

# JSP sem scripts

JSP-Expression Language  
Objetos implícitos



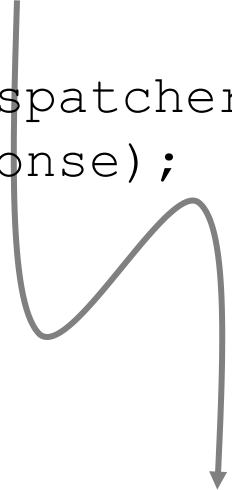
# Ações padrões para JavaBeans



# Transferindo dados para um JSP

## ■ Código do servlet:

```
public void doPost(request, response) {  
    String nome = request.getParameter("uname");  
    request.setAttribute("nome", nome);  
    RequestDispatcher v =  
        request.getRequestDispatcher("/mostra.jsp");  
    v.forward(request, response);  
}
```



## ■ Código do JSP:

```
<html><body>  
Nome: <%=request.getAttribute("nome") %>  
</body></html>
```

# Transferindo dados para um JSP

## ■ Código do servlet:

```
public void doPost(request, response) {  
    Aluno a = new Aluno();  
    a.setNome("Fred");  
    request.setAttribute("aluno", a);  
    RequestDispatcher v =  
        request.getRequestDispatcher("/mostra.jsp");  
    v.forward(request, response);  
}
```

## ■ Código do JSP:

```
<html><body>  
  <% Aluno a = (Aluno) request.getAttribute("aluno"); %>  
  Nome: <%=a.getNome() %>  
</body></html>
```

# Expression Language

A Expression Language  
resolve este (e outros)  
problemas e  
desburocratiza o  
código JSP!

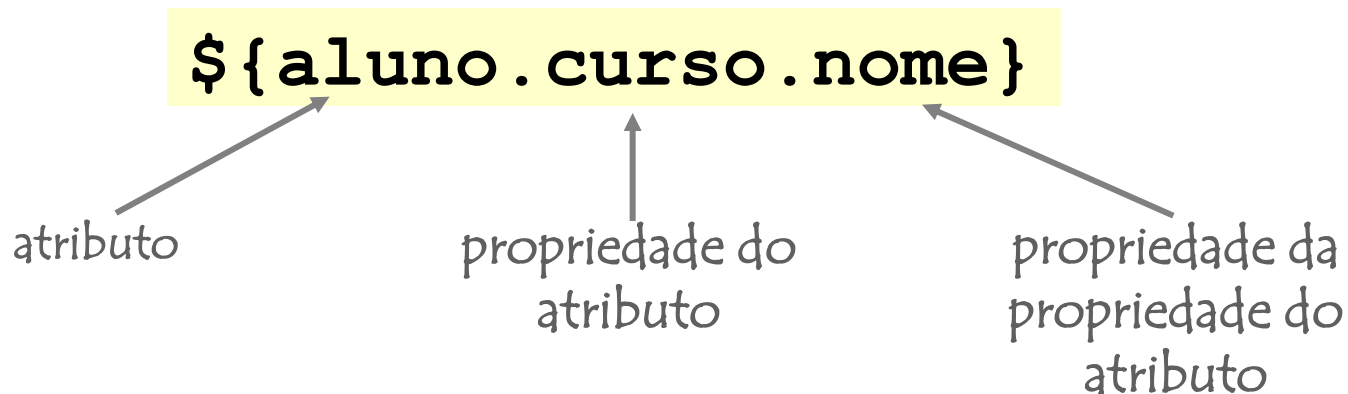


# Expression Language

## ■ Código sem EL:

```
<%= ((Aluno)  
request.getAttribute("aluno")).getCurso().getNome()  
%>
```

## ■ Código com EL:



# Expression Language

## ■ A sintaxe da EL

- Não é Java!
- Forma de acessar objetos Java de outra maneira
- Sempre envoltas entre **`${ }`**
- O primeiro elemento de uma EL é sempre um atributo ou um objeto implícito (grave isso!)

