

# JavaServer Pages (JSP)



# Especificação/IDE/Implementação

- Esse curso foi preparado em 03/2015 usando a seguinte especificação, IDE e implementação
  - JavaServer Pages 2.3 (06/2013, JEE 7)
  - JavaServer Pages 2.2 (12/2009, JEE 6)
  - JavaServer Pages 2.1 (05/2006, JEE 5)
  - JavaServer Pages 2.0 (11/2003, JEE 1.4)
- IDE
  - JDK 8u40
  - NetBeans 8.0.2 na distribuição Java EE
- Implementação
  - GlassFish 4.1 (vem no NetBeans)

# Agenda

- O que são JSP?
- Elementos de script
- Variáveis predefinidas
- Inclusão de arquivos
- Encaminhamento de requisições
- Acesso a Java Beans
- Cookies

# O que são JSP

- Páginas HTML com códigos adicionais (.jsp)
  - Códigos executam lógica de negócio para tratar requisições e gerar **conteúdo dinâmico** (ex.: montar listagem de clientes lidos da base de dados)
  - O conteúdo dinâmico é **apresentado** no cliente como uma **página HTML convencional**
  - Permite **uso de componentes JavaBeans** e **mecanismos de extensão** da própria linguagem (*Tag Libraries*)
  - As páginas JSP são e são **transformadas em Servlets automaticamente**

