J550

Desenvolvimento de Aplicações Web com Servlets e JSP

Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este curso explora os recursos da linguagem Java para o desenvolvimento de aplicações Web*
- *Tem como finalidade torná-lo(a) capaz de criar aplicações Web interativas usando os mais poderosos recursos disponíveis*
- *O ambiente utilizado é open-source (Tomcat ou JBoss) e as aplicações desenvolvidas devem executar em qualquer servidor compatível com as especificações*
  - *Servlet 2.3 em diante*
  - *JSP 1.2 em diante*

Pré-requisito essencial

Experiência com programação em Java

e

Conhecimentos elementares de HTML

(tabelas, formulários)

Assuntos abordados

- Este curso explora os seguintes assuntos
  - Criação de aplicações Web usando Java: servlets e JSP
  - Filtros e correntes de filtros
  - Uso de custom tags e criação de tags simples
  - Criação de componentes J2EE para a Web: arquivos WAR com configuração em web.xml
  - Integração de aplicações Web com bancos de dados
  - Recursos elementares de autenticação e autorização controlados em aplicações Web
  - Boas práticas de arquitetura e principais padrões de projeto para aplicações Web, como MVC, View Helpers
  - Uma visão geral do framework MVC Struts
  - Jakarta Tomcat como ambiente de desenvolvimento

Assuntos não abordados

- Este curso não aborda os seguintes assuntos
  - Questões relacionadas à segurança na Web além de simples autenticação e autorização (não trata de criptografia, conexão segura, auditoria, etc.)
  - Enterprise JavaBeans
  - Implementação avançada de custom tags: é abordada apenas o uso da API básica, usando as interfaces Tag e BodyTag
  - JSTL avançado: o assunto é abordado superficialmente
  - Struts avançado: o framework é abordado superficialmente
  - JavaServer Faces
  - Uso de Datasources e connection pools
  - Design de páginas Web, XML e HTML elementar

Roteiro (I)

- Parte I - Fundamentos e Servlets
  - 1. Fundamentos de arquitetura Web
  - 2. Servlets
  - 3. Contextos
  - 4. Sessões e escopo
  - 5. Aplicações Web J2EE (WARs)
  - 6. Filtros
- Parte II - Integração e Segurança
  - 7. Segurança e controle de erros
  - 8. Integração com bancos de dados

Roteiro (II)

- *Parte III - Java Server Pages*
  - *9. Introdução a Java Server Pages*
  - *10. JSP com JavaBeans e páginas compostas*
  - *11. Taglibs e JSTL*
  - *12. Aplicações MVC com JSP e servlets*
- *Parte IV - Tópicos avançados**
  - *13. Padrões de Projeto J2EE para a camada Web*
  - *14. Testes em aplicações Web com o Cactus*
  - *15. Introdução ao framework Apache Struts*
  - *16. Como criar Custom Tags*
  - *17. Aplicações Web em J2EE*

** Abordagem superficial (opcionais)*

Metas

- *Ao final deste curso o aluno terá condições de*
  - *Configurar um servidor para que rode servlets e JSP*
  - *Instalar, alterar e desenvolver aplicações Web eficientes usando JSP, servlets, filtros e JavaBeans, controle de sessões, segurança e integração com bancos de dados*
  - *Usar tags personalizados e conhecer a API de extensão do JSP usada para desenvolver novos tags.*
  - *Escolher entre técnicas (J2EE patterns, MVC) e ferramentas (Struts) para separar a apresentação do código e tornar suas aplicações mais eficientes e mais fáceis de manter*
  - *Distribuir suas aplicações em arquivos WAR para instalação automática em servidores J2EE.*

Como tirar o melhor proveito deste curso

- Faça perguntas
- Faça os exercícios
- Explore os exemplos
- Vá além dos exemplos e exercícios: invente exemplos similares, teste trechos de código
- Explore e se familiarize com a documentação e as especificações de JSP, servlets e J2EE
- Procure desenvolver um projeto que utilize JSP, servlets ou J2EE, seja no trabalho, seja no seu tempo livre
- Leia revistas, artigos e livros sobre Java, JSP, servlets e J2EE e mantenha-se atualizado.

