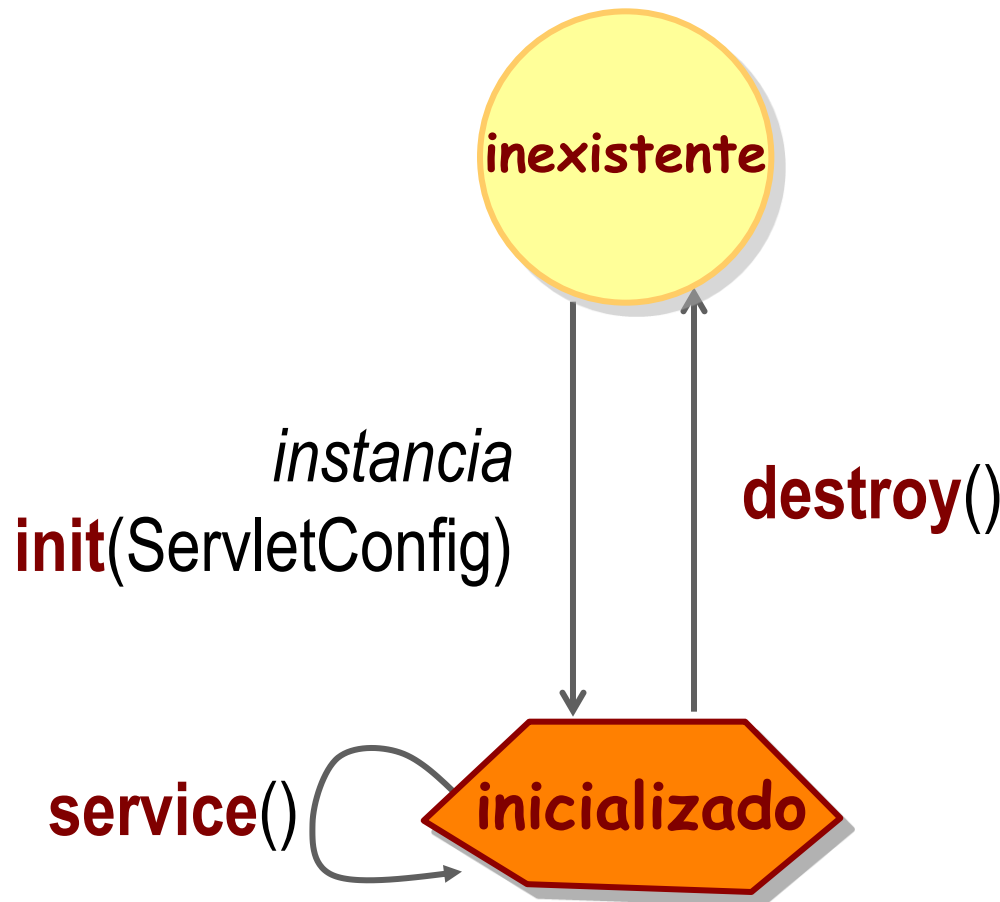
# Ciclo de vida de um servlet

- Atendimento de uma requisição: ciclo de vida já iniciado!

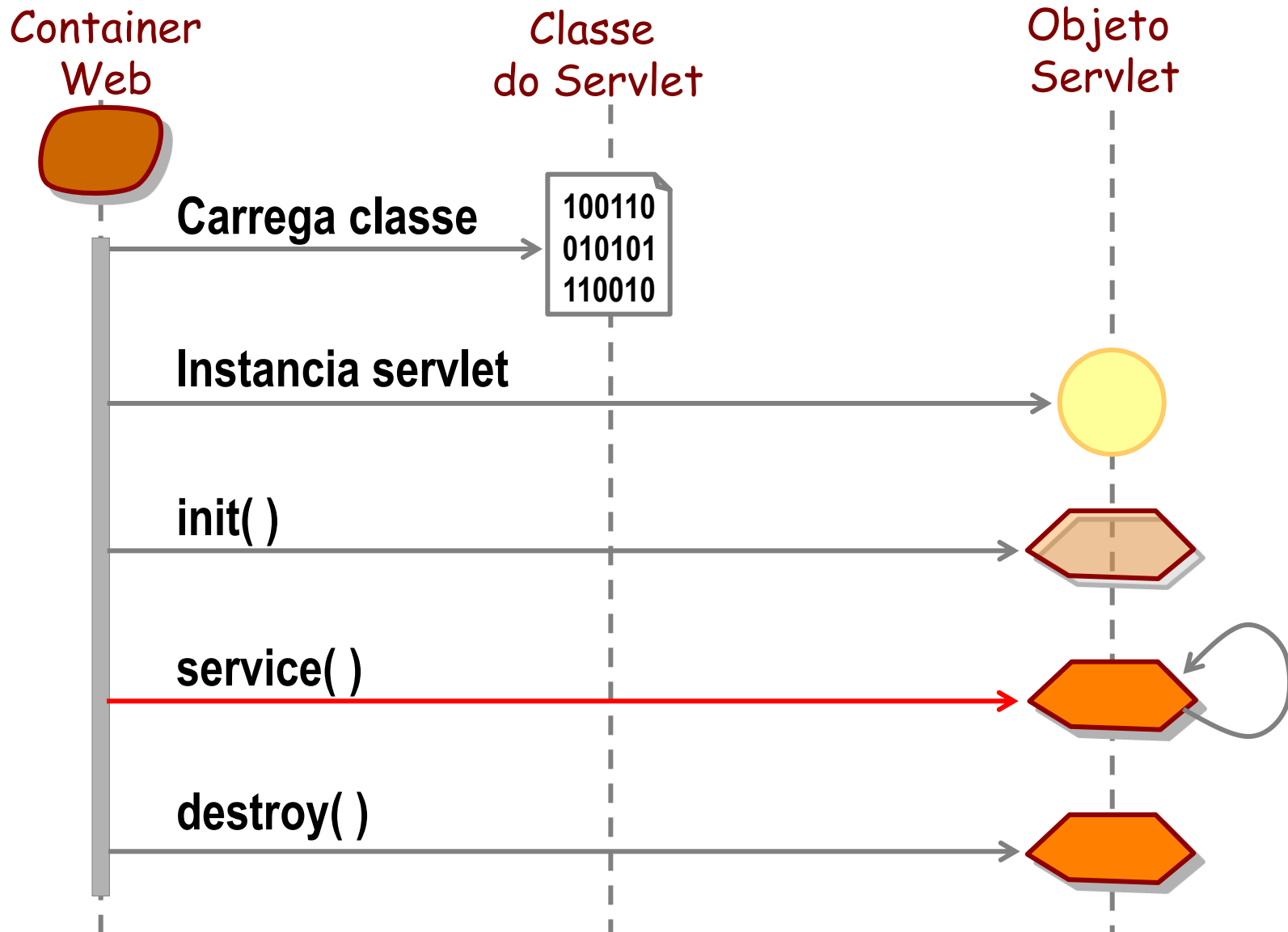


# Ciclo de vida de um servlet

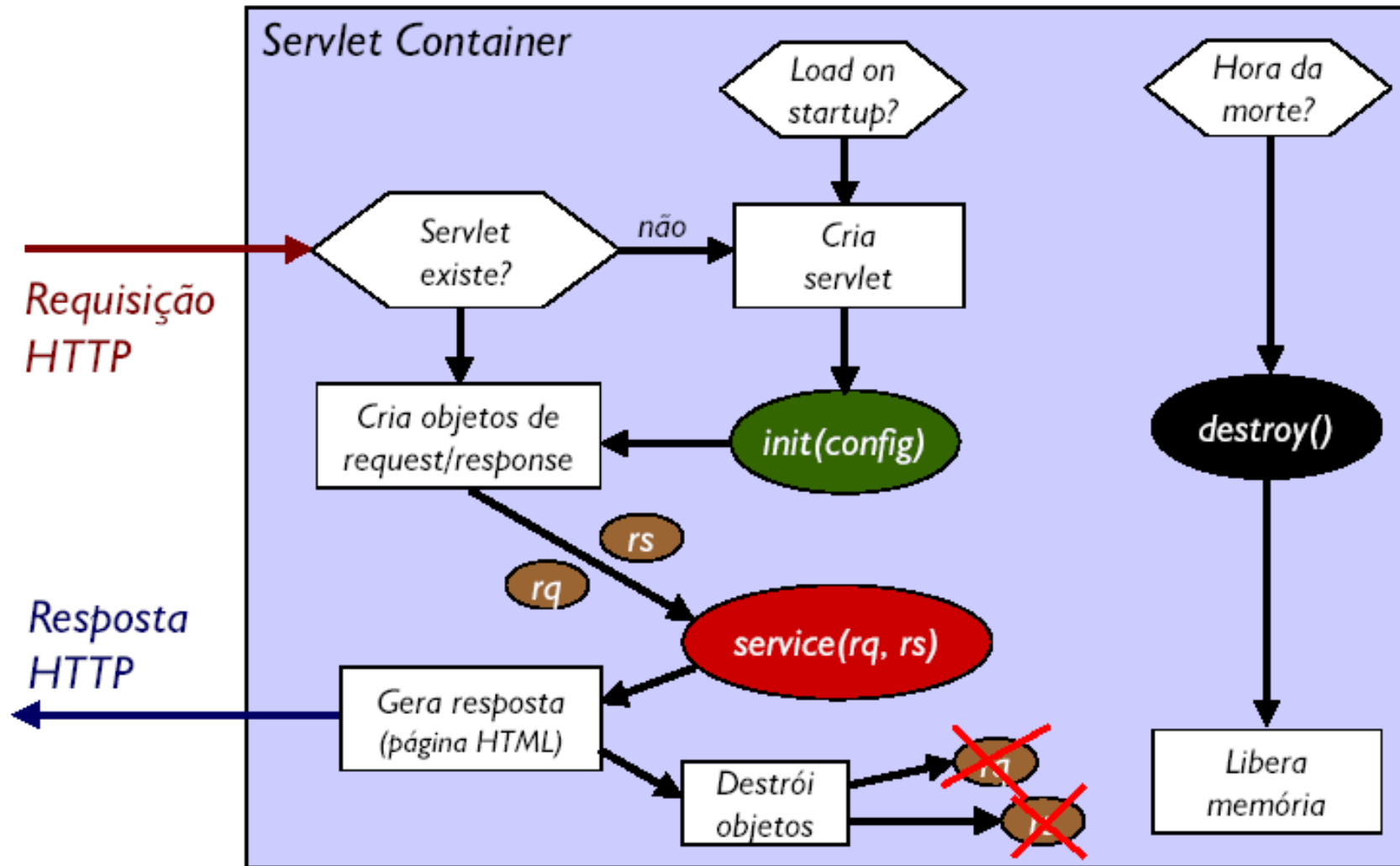
- Dois estados: **inexistente** e **inicializado**