**`${primeiro.segundo}`**

↑  
um atributo ou um objeto implícito

# Expression Language

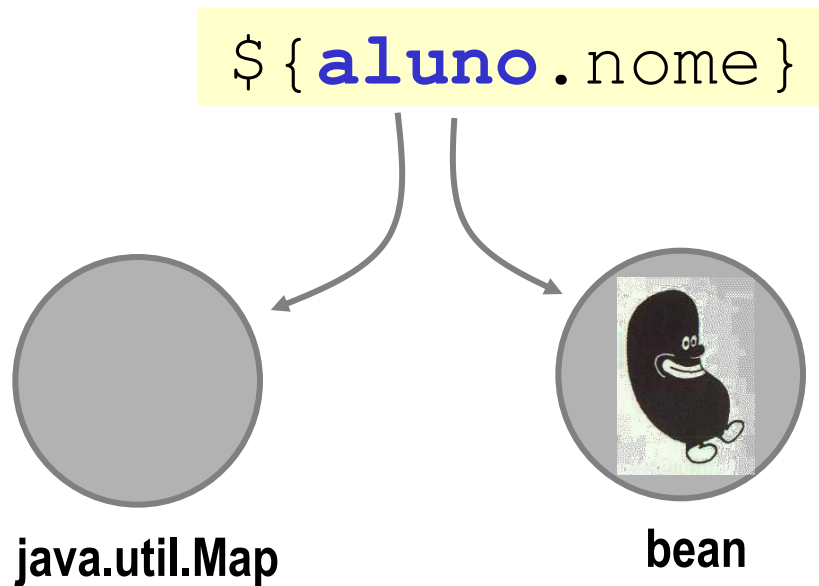
- Se for um **atributo**...
    - Pode estar num dos escopos: **page**, **request**, **session** e **application**
  - Se for um **objeto implícito**...

■ pageScope	■ headerValues
■ requestScope	■ cookie
■ applicationScope	■ initParam
■ param	■ pageContext
■ paramValues	
■ header	
- ↑  
*Apenas este também é um objeto em JSP puro, os outros são **Maps**!*



# Expression Language

- Se a expressão é formada por uma variável seguida de um ponto, a variável só pode ser um mapa ou um bean

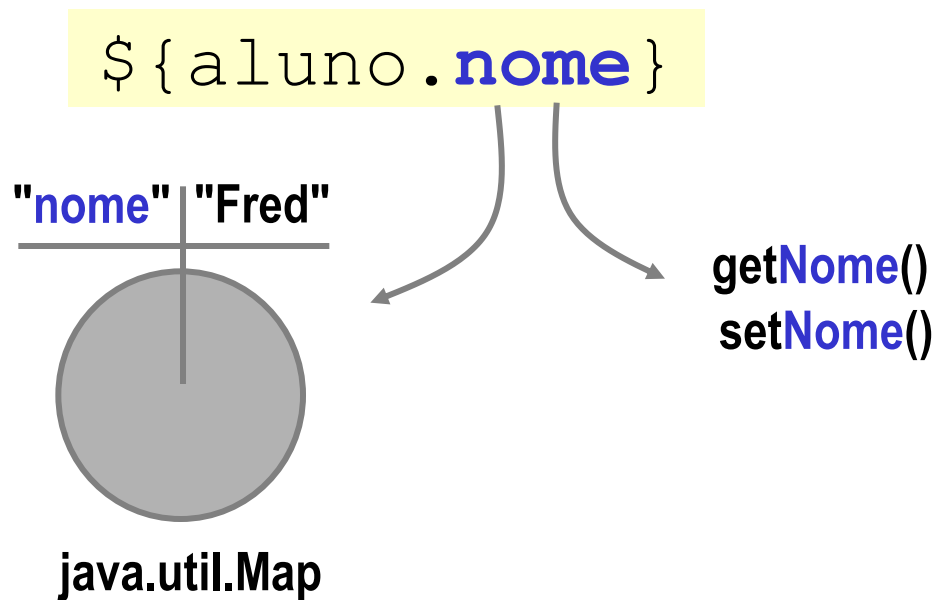


**Grave!**

Na notação com ponto, a variável da esquerda só pode ser um mapa ou um bean

# Expression Language

- O que vier à direita do ponto deve ser uma propriedade de um bean ou uma chave de mapa



# Expression Language

- Em outras palavras...