Recursos didáticos utilizados

- Apresentação
  - Slides (em alguns módulos), demonstrações interativas (veja CD)
  - Roteiros didáticos em livros-texto (veja a seguir)
- Exercícios são propostos ao final de cada módulo para que o aluno possa experimentar cada tecnologia J2EE
  - Aplicações triviais tipo Hello World (para fixar conceitos básicos)
  - Aplicações maiores (exemplos dos livros-texto) que devem ser terminadas, configuradas ou instaladas

CD

- O CD que acompanha este curso contém todo o material necessário, software e fontes adicionais de informação
- A sua estrutura geral é a seguinte
  - **cap01** a **cap17**: arquivos com código-fonte correspondente a cada módulo do curso. A maioria possui um ou mais build.xml (roteiro para o Ant) que permite instalar e rodar as aplicações
  - **slides**: contém as apresentações em PDF utilizadas em aula
  - **software**: contém todo o software usado em aula e mais (J2SDK, J2EE SDK Win e Linux, JBoss, Tomcat, JEdit, JUnit, Cactus, HttpUnit, etc.)
  - **docs**: livros-texto em PDF, tutoriais online da Sun, especificações de servlets e JSP em PDF, whitepapers, código-fonte
- O objetivo do CD é facilitar o acesso ao material do curso. Sempre que possível, procure versões mais atuais na Internet.

Apresentações

- Instrutor: Helder da Rocha (helder@acm.org)
  - Utiliza Java desde 1995
  - XML, J2EE, JSP, servlets, Web
  - http://www.argonavis.com.br
- Alunos?
  - Nome?
  - O que faz? Onde trabalha?
  - Background (sabe Web, HTML, HTTP, CGI? ASP, PHP? Web? Que linguagens e plataformas?)
  - Expectativas?

Check-list de Instalação (J500/530/550)

- J2SDK 1.4.0
  - Rode /software/java/j2sdk1.4.0-win.exe
- J2SDK EE 1.3.1 e documentação J2EE
  - Rode /software/j2ee/j2sdkee-1_3_1-win.exe
- JBoss 3.0.0 ou Tomcat
  - Descompacte o arquivo /software/j2ee/jboss-3.0.0.zip em C:\
- Jakarta Ant
  - Descompacte /software/java/jakarta-ant-1.5-bin.zip em C:\
  - Mude o nome do diretório raiz criado (jakarta-ant-*) para ant
- JEdit 4.0
  - Rode /software/java/jedit40install.jar e siga as instruções

Check-list de configuração (J500/530/550)

- Crie as seguintes variáveis de ambiente
  - JAVA_HOME=c:\j2sdk1.4.0
  - ANT_HOME=c:\ant
  - J2EE_HOME=c:\j2sdkee1.3.1
  - JBOSS_HOME=c:\jboss-3.0.0 ou TOMCAT_HOME=c:\tomcat-4.0
- Acrescente, à sua variável PATH os seguintes caminhos
  - %JAVA_HOME%\bin;%J2EE_HOME%\bin;%ANT_HOME%\bin;
- Crie atalhos na sua área de trabalho para:
  - c:\j2sdkee1.3.1\bin\cloudscape98.bat (J500/530)
  - c:\jboss-3.0.0\bin\run.bat ou
  - c:\tomcat-4.0\bin\startup.bat e c:\tomcat-4.0\bin\shutdown.bat
- Suporte a Cloudscape (banco de dados) - J500/530
  - Copie D:\jboss\cloudscape_config\j2ee_ri_windows\cloudscape98.bat para c:\j2sdkee1.3.1\bin\
  - Siga as outras instruções de D:\jboss\cloudscape_config\README.txt para copiar arquivos para diretórios do JBoss

helder@acm.org

argonavis.com.br

J550 - Revisão 4.0 - Abril 2003

Servlets e JSP, Dezembro 2000

Fundamentos de arquitetura Web