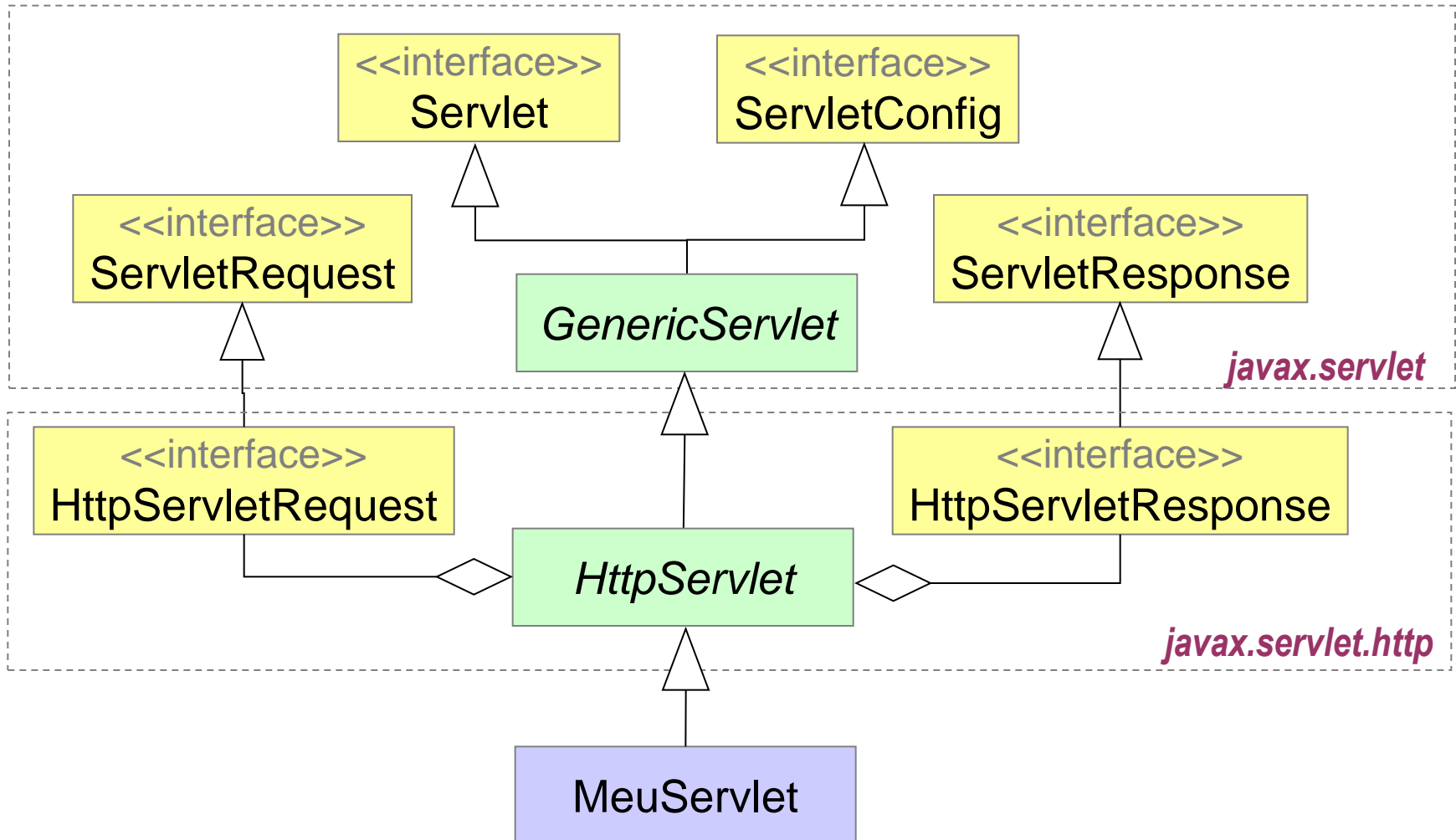
# Ciclo de vida de um servlet



# Ciclo de vida de um servlet



# API de Servlets



# API de Servlets

## ■ Métodos definidos em `javax.servlet.Servlet`

- void **init**(ServletConfig conf)
- void **service**(ServletRequest req, ServletResponse resp)
- void **destroy**()
- ServletConfig **getServletConfig**()
- String **getServletInfo**()

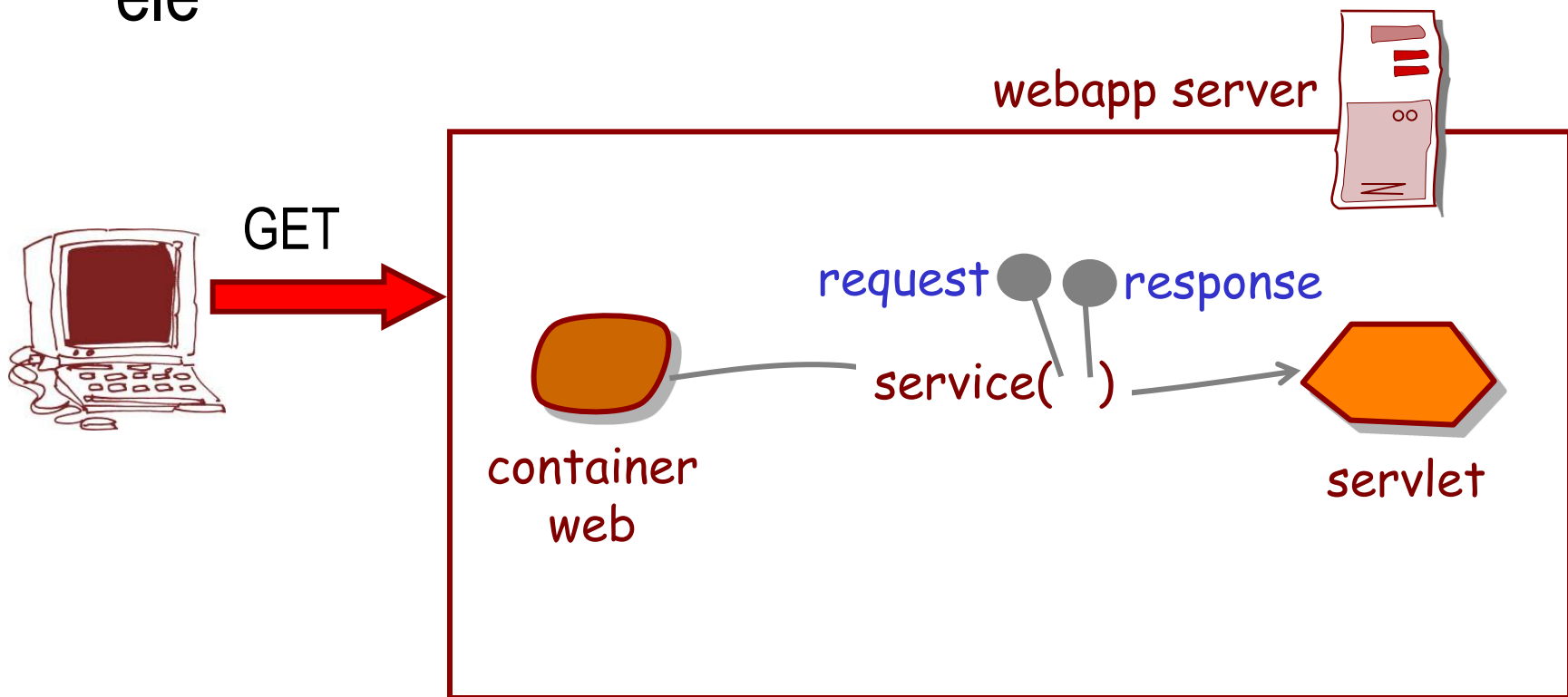
## ■ Alguns métodos de `javax.servlet.http.HttpServlet`

- void **doGet**(HttpServletRequest req, HttpServletResponse resp)
- void **doPost**(HttpServletRequest req, HttpServletResponse resp)
- void **doHead**(HttpServletRequest req, HttpServletResponse resp)
- void **doOptions**(HttpServletRequest req, HttpServletResponse resp)