- Isto:

```
${aluno.nome}
```

- Pode significar isto:

```
<%= aluno.getNome() %>
```

aluno é um bean



- Ou isto:

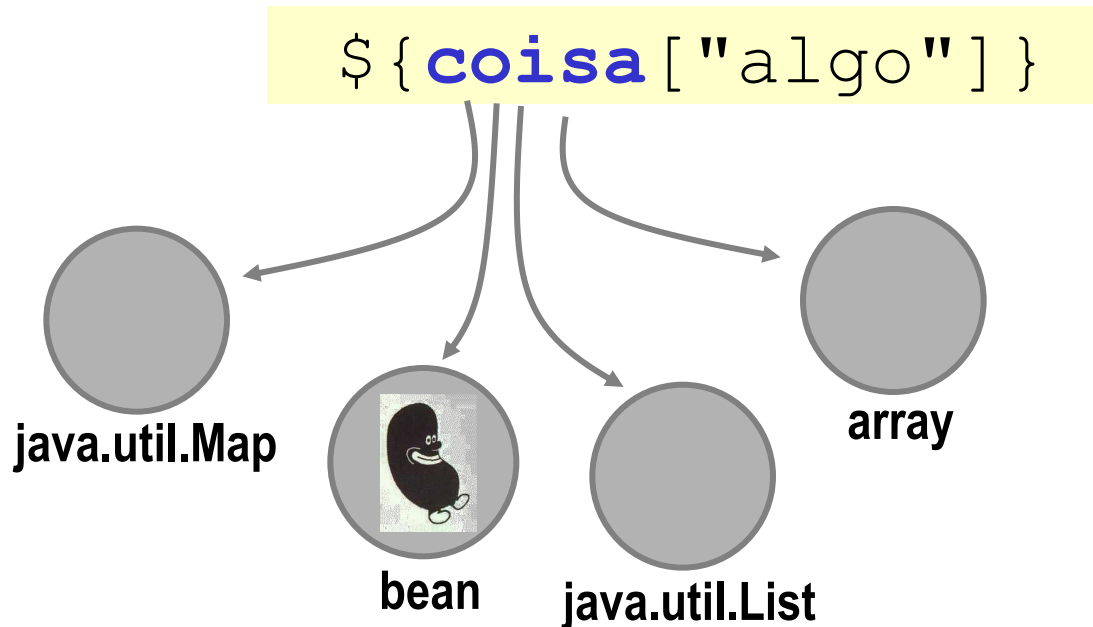
```
<%= aluno.get("nome") %>
```

aluno é um map



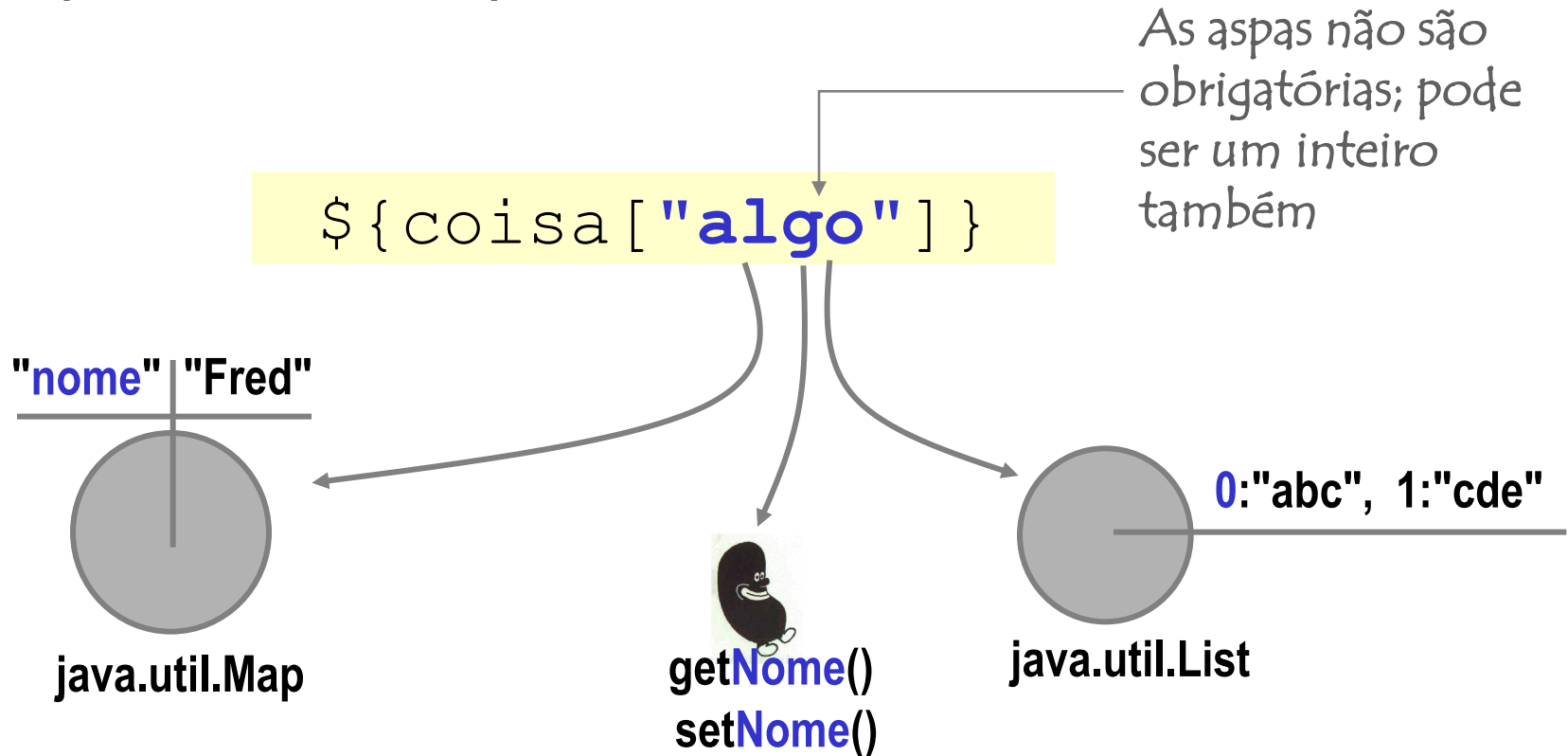
# Expression Language

- Se a expressão é formada por uma variável seguida de um par de [ ], a variável pode ser um mapa, um bean, um List ou um array



# Expression Language

- O que vem entre os [ ] pode ser muitas coisas, dependendo do que for a variável!



# Expression Language

## ■ Código em um servlet:

```
String[] cities = {"Jampa", "Maceio", "Natal"};  
request.setAttribute("cidades", cities);
```

## ■ Código no JSP:

Cidades: `${cidades}`

Primeira cidade: `${cidades[0]}`

Segunda cidade: `${cidades["1"]}`

↑  
Eu disse lá atrás que EL  
não é Java!!!

# Expression Language

## ■ Em um servlet:

```
Map capitais = new HashMap();  
capitais.put("PB", "Jampa");  
capitais.put("RN", "Natal");  
capitais.put("AL", "Maceió");  
request.setAttribute("mapa", capitais);
```

Para mapas e  
beans, ambos  
operadores podem  
ser usados

## ■ Código no JSP:

Paraíba: \${mapa.PB}      → Paraíba: Jampa

Paraíba: \${mapa["PB"]}      → Paraíba: Jampa

# Expression Language

- **Atenção!** Se o conteúdo entre os [ ] não possuir aspas, ele é avaliado
  - Um atributo com o nome é procurado e seu valor é utilizado no local

```
Map capitais = new HashMap();  
capitais.put("PB", "Jampa");  
capitais.put("RN", "Natal");  