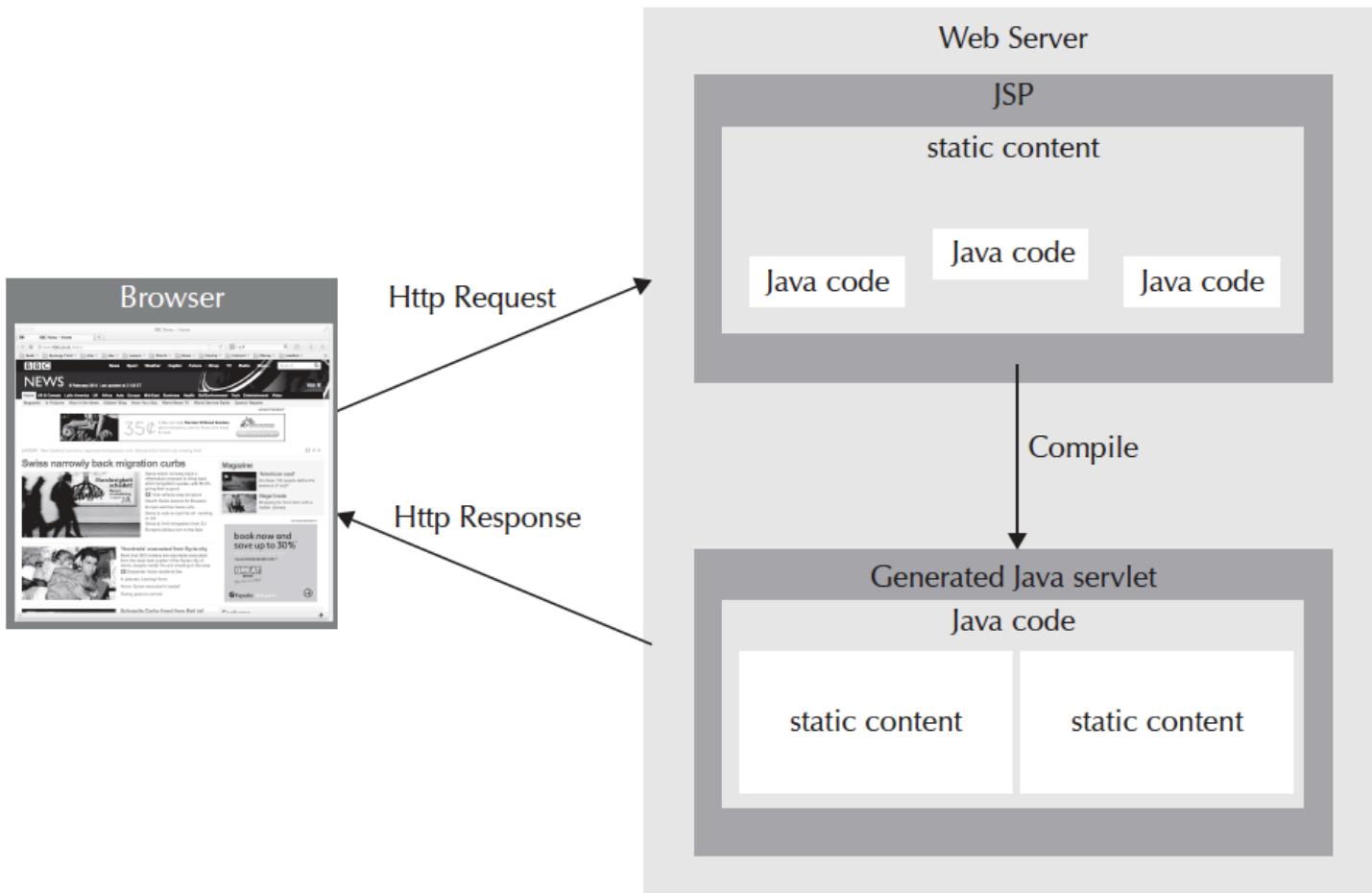
# Comparação com Servlets

- Facilita o desenvolvimento
  - Criar páginas JSP é mais fácil do que criar aplicações completas
  - Código Java é escrito no HTML (no Servlet é o oposto)
  - Mais fácil de alterar do que um Servlet (classe) e não precisam ser compiladas pelo desenvolvedor antes de entrarem em produção, como em um Servlet
- Indicado para apresentação, quando se tem **muito HTML e pouco Java**

# Não confundir!!!

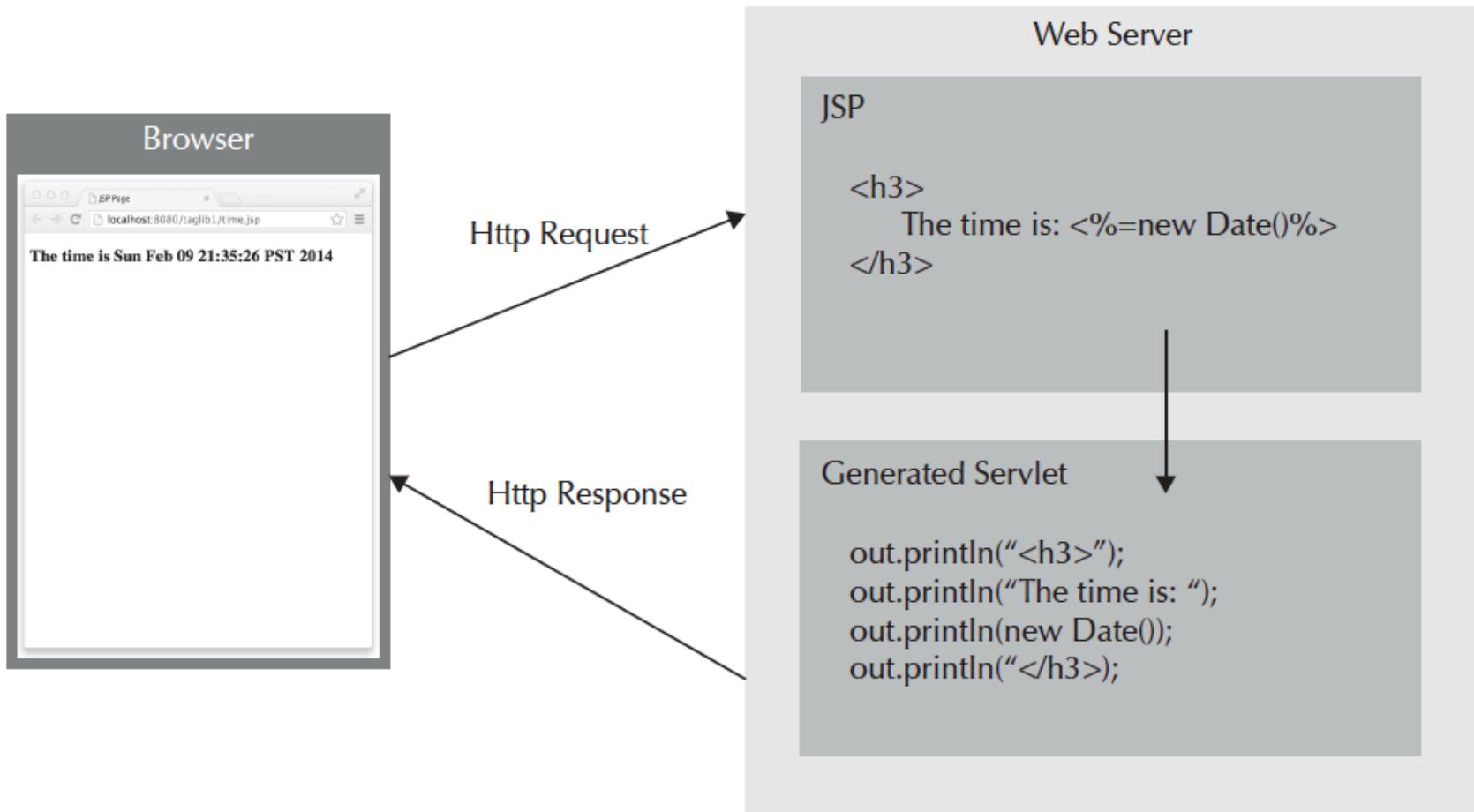
- JSP é uma tecnologia do lado do servidor (*server-side*)
- Tempo de tradução
  - Primeiro acesso
  - Transformação em Servlet
- Tempo de requisição
  - Todo acesso
  - Execução do Servlet associado
- Pode combinar com HTML, Servlets, JavaBeans e outras classes Java