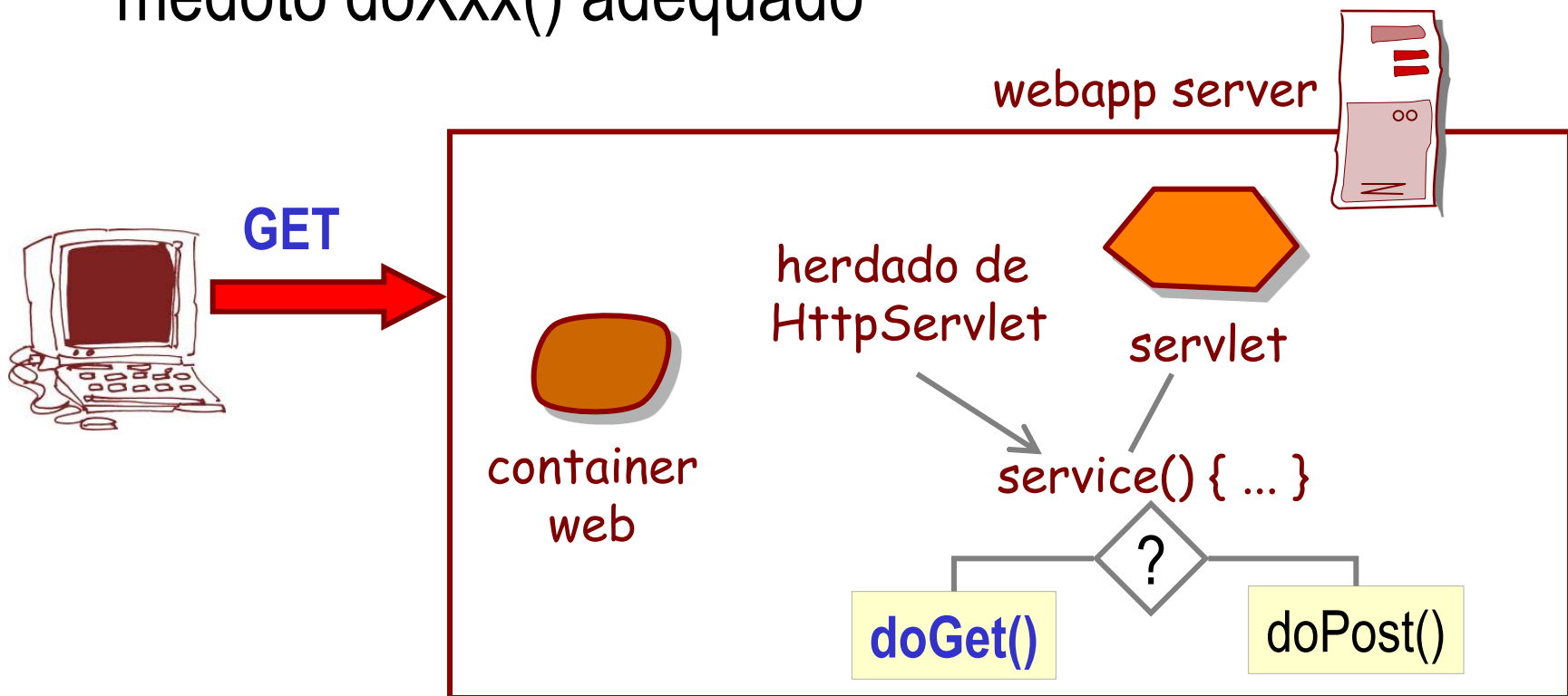
# O serviço de um servlet

- Todo Servlet recebe uma mensagem **service()** do container quando chega uma requisição HTTP para ele



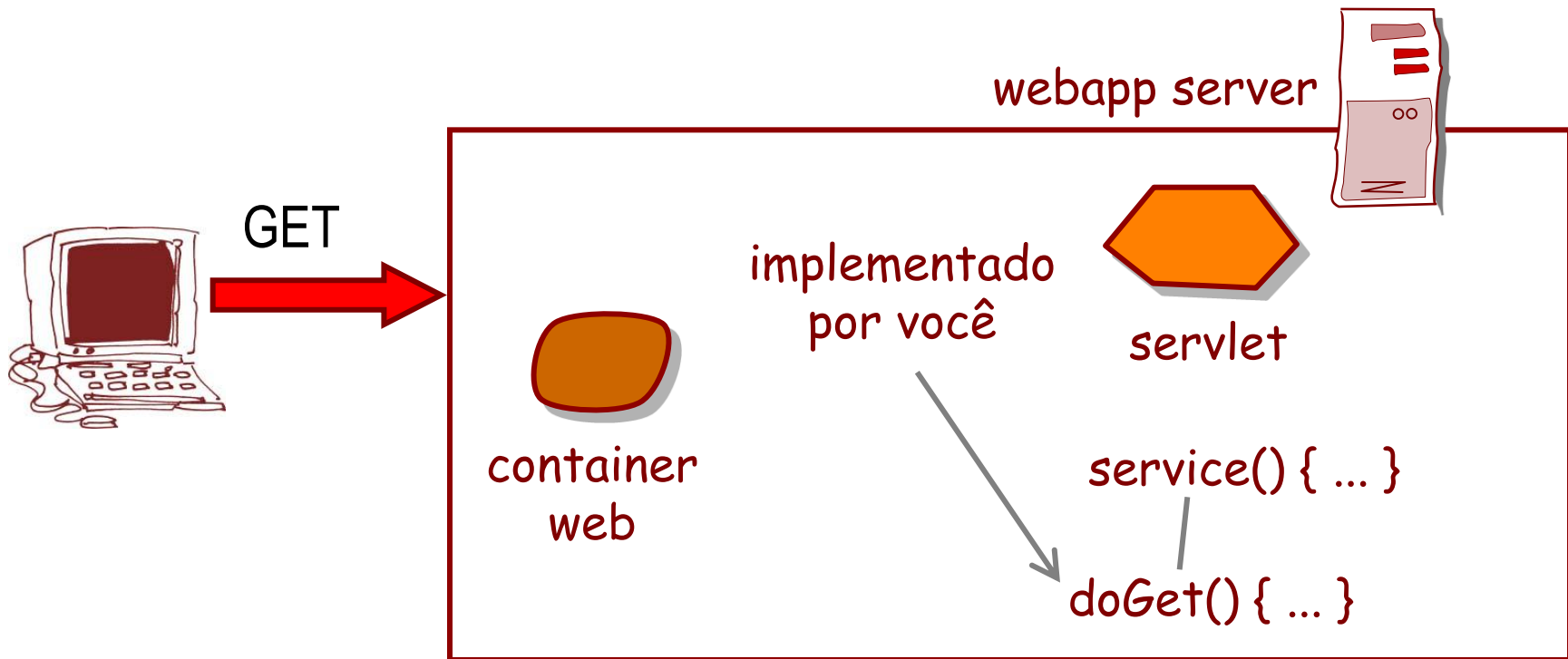
# O serviço de um servlet

- O método `service()` implementado por `HttpServlet` verifica se a requisição foi GET ou POST e chama o método `doXxx()` adequado



# O serviço de um servlet

- Já que o método `service()` herdado de `HttpServlet` decide qual é o método, só precisamos implementar o `doGet` ou `doPost` na nossa classe