capitais.put("AL", "Maceió");  
request.setAttribute("mapa", capitais);  
request.setAttribute("uf", "PB");
```

---

Paraíba: \${mapa[uf]}      → \${mapa["PB"]} → Jampa  
Paraíba: \${mapa["uf"]}    → **Não funciona!!!!**



# Expression Language

- Há objetos implícitos em EL
  - Não são os mesmos objetos implícitos de JSP, exceto o `pageContext`

**pageScope**  
**requestScope** ← Mapas de  
**sessionScope** atributos (um  
**applicationScope** para cada  
escopo)

**param** ← Mapas para os  
**paramValues** parâmetros

**header** ← Mapas para  
**headerValues** cabeçalhos da  
requisição

**cookie** ← Mapa de cookies

**initParam** ← Mapa dos  
parâmetros de  
inicialização do  
contexto

**pageContext** ← Objeto, mas  
pode ser visto  
como um bean

# Expression Language

## ■ No HTML:

```
<form action="mostra.jsp">  
  Nome: <input type="text" name="nome">  
  Endereco: <input type="text" name="endereco">  
  Cartao: <input type="text" name="cartao">  
  Repete: <input type="text" name="cartao">  
</form>
```

## ■ No JSP:

```
Nome: ${param.nome}  
Endereco: ${param.endereco}  
Cartão: ${paramValues.cartao[0]}  
Repete: ${paramValues.cartao[1]}
```

# Expression Language

## ■ Outro exemplo:

```
Host: ${header.host}
Host: ${header["host"]}
```

## ■ Porém... Método: `<%= request.getMethod() %>`

Método: `${request.method}`

- **ERRO!** `request` não é objeto da EL

Método: `${requestScope.method}`

- **ERRO!** `requestScope` só possui os atributos do request e não o request!