# Compilação JSP → Servlet



Fonte: livro Java EE 7: The Big Picture

# Compilação JSP → Servlet



Fonte: Livro Java EE 7: The Big Picture

# Elementos de script

- Comentários
  - HTML: São exibidos na página HTML gerada  
`<!-- isto é um comentário -->`
  - JSP: Não são inseridos na página HTML gerada  
`<%-- isto é um comentário escondido --%>`
- Declarações
  - Definem variáveis ou métodos para uso subsequente  
`<%! int totalVisitas = 0; %>`  
`<%! public int getTime() { return time; } %>`

# Elementos de script

- Scriptlets
  - Código Java a ser executado

```
<% for (int i = 0; i < 10; i++) { %>
<p>Teste de script</p>
<% } %>
```
- Expressões
  - Valores inseridos na página HTML gerada

O total de visitas é <%= ++total %>

A data atual é <%= new java.util.Date() %>

# Elementos de script

- Diretivas
  - Permite definir propriedades gerais do JSP processadas no momento da tradução para Servlet  
`<%@ ... %>`
- Include
  - Inclusão de arquivos na página  
`<%@ include file="rodape.htm" %>`
- Page
  - Importação de pacotes de classes Java utilizadas  
`<%@ page import="java.util.* , java.io.*" %>`
  - Informações de configuração para geração  
`<%@ page contentType="text/plain" %>`  
`<%@ page language="java" %>`  
`<%@ page session="true" %>`
  - Outras opções disponíveis: buffer, autoflush, info, errorPage, etc.

# Variáveis implícitas

- É possível usar variáveis internas pré-definidas
- **request** – representa a requisição HTTP
  - Tipo: HttpServletRequest
- **response** – representa a resposta HTTP
  - Tipo: HttpServletResponse
- **session** – representa a sessão HTTP associada à requisição
  - Tipo: HttpSession
- **out** – representa a saída de escrita na página gerada
  - Tipo: JSPWriter
- **application** – estrutura de dados compartilhada
  - Tipo: ServletContext
- **config** – dados de configuração do JSP
  - Tipo: ServletConfig
- Confira outros objetos internos pré-definidos !
  - exception, pageContext, page

# Objeto interno *request*

- Recepção de dados
  - Permite a recepção de dados provenientes de formulários dispostos em páginas HTML (métodos *getParameter*, *getParameterNames*, *getParameterValues*)
  - Permite a verificação do método de envio (POST/GET) dos dados de formulário (método *getMethod*)
  - Permite verificar se a conexão entre o cliente e o servidor é segura (método *isSecure*)

```
nome = request.getParameter ("Nome")
endereco = request.getParameter ("Endereco")
bairro = request.getParameter ("Bairro")
telefone = request.getParameter ("Telefone")
```