# O serviço de um servlet

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

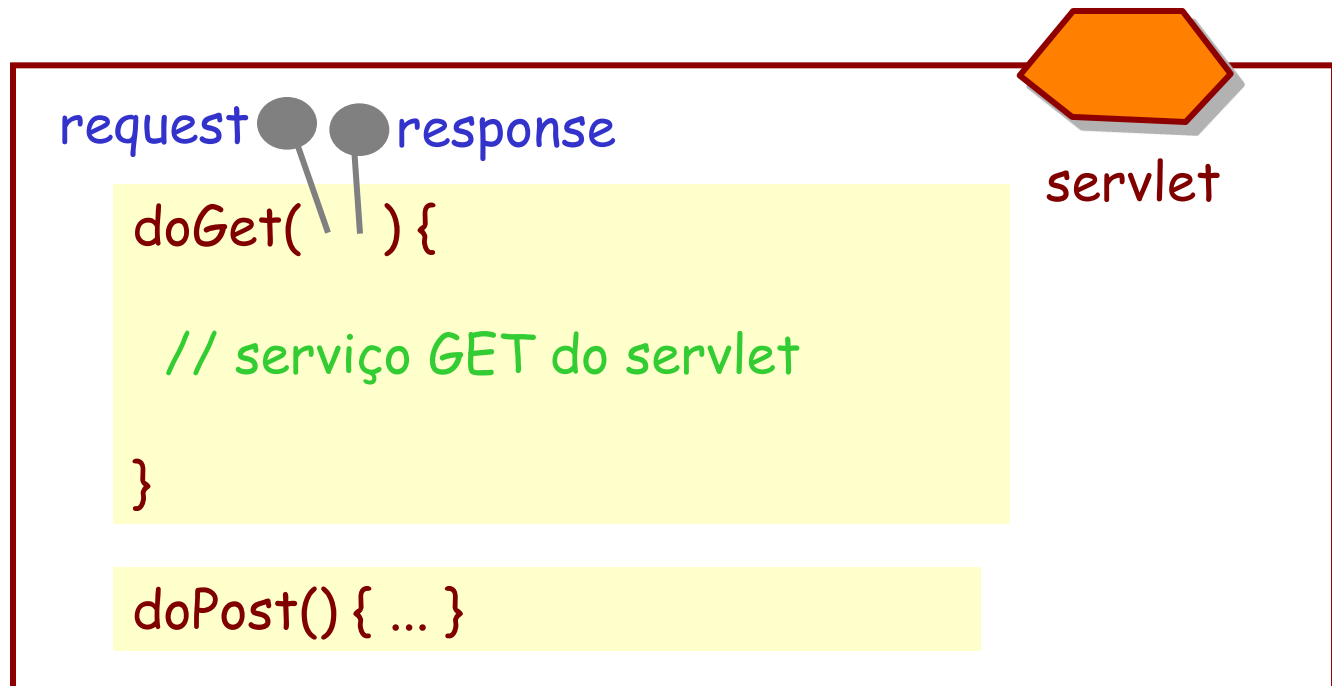
public class AloMundoServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
    }
}
```

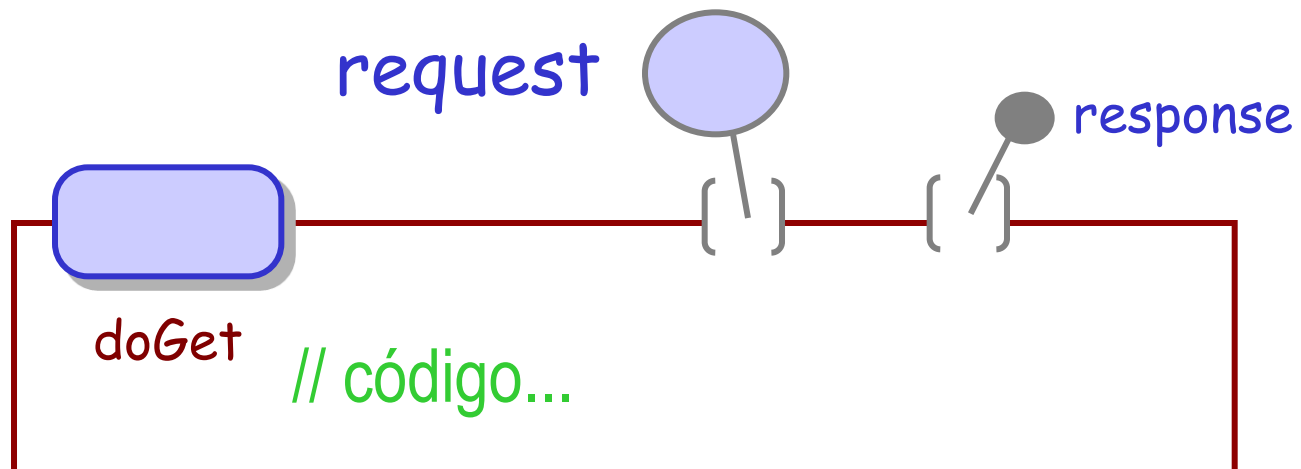
# O serviço de um servlet

- Os métodos doGet() e doPost() recebem sempre dois parâmetros dos tipos HttpServletRequest e HttpServletResponse



# O serviço de um servlet

- O objeto *request* representa a requisição:
  - Valores de cabeçalhos HTTP
  - Parâmetros (dados enviados por URL ou formulário)
  - Cookies previamente gravados no browser
  - Atributos (ler ou associar)



# Interface HttpServletRequest

## ■ API:

- String `getHeader(String cabec)` – obtém 1º valor do cabeçalho
  - Enumeration `getHeaders(String cabec)` – obtém todos os valores
  - Enumeration `getHeaderNames()` – obtém nomes dos cabeçalhos
- 
- String `getParameter(String par)` – obtém valor do parâmetro
  - String[ ] `getParameterValues(String par)` – obtém todos os valores
  - Enumeration `getParameterNames()` – obtém nomes parâmetros
- 
- String `getMethod()` – obtém o método HTTP usado
  - String `getHeader(String nome)` – obtém cabeçalho HTTP
  - String `getQueryString()` – obtém dados codificados

# Interface HttpServletRequest

## ■ API:

- String `getRequestURI()` – obtém URI do servlet
  - String `getRemoteHost()` – obtém nome do host remoto
  - String `getRemoteAddr()` – obtém IP do host remoto
  - String `getProtocol()` – obtém o protocolo utilizado
- 
- Cookie[ ] `getCookies()` – obtém todos os cookies recebidos
  - HttpSession `getSession(boolean cria)` – pega/cria sessão
- 
- Object `getAttribute(String atr)` – obtém valor do atributo
  - void `setAttribute(String nome, Object obj)` – cria/modifica atributo



# O serviço de um servlet

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

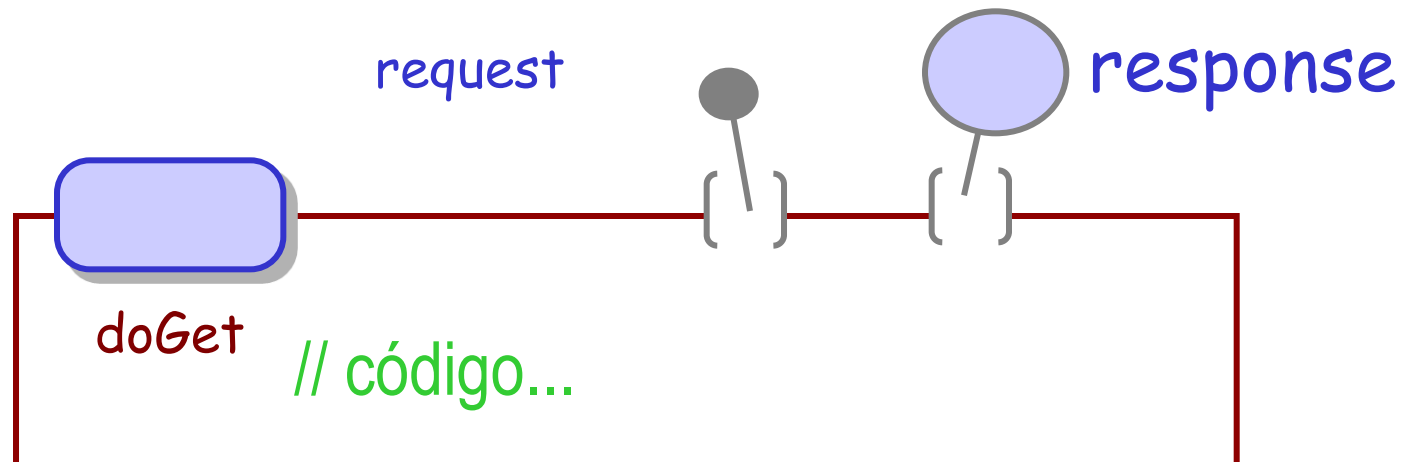
public class AloMundoServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {

        String user = req.getParameter("user");
        String password = req.getParameter("password");
        ...
    }
}
```

# O serviço de um servlet

- O objeto *response* representa a resposta:
  - Valores de cabeçalhos HTTP
  - Conteúdo da resposta
  - Cookies a serem gravados no browser
  - Redirecionamentos



# Interface HttpServletResponse

## ■ API :

- void `setStatus`(int code) – define o código da resposta
  - void `setHeader`(String cabec , String valor) – define valor
  - void `addHeader` (String cabec, String valor) – define +1 valor
- 
- void `setContentType`(String mime) – define tipo MIME
  - void `addCookie` (Cookie c) – define um cookie
  - void `sendError` (int cod, String msg) – devolve uma página de erro
  - void `sendRedirect` (String URI) – redireciona requisição
- 
- PrintWriter `getWriter`() – obtém *stream* de caracteres
  - ServletOutputStream `getOutputStream`() – idem, só que de bytes

# O serviço de um servlet

```
public class AloMundoServlet extends HttpServlet {  
  
    public void doGet(HttpServletRequest req, HttpServletResponse resp)  
        throws IOException, ServletException {  
        String user = req.getParameter("user");  
        String password = req.getParameter("password");  
        if (! valide(user, senha)) {  
            resp.sendError(404, "Usuário ou senha inválido(a).");  
            return;  
        }  
        resp.setContentType("text/html");  
        PrintWriter out = resp.getWriter();  
        out.println("<html><body>Login bem sucedido!</body></html>");  
    }  
}
```

# Interface HttpServletResponse

- Deve-se utilizar os "métodos de cabeçalho" antes de se usar o *stream* para montar a resposta do servlet
  - A estrutura da mensagem HTTP de resposta deve ser obedecida!
- Os métodos `sendError()` e `sendRedirect()` só podem ser usados se o *stream* não estiver comprometido
  - Teste se ele está comprometido com `isCommitted()`
  - Na prática, só use estes métodos quando não tiver usado o *stream*
- Todo servlet que monta a resposta deve usar `setContentType()`
  - O código de status default é o 200 (sucesso no atendimento)

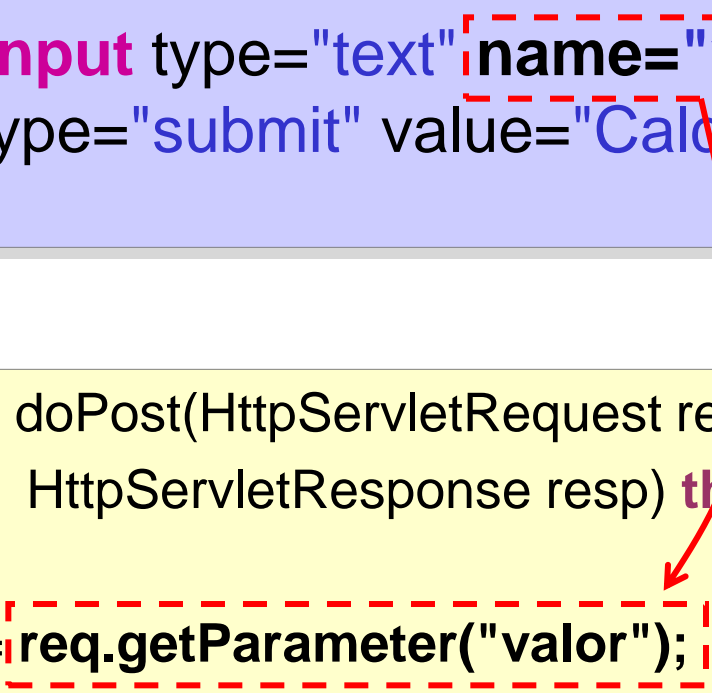
# Parâmetros

## Formulário HTML

```
<form action="ContaCedulas.do" method="POST">  
  Valor: <input type="text" name="valor"><br>  
  <input type="submit" value="Calcula...">  
</form>
```

## Servlet

```
public void doPost(HttpServletRequest req,  
    HttpServletResponse resp) throws IOException {  
    ...  
    String v = req.getParameter("valor");  
    ...  
}
```



# Parâmetros

- O que são?
  - São dados submetidos ao servlet por uma requisição HTTP (GET ou POST).
- Como obtê-los?
  - São obtidos do objeto `HttpServletRequest` via método `getParameter(String)`.
- Qual o tipo do dado retornado?
  - São sempre `String`.
- Posso modificá-los?
  - Não, nem tem por quê.