Método: `${pageContext.request.method}`

- **CERTO!** use `pageContext` para pegar propriedades da requisição

# Expression Language

## ■ Acessando cookies sem EL:

```
<% Cookie[] cookies = request.getCookies();  
    for(int i=0 ;i < cookies.length ;i++) {  
        if ((cookie[i].getName()).equals("aluno")) {  
            out.println(cookies[i].getValue());  
        }  
    } %>
```

## ■ Com EL: `${cookie.aluno.value}`

# Expression Language

## ■ Acessando parâmetros de contexto sem EL:

```
<context-param>  
  <param-name>jdbc.url</param-name>  
  <param-value>jdbc:postgresql:banco</param-value>  
</context-param>
```

URL: `<%=application.getInitParameter("jdbc.url") %>`

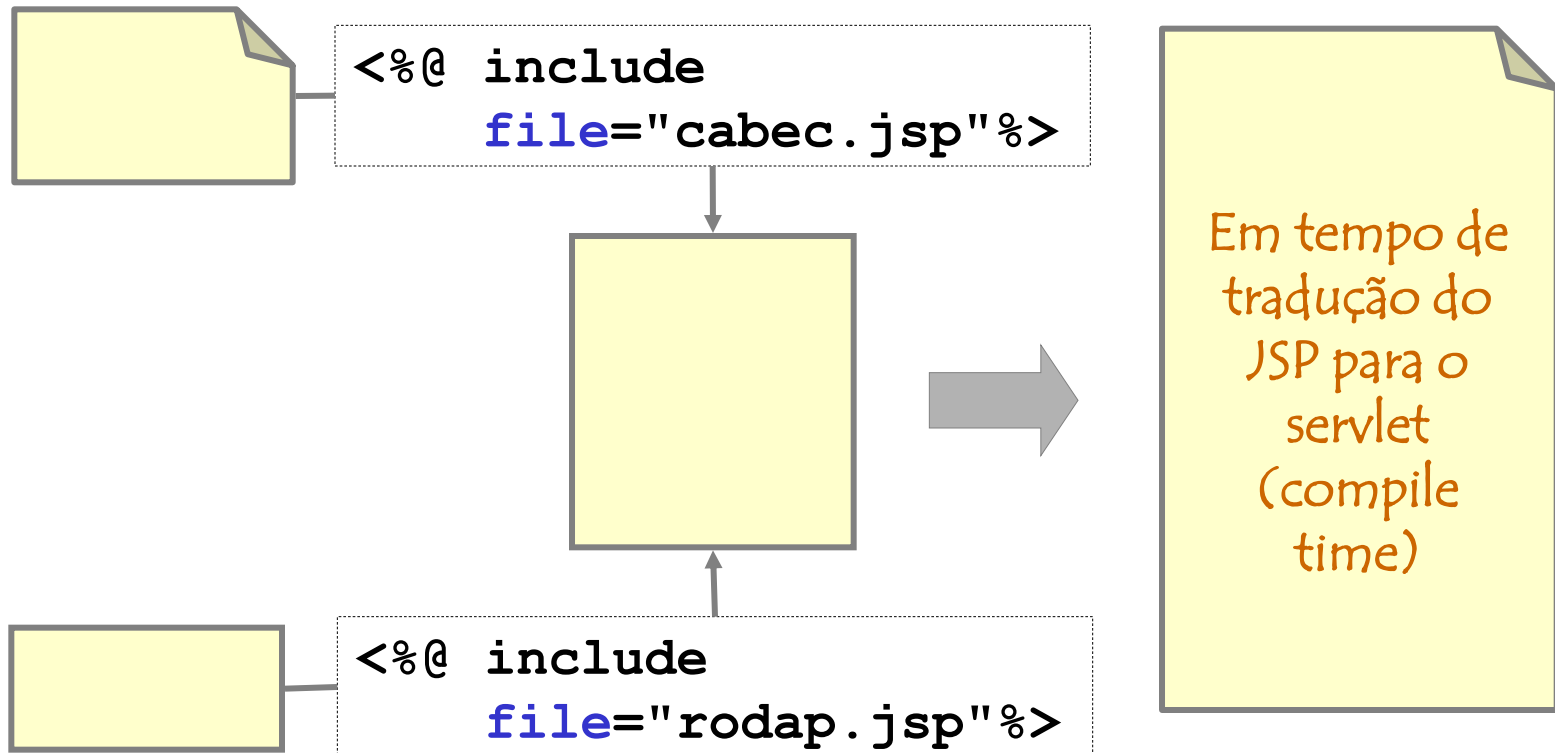
## ■ Com EL: `${initParam["jdbc.url"]}`

~~`${initParam.jdbc.url}`~~

← Ooops!  
Assim não  
pode!