Nomes de controles  
do formulário que  
disparou o script JSP

# Objeto interno *response*

- Responsável pela manipulação do cabeçalho HTML
  - **addHeader(name, value)**, **setHeader(name, value)** e **addCookie(cookie)**, entre outros métodos
- Direcionamento da aplicação para outras páginas
  - **sendRedirect(location)**

# Objeto interno *session*

- Gerenciamento da memória de sessão:
  - Permite armazenar (***setAttribute***) e recuperar valores (***getAttribute***) da memória de sessão da aplicação
  - Cada valor é referenciado por seu nome, e a lista de todos os nomes pode ser obtida com ***getAttributeNames***
  - O acesso ao objeto ***session*** deve ser realizado antes de qualquer escrita na página de resposta
  - O método ***setMaxInactiveInterval*** do objeto *session* configura o tempo máximo de atividade da sessão
  - O método ***invalidate*** finaliza a sessão, eliminando todos os valores em memória

# Objeto interno **out**

- Permite a impressão de código HTML para a formação de uma página no cliente (método *println*)

```
out.println (“<p>Testando 1, 2, 3</p>”);
```

# Exemplo de elementos de script e variáveis predefinidas

```
<%@ page import="java.util.*" %>  
...  
<%-- Check for AM or PM --%>  
<%! int time = Calendar.getInstance().get(Calendar.AM_PM) ; %>  
<%  
    String nome = request.getParameter("nome");  
    out.println("Olá, " + nome);  
    if (time == Calendar.AM) {  
        %> Bom dia ! <%  
    } else {  
        out.println("Boa tarde !");  
    }  
%>  
...
```

# Exercício

- Criar JSP Alo mundo (alomundo.jsp)
  - listando números de 0 a 99
- Acessar o JSP em
  - <http://localhost:8080/exercicio/alomundo.jsp>

# Inclusão de arquivos

- “`<%@ include ... %>`”
  - Inclui a página alvo em **tempo de tradução**
  - Precisa traduzir novamente a página se uma página incluída for modificada
  - Pode usar as definições feitas nas páginas incluídas
- “`<jsp:include .../>`”
  - Inclui a página alvo em **tempo de requisição**
  - Não precisa traduzir a página se uma página incluída for modificada
  - Exemplo
    - `<jsp:include page="rodape.html" />`

# Inclusão de arquivos

- Passagem de parâmetros para “<jsp:include .../>”
  - Quando a página a ser incluída é um JSP, pode ser necessário passar parâmetros para esse página
  - O request original é automaticamente passado para a página incluída
  - É possível passar novos parâmetros com
    - <jsp:param name=... Value=... />
  - Exemplo
- As páginas incluídas podem ficar em WEB-INF
  - Não serão acessadas diretamente pelo cliente

# Encaminhamento

- “jsp:forward ... /”
  - Redirecionar para uma nova página  
`<jsp:forward page=“http://www.google.com” />`
  - A página JSP origem não pode ter iniciado a escrita de resposta se for encaminhar a requisição

**Raciocínio:** Esse tipo de operação, assim como acesso a cookies e sessão, precisa enviar dados pelo cabeçalho. Se o corpo da página já tiver começado a ser respondido ao cliente, não será possível mais alterar o cabeçalho.

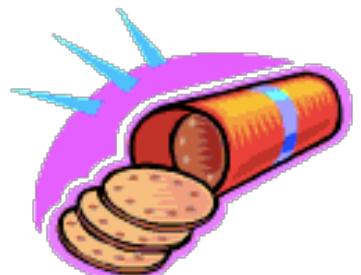
**Lembre-se:** A comunicação entre cliente e servidor (e vice-versa) é feita por demanda.