# Elementos de formulário HTML

## ■ Caixas de entrada de texto

Nome:

Senha:

## ■ Caixa combo

▼

Domingo  
Segunda  
Terça  
Quarta  
Quinta  
Sexta  
Sábado

## ■ Botões de rádio

☐ Masculino

☒ Feminino

## ■ Botões de envio

## ■ Lista (simples e múltipla)

▲

□

□

▼

## ■ Caixas de checagem

☒ Confirmar apagar

☐ Enviar copia

☒ Lembrar senha



# Enviam parâmetros de valores simples

?**sexo=M**&**nome=fred**&**pw=abcdef**&**diasemana=3**

Nome:

Nome: **<input type="text" name="nome">**

Senha:

Senha: **<input type="password" name="pw">**

☒ Masculino **<input type="radio" name="sexo" value="M">** Masculino

☐ Feminino **<input type="radio" name="sexo" value="F">** Feminino

▼

- Domingo
- Segunda
- Terça**
- Quarta
- Quinta
- Sexta
- Sábado

**<select name="diasemana">**

**<option value="1" >**Domingo**</option>**

**<option value="2">**Segunda**</option>**

**<option value="3" selected >**Terça**</option>**

**<option value="4">**Quarta**</option>**

**<option value="5">**Quinta**</option>**

**<option value="6">**Sexta**</option>**

**<option value="7">**Sábado**</option>**

**</select>**

# Enviam parâmetros de valores múltiplos

```
?opcoes=A&opcoes=S&diasemana=2&diasemana=3
```

☒ Confirmar apagar

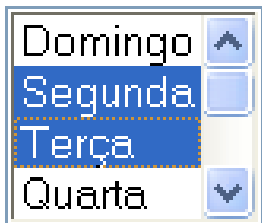
☐ Enviar copia

☒ Lembrar senha

**<input** type="checkbox" name="opcoes" value="A" checked> Confirmar apagar

**<input** type="checkbox" name="opcoes" value="C"> Enviar copia

**<input** type="checkbox" name="opcoes" value="S"> Lembrar senha



**<select** name="diasemana" size="4" multiple>

**<option** value="1">Domingo</option>

**<option** value="2" selected>Segunda</option>

**<option** value="3">Terça</option>

**<option** value="4">Quarta</option>

...

**<option** value="7">Sábado</option>

**</select>**

# Parâmetros

E agora? Como faço para  
pegar os outros valores?



# Parâmetros

- Usa outro método de "pegar" parâmetros:

```
public void doPost(HttpServletRequest req,  
                    HttpServletResponse resp) throws IOException {  
    ...  
    String[ ] v = req.getParameterValues("valor");  
    ...  
}
```

- Retorna um array com os múltiplos valores
- Pode ser usado em parâmetros de valores simples

# Exemplo

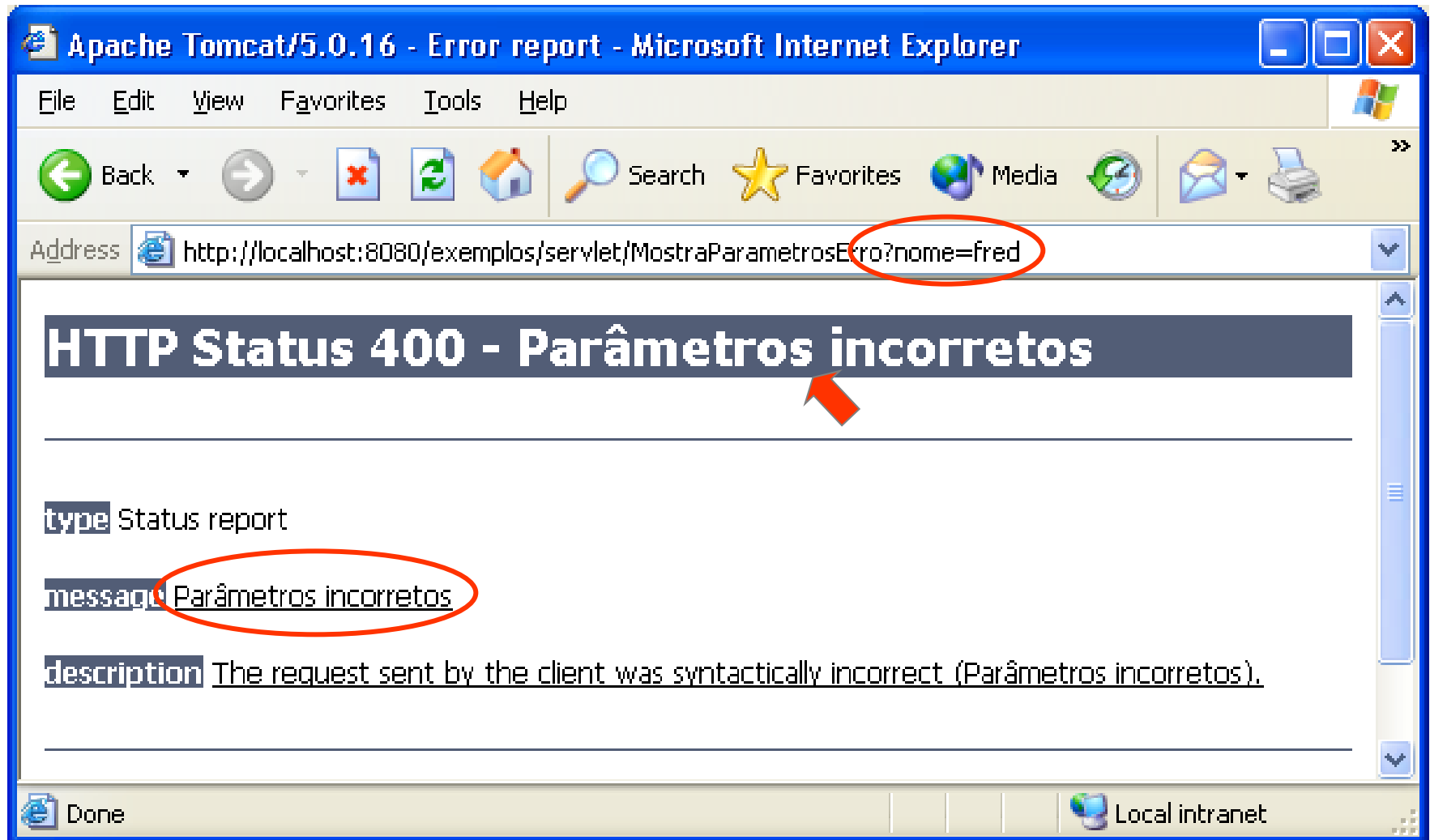
```
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp)
    throws ServletException, IOException {

    String nome = req.getParameter("nome");
    String teste = req.getParameter("teste");

    if (nome == null || teste == null )
        resp.sendError(
            resp.SC_BAD_REQUEST, //Erro 400
            "Parâmetros incorretos");

    else
        this.processRequest(req, resp);
}
```

# Exemplo



# Redirecionamento

## ■ Formas para um componente web chamar outro

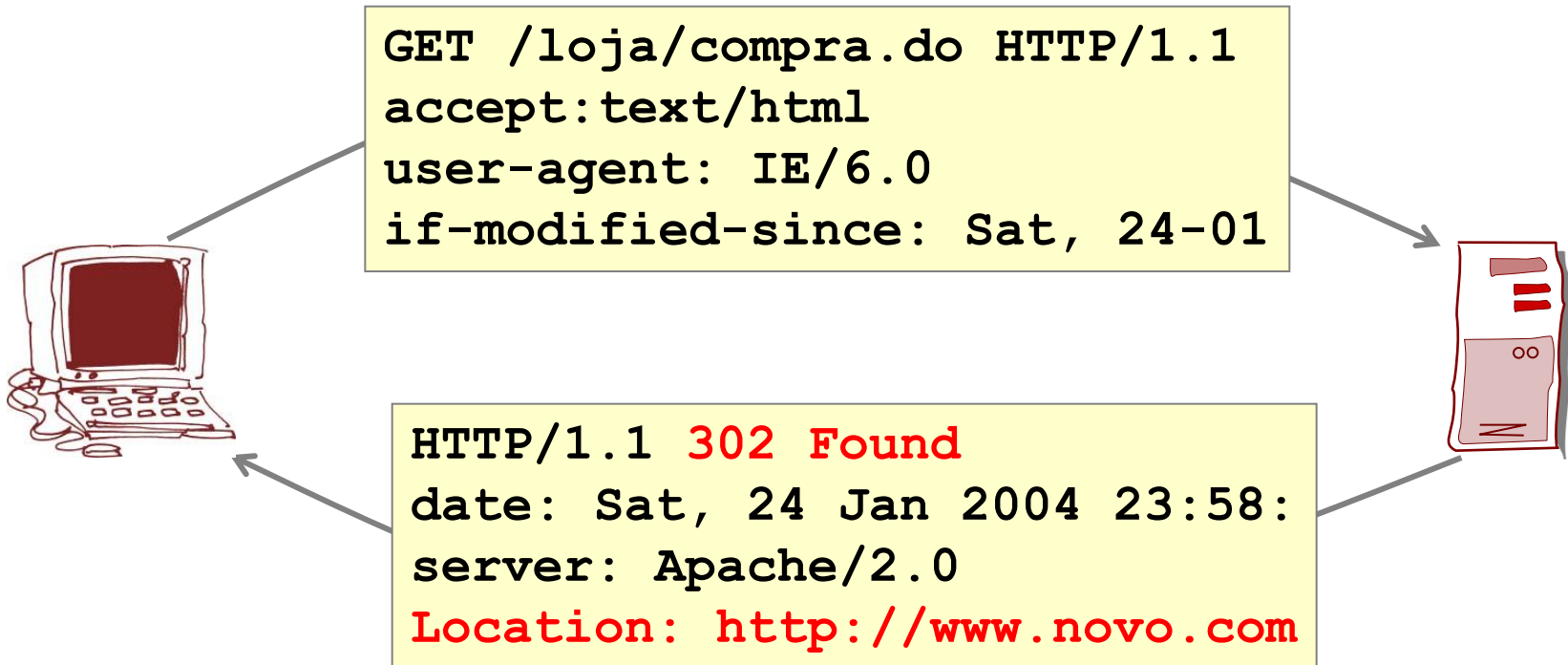
- já vistas {
- Incluir na resposta um link para o outro componente
  - Incluir na resposta um formulário com ação que aponte para o outro componente
  - Redirecionamento HTTP
  - Encaminhamento de requisições