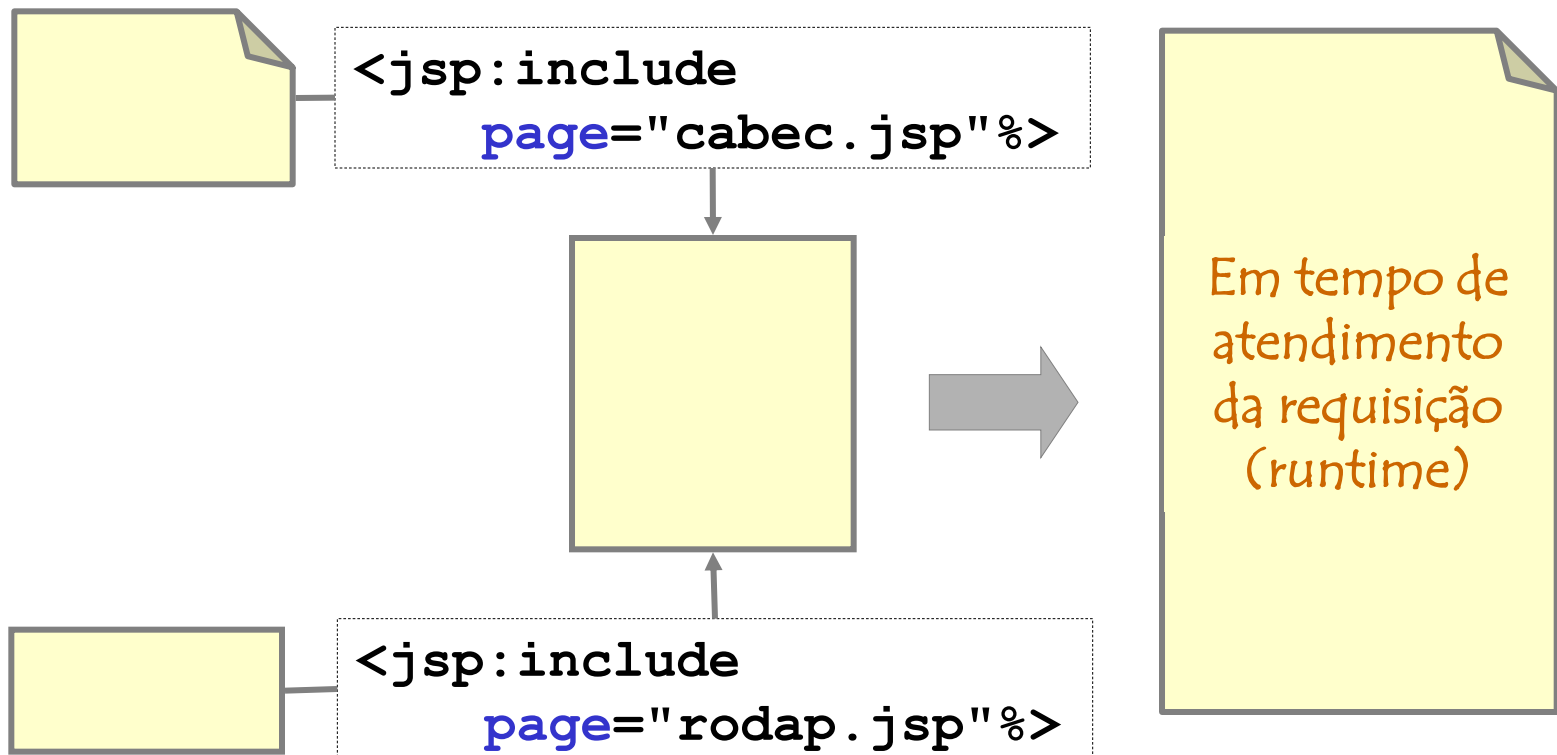
# Reutilização

## ■ Diretiva de inclusão



# Reutilização

## ■ Ação de inclusão



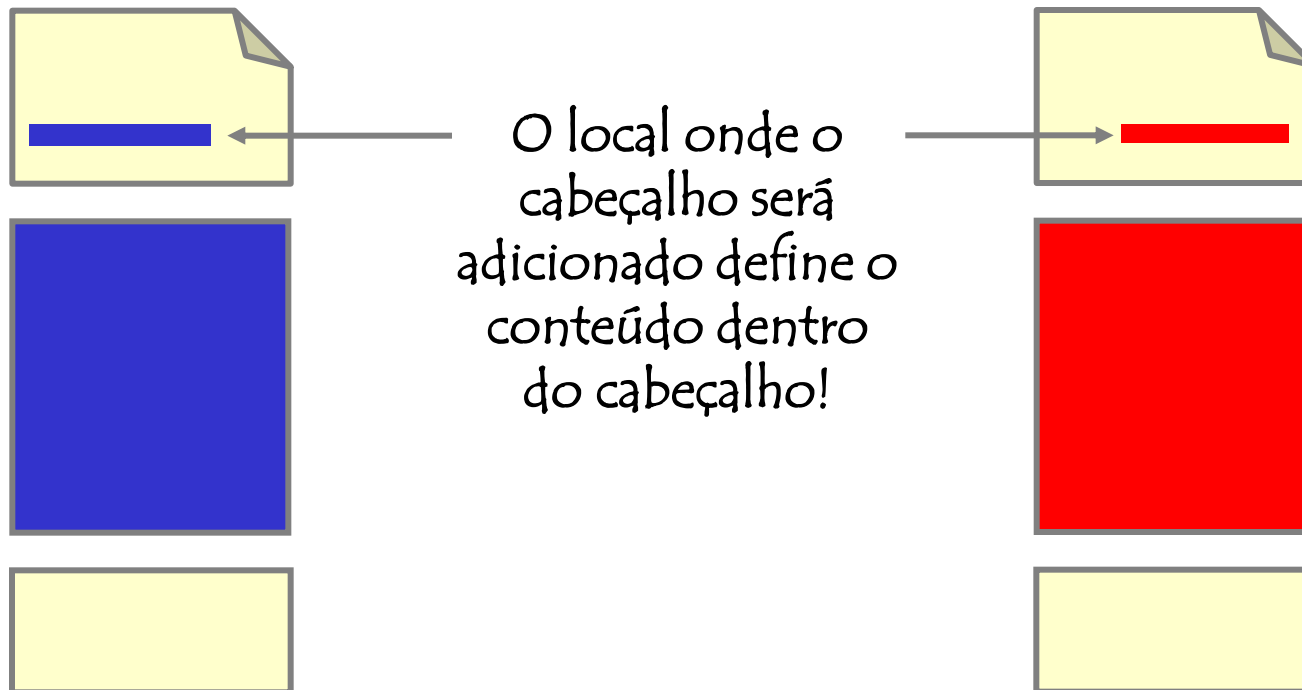
# Reutilização

- Com a diretiva `<%@ include>`
  - O código fonte do servlet fica maior
  - O overhead de execução é menor
  - Alteração no arquivo a ser incluído não é detectada (o tomcat5 já detecta!)
- Com a ação `<jsp:include/>`
  - Há um overhead maior na execução
  - O arquivo incluído pode ser alterado entre requisições sem haver necessidade de redeploy



# Reutilização

## ■ Pedacos parametrizados



# Reutilização

## ■ No JSP que faz a inclusão:

```
<html><body>
```


```
<jsp:include page="cabec.jsp">
```

```
<jsp:param name="titulo" value="Conteúdo  
dependente"/>
```

```
</jsp:include>
```

```
</body></html>
```

Adiciona um novo  
parâmetro para a  
parte incluída



## ■ em cabec.jsp:

```
<br>
```

```
<b>${param.titulo}</b><br>
```

```
<hr>
```



A parametrização  
funciona para casos  
simples como esse. E se  
precisarmos de decisões  
mais elaboradas?  
Teremos que recorrer aos  
scripts novamente?

Aguarde sensacionais revelações no próximo  
capítulo...

# Bibliografia

- **Bashan, B., Sierra, K. e Bates, B.** "Head First Servlets & JSP". Capítulo 8. 2005.