# Exercício

- Criar um JSP para somatório, onde o valor inicial e o valor final são informados
- Informar o número de vezes que
  - O usuário acessou o serviço na mesma sessão
  - O usuário acessou o serviço no mesmo browser
  - Todos os usuários acessaram o serviço desde quando o servidor entrou no ar
- Incluir as páginas padrões cabecalho.html e rodape.html no JSP de somatório
- Encaminhar para uma página padrão de erro caso algum parâmetro não tenha sido informado

# Cookies

- Um mecanismo de gerenciamento de sessão
- Um *cookie* armazena um pequeno trecho de informação (par nome-valor) que podem ser recuperados ou alterados nos lados cliente e servidor
- O *cookie* é enviado através do cabeçalho HTTP



# Cookies

- Problemas com *cookies*
  - Clientes podem desabilitar a recepção e armazenamento de *cookies* em seu navegador
  - A informação em *cookies* não é segura
  - Clientes podem alterar o conteúdo dos *cookies* em disco
  - *Cookies* devem ser pequenos, normalmente limitados a 4Kb de memória

# Cookies

- Recuperando *Cookies*
  - O objeto interno **request** oferece um método para acesso aos *cookies* recebidos em uma requisição de página
  - Cada página recebe um conjunto de cookies, representados em um vetor

```
Cookie cookies[] = request.getCookies();  
  
for (int i = 0; i < cookies.length(); i++)  
    out.println (cookies[i].getValue());
```

# Cookies

- Armazenando *Cookies*
  - O objeto interno **response** oferece um método para adicionar um cookie em uma página de resposta
  - Diversos cookies podem ser adicionados em uma mesma página de resposta

```
Cookie c = new Cookie("nome", "valor");
c.setMaxAge(tempo); // em segundos
response.addCookie(c);
```

# *Expression Language (EL)*

- Permite, de forma simples, avaliar expressões ou acessar valores de variáveis
- Sintaxe
  - \${expressão}
- Exemplo
  - \${idade > 18}
  - \${param["idade"]} ou \${param.idade}
  - \${pageContext.servletContext.serverInfo}

# *Expression Language (EL)*

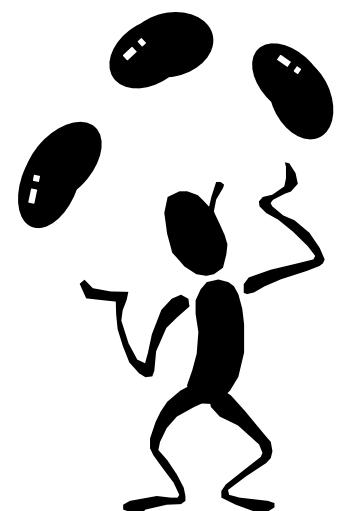
- Expressões aceitam
  - Literais (booleano, numérico, String, null)
  - Objetos
  - Operadores aritméticos
  - Operadores relacionais
  - Operadores lógicos (inclusive em forma textual: or e and)
  - Decisão ( A ? B : C )

# Separação de responsabilidades

- Antes tinha muito HTML no código Java (Servlet)
- Agora tem muito Java no código HTML (JSP)
- Para tentar minimizar esse problema, são utilizadas duas estratégias de separação de responsabilidades
  - JavaBeans (classes de dados)
  - Tag Libraries (classes de controle)

# JavaBeans

- Permitem a construção de componentes reutilizáveis
- Um JavaBean é uma classe implementada em Java que encapsula propriedades com métodos get e set
- JavaBeans podem ser utilizados em:
  - Applets
  - Aplicações standalone
  - *Server side scripts (JSP)*
  - Outros JavaBeans



# JavaBeans

- Características
  - Construtor default, sem argumentos
    - Útil para atribuir valores iniciais para as propriedades
  - Atributos devem ser privados!
  - Atributos devem ser acessados pelos métodos getXXX e setXXX
    - XXX é o nome do atributo
  - Atributos booleanos devem ser acessados pelos métodos isXXX e setXXX

# JavaBeans

```
...
public class PontoBean {
    private int x, y;

    public PontoBean() {
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    ...
}
```

Atributos privados

Construtor default

Acesso de Leitura

Acesso de Escrita

# JavaBeans

- Declaração de componente
  - Tag “**<jsp:useBean ... />**”
  - Parâmetro “**id**”: nome da variável que acessa o bean no JSP
  - Parâmetro “**scope**”: Duração do bean
    - **page**: uso somente nessa página (default)
    - **request**: acessível em outros JSP/Servlet via include
    - **session**: acessível em toda a sessão
    - **application**: acessível por todos os JSP/Servlet (global)
  - Parâmetro “**class**”: Classe (tipo) do bean
  - Exemplo:  
`<jsp:useBean id="ponto" scope="page" class="beans.PontoBean" />`
- Dica: O nome da classe deve ser totalmente qualificado (incluir pacote)