## ■ Diferença entre elas

- As três primeiras utilizam **mais de uma** conexão HTTP
- A última utiliza **a mesma** conexão HTTP

# Redirecionamento

- A requisição e a resposta de um redirecionamento:





# Redirecionamento

## ■ Regra para uso do sendRedirect:

- Só se pode usar sendRedirect() se o objeto **out** não tiver sido comprometido (mesma regra do método sendError)
- Exemplo:

```
if (! login) {  
    String url = "/contexto/login.do";  
    url = response.sendRedirect(url);  
    return;  
}  
out.println("...");  
...
```

# Redirecionamento

- Outra forma mais complicada de redirecionar:

```
if (! login) {  
    String url = "/contexto/login.html";  
    url = response.encodeRedirectURL(url);  
    response.setStatus(302);  
    response.setHeader("Location", url);  
    return;  
}  
out.println("...");  
...
```

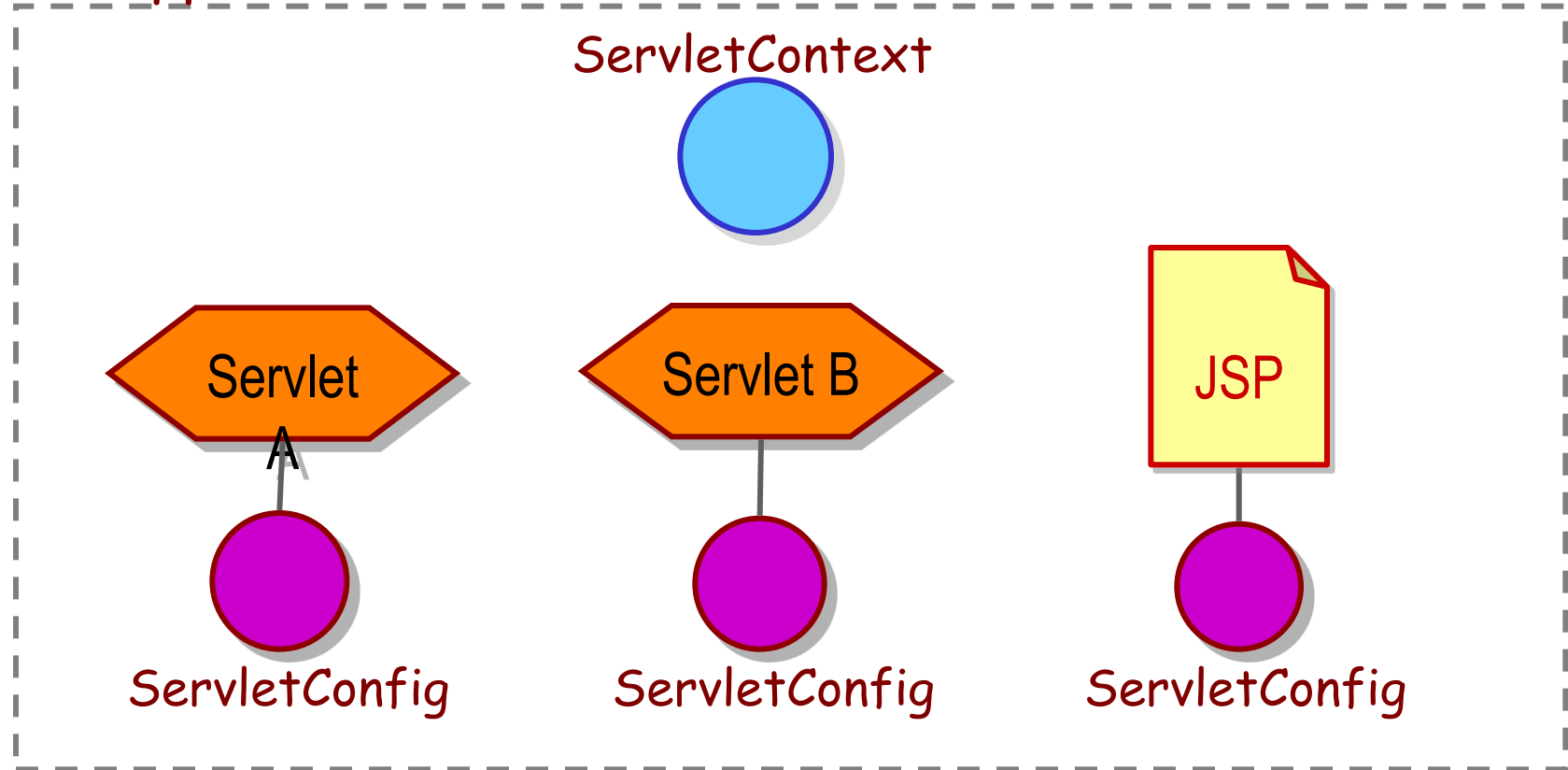
→ response.sendRedirect(url);

# ServletContext e ServletConfig

- Toda **aplicação web** possui **um único** objeto **ServletContext** que representa o contexto
  - Criado quando a aplicação é iniciada pelo container
  - Obtido com ***this.getServletContext()***
- Todo **servlet** possui seu próprio objeto **ServletConfig**
  - Criado quando o servlet é iniciado pelo container com o método *init(ServletConfig c)*
  - Obtido com ***this.getServletConfig()***

# ServletContext e ServletConfig

webapp



Não se engane com o nome,  
o **ServletContext** é UM POR APLICAÇÃO!

# Interface ServletContext

<<interface>>

## **ServletContext**

***getInitParameter(String)***

***getInitParameterNames()***

***getAttribute(String)***

***getAttributeNames()***

***setAttribute(String, Object)***

***removeAttribute(String)***

-----  
***getMajorVersion()***

***getServerInfo()***


-----  
***getRealPath()***

***getResourceAsStream(String)***

***getRequestDispatcher(String)***

...

Métodos para ler  
parâmetros de  
contexto do web.xml e  
manipular atributos



Facilita a portabilidade  
de arquivos dentro da  
aplicação web




Transfere requisições



# Interface ServletConfig

```
<<interface>>  
ServletConfig  
getInitParameter(String)  
getInitParameterNames()  
  
getServletContext()  
getServletName()
```

Métodos para ler  
parâmetros de servlet  
do web.xml

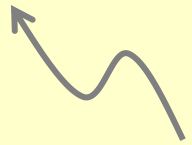


Devolve o  
ServletContext da  
aplicação



# Exemplo de uso

```
public class AloMundoServlet extends HttpServlet {  
    private static Properties passwd;  
  
    public void init() throws ServletException {  
        passwd = new Properties();  
        try {  
            passwd.load(  
                this.getServletConfig()  
                    .getServletContext()  
                    .getResourceAsStream("/WEB-INF/user.properties"));  
        } catch (IOException e) {  
            passwd = null;  
        }  
    }  
}
```



continua...

Evita o uso de paths  
completos dentro do  
código (comece com /)

# Exemplo de uso

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException {
    // pega os parâmetros fornecidos pelo formulário
    String user = req.getParameter("user");
    String password = req.getParameter("password");
    // verifica se o usuário é valido
    if (passwd.contains(user))
        // verifica se a senha confere
        if (password.equals(passwd.get(user)))
            ...
    }
}
```





Como é que o container sabe que é para executar um servlet? Como ele descobre, qual é o servlet a partir da URL?

# Mapeando uma URL num servlet

- Todo servlet tem 3 nomes:
  - Um **nome URL**, que é o que o usuário e o programador HTML conhece
  - Um **nome interno** usado no arquivo de mapeamento XML
  - O **nome da classe** (precedido do pacote, se for o caso)
- O que o container recebe é o path para o recurso (servlet) e tem que se virar com ele

```
GET /teste/show.do HTTP/1.1
```

# Mapeando uma URL num servlet

- Deve-se registrar o servlet no ***descriptor de implantação*** (DD) da aplicação web
  - **<servlet>**
    - mapeia o nome interno no nome completo da classe (do pacote em diante)
  - **<servlet-mapping>**
    - mapeia o nome interno no nome URL
- O DD é um arquivo chamado **web.xml** que fica na pasta **WEB-INF/** da aplicação

# Mapeando uma URL num servlet

## ■ Exemplo de arquivo web.xml

```
<webapp>
  <servlet>
    <servlet-name>Alo</servlet-name>
    <servlet-class>br.fred.AloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Alo</servlet-name>
    <url-mapping>/show.do</url-mapping>
  </servlet-mapping>
</web-app>
```

Nome interno do servlet

Nome qualificado da classe

Nome URL do servlet



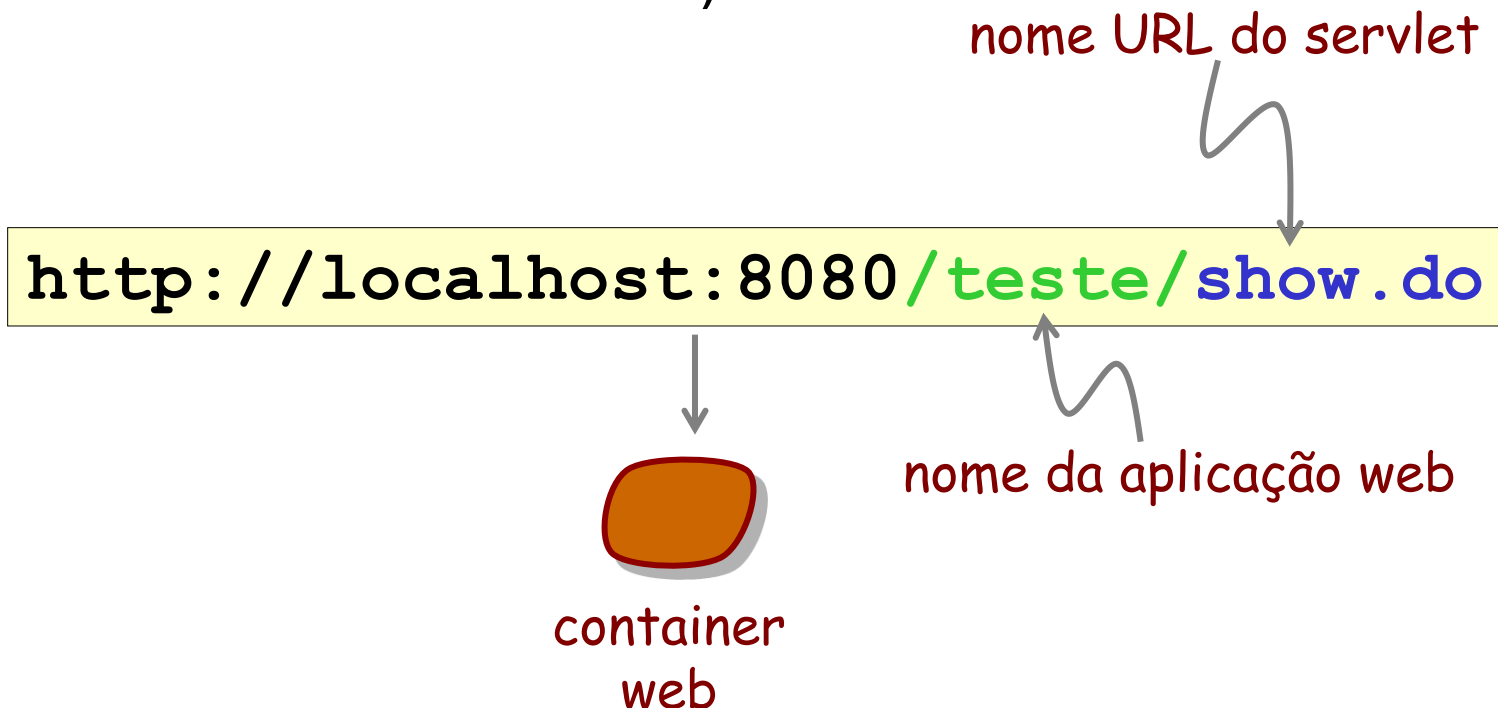
Neste arquivo você só mapeou o **nome da classe** no nome URL. Como o container encontra o arquivo da classe se o caminho não foi indicado?



O container sabe onde encontrar as aplicações web na sua estrutura de diretórios. O Tomcat mantém as aplicações web numa pasta chamada **webapps**. Como toda aplicação web tem a mesma estrutura, fica fácil achar o arquivo .class

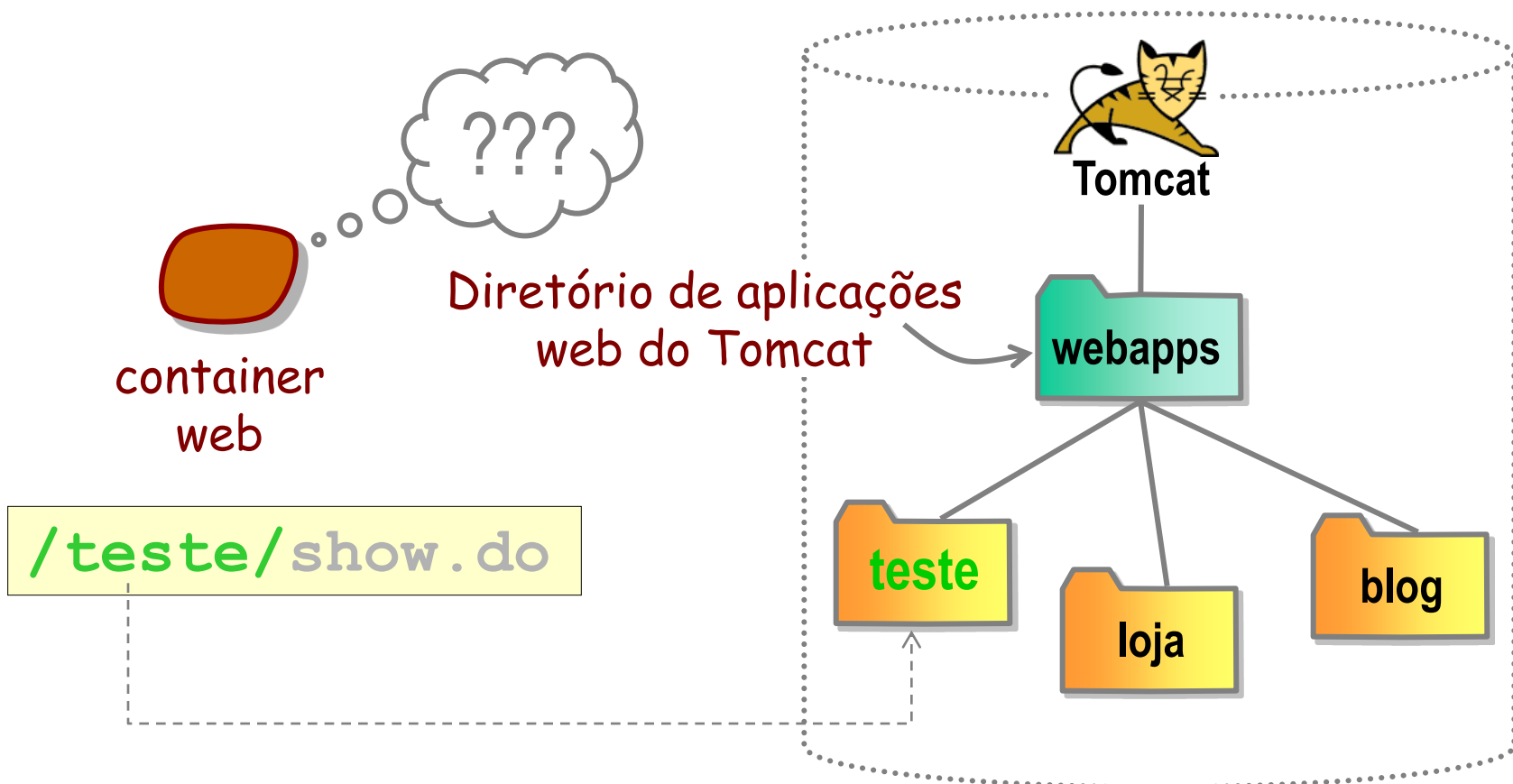
# Mapeando e executando um servlet

- O container recebe a requisição e o path do recurso (que neste caso é o nome da aplicação seguida do nome URL do servlet)



# Mapeando e executando um servlet

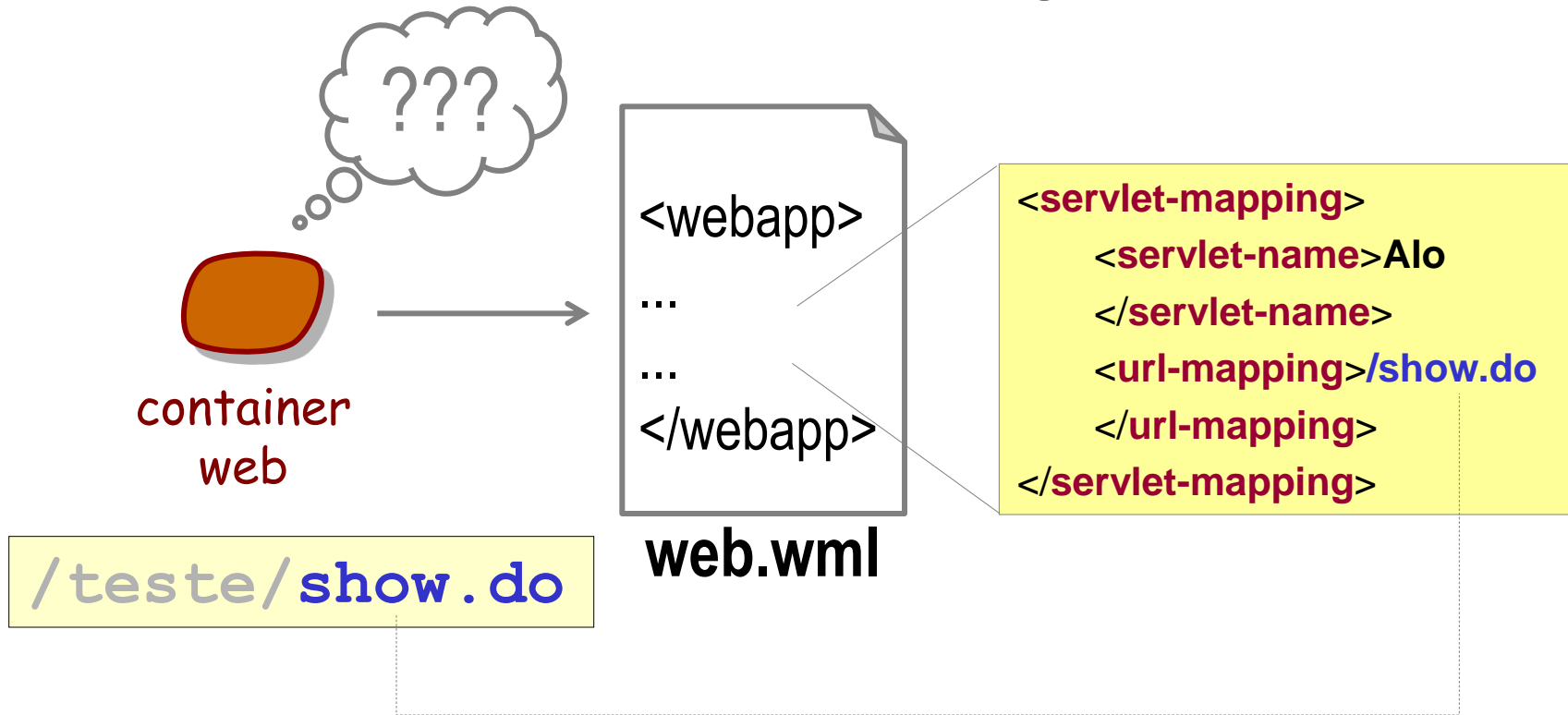
- O container procura no diretório de aplicações web por uma aplicação chamada **teste**





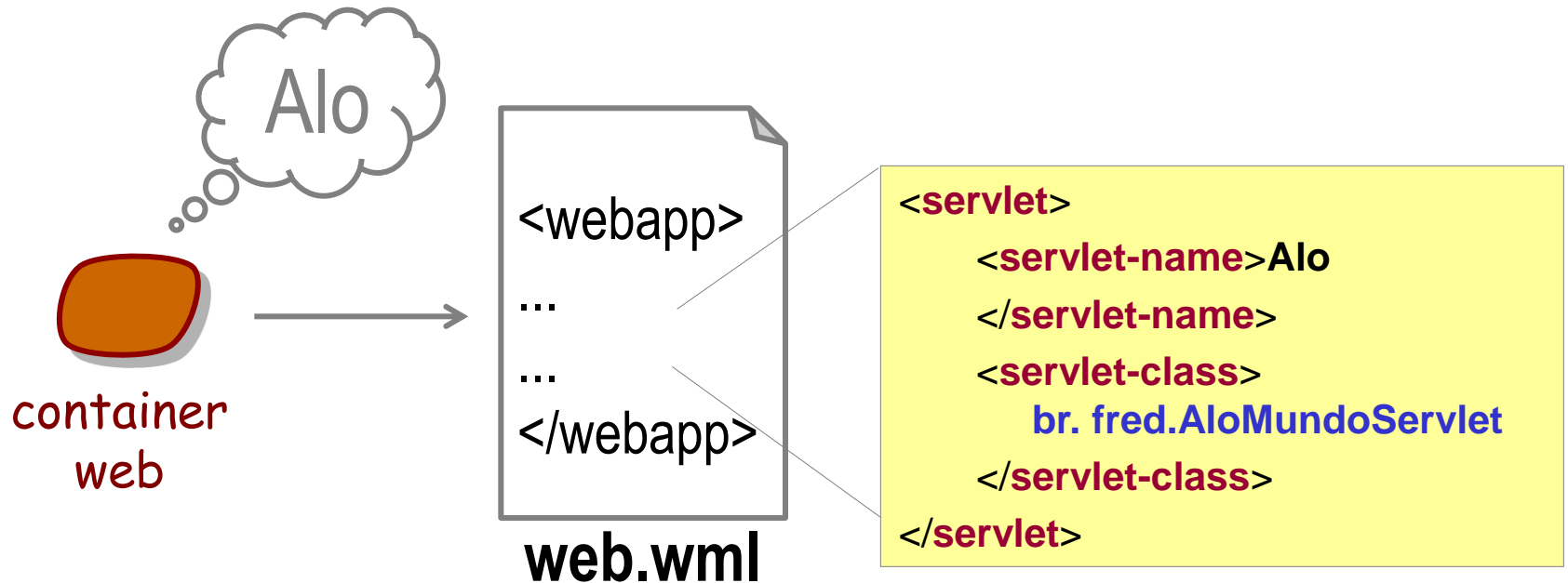
# Mapeando e executando um servlet

- Utilizando o arquivo web.xml da aplicação **teste**, ele procura por um **<servlet-mapping>** para **/show.do**



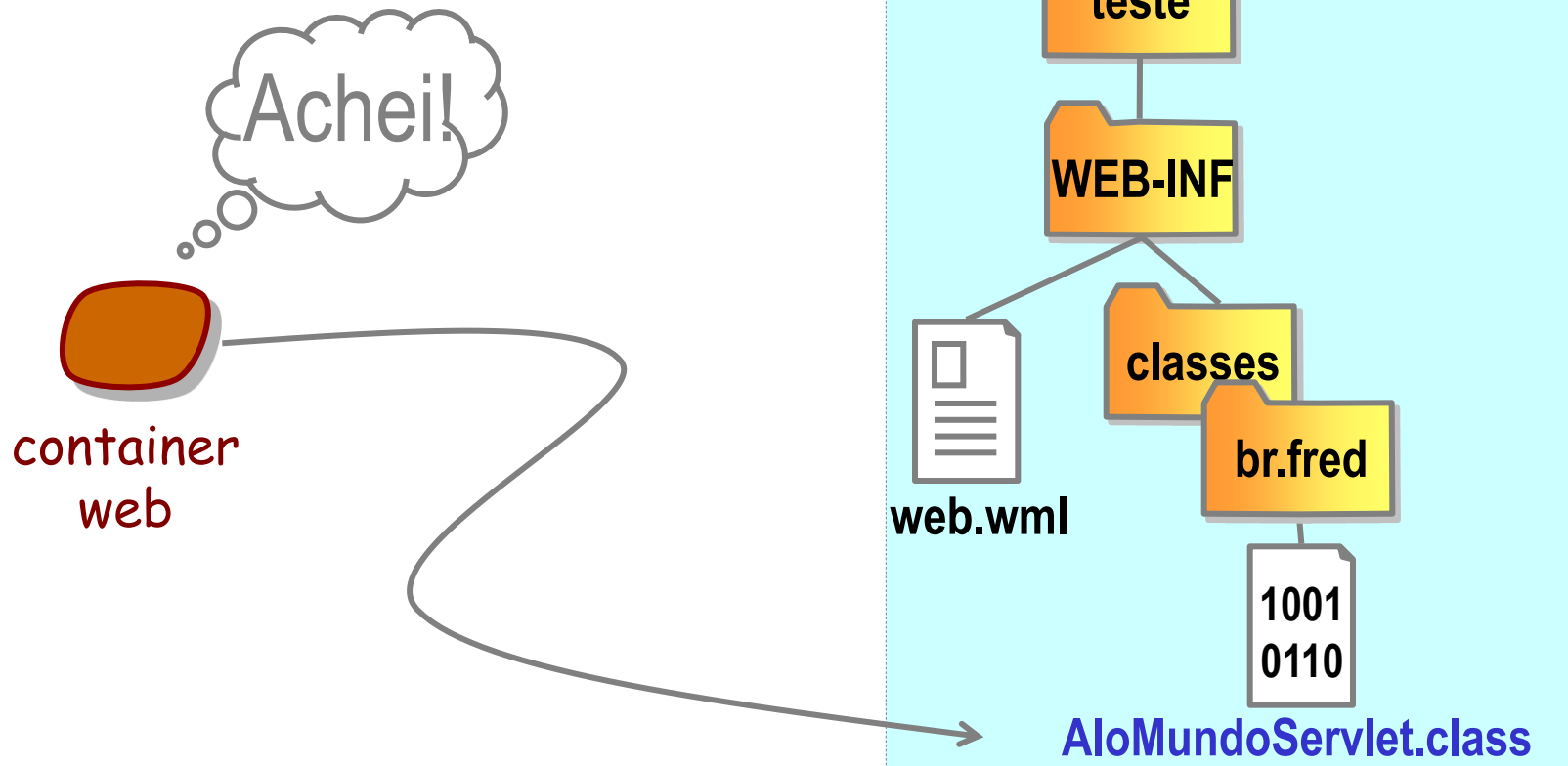
# Mapeando e executando um servlet

- Com o nome interno do servlet (**Alo**), ele pega o nome da classe lá no marcador **<servlet>**



# Mapeando e executando um servlet

- Por fim, procura pelo arquivo AloMundoServlet.class dentro do diretório **classes** na aplicação web:



# Servlets 3.0

## ■ Anotação @WebServlet

- Deve ter um parâmetro *value* ou *urlPattern* definido
- A classe deve herdar de HttpServlet