# JavaBeans

- Leitura de dados do bean
  - Tag “**<jsp:getProperty ... />**”
  - Parâmetro “**name**”: nome da variável que acessa o bean no JSP
  - Parâmetro “**property**”: Propriedade a ser lida
  - Exemplo:  
`<jsp:getProperty name="ponto" property="x" /> ou ${ponto.x}`

- Escrita de dados do bean
  - Tag “**<jsp:setProperty ... />**”
  - Parâmetro “**name**”: nome da variável que acessa o bean no JSP
  - Parâmetro “**property**”: Propriedade a ser escrita
  - Parâmetro “**value**”: Valor a ser escrito na propriedade
  - Exemplo:  
`<jsp:setProperty name="ponto" property="x" value="14"/>`

# JavaBeans

- Componentes JavaBeans podem ser utilizados para troca de informações entre camadas da aplicação
- Implementado através do padrão **Value Object**
  - Java Bean representando o formulário HTML preenchido pelo cliente
  - Uma propriedade do bean por campo do HTML
- Escrita de TODOS os dados do bean com os valores do formulário HTML preenchido

```
<jsp:setProperty name="ponto" property="*"/>
```
- Características
  - Valores default são usados quando uma propriedade não é informada
  - Conversão de tipos automática – pode gerar erro de *casting*
  - *Case sensitive*

# Tag Library

- Também conhecida como Tag Library ou JSP Standard Tag Library (JSTL)
- Visam reduzir código de controle Java no JSP
- Permitem acesso a serviços como
  - Execução de SQL
  - Acesso a XML
  - Formatação de texto
  - Manipulação de string
  - Controle de fluxo

# Tag Library

- Antes de usar uma tag Lib, é necessário declarar
  - Exemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

# Tag Library

- Core (prefixo c)
  - URI: <http://java.sun.com/jsp/jstl/core>
- XML (prefixo x)
  - URI: <http://java.sun.com/jsp/jstl/xml>
- Internacionalização (prefixo fmt)
  - URI: <http://java.sun.com/jsp/jstl/fmt>
- SQL (prefixo sql)
  - URI: <http://java.sun.com/jsp/jstl/sql>
- Funções de apoio (prefixo fn)
  - URI: <http://java.sun.com/jsp/jstl/functions>

# Tag Library (Decisão)

- Decisão simples (código opcional)
  - <c:if test="“condição”">
- Exemplo

```
<c:if test="${myBean.readableDate=='PM'}">  
    time for tea!  
</c:if>  
<c:if test="${myBean.readableDate=='AM'}">  
    time for coffee!  
</c:if>
```

# Tag Library (Decisão)

- Decisão complexa (código alternativo)

```
<c:choose>
  <c:when test="${customer.category == 'trial'}" >
    ...
  </c:when>
  <c:when test="${customer.category == 'member'}" >
    ...
  </c:when>
  <c:when test="${customer.category == 'preferred'}" >
    ...
  </c:when>
  <c:otherwise>
    ...
  </c:otherwise>
</c:choose>
```

# Tag Library (Repetição)

- Loop em uma sequência de números
  - <c:forEach var=“nome do cursor” begin=“valor inicial” end=“valor final” step=“incremento” >
- Loop em um array ou coleção
  - <c:forEach var=“nome do cursor” items=“nome da coleção” >
- Em ambos os casos, o cursor é acessível por
  - \${nome do cursor}

# Exercício

- Exiba uma contagem de 1 a 10 usando a tag `forEach`
- Use a tag `forEach` para listar todos os dados passados no cabeçalho da requisição
  - O cabeçalho é representado por um objeto `header` do tipo Map
  - Cada entrada no objeto `header` representa uma tupla key-value do cabeçalho (acessar o item com `.key` e `.value`)

# Exercício

- Faça uma aplicação que registra lembretes para cada um dos usuários, e que permita a listagem de todos os lembretes do usuário
  - Use seção para guardar os lembretes
  - Crie um JavaBean Lembretes
  - Para a listagem dos lembretes do usuário, use a tag forEach

# JavaServer Pages (JSP)