```
@WebServlet("/foo")  
public class CalculatorServlet extends HttpServlet{  
    //...  
}
```

```
<servlet>  
    <servlet-name>CalculatorServlet</servlet-name>  
    <servlet-class>CalculatorServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>CalculatorServlet</servlet-name>  
    <url-mapping>/foo</url-mapping>  
</servlet-mapping>
```

# Servlets 3.0

```
@WebServlet(name="MyServlet", urlPatterns={"/foo", "/bar"})  
public class CalculatorServlet extends HttpServlet {  
    ...  
}
```

```
<servlet>  
    <servlet-name>MyServlet</servlet-name>  
    <servlet-class>CalculatorServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>MyServlet</servlet-name>  
    <url-mapping>/foo</url-mapping>  
</servlet-mapping>  
<servlet-mapping>  
    <servlet-name>MyServlet</servlet-name>  
    <url-mapping>/bar</url-mapping>  
</servlet-mapping>
```

# Referências

- Apostila do curso Caelum FJ-21:Java para Desenvolvimento Web
  - Disponível em <http://www.caelum.com.br/apostila-java-web/>
- Especificações oficiais de Servlet 3.0:
  - <http://download.oracle.com/otndocs/jcp/servlet-3.0-fr-eval-oth-JSpec/>
- A Servlet and JSP Tutorial
  - <http://aplacenmp.apl.jhu.edu/~hall/java/Servlet-Tutorial/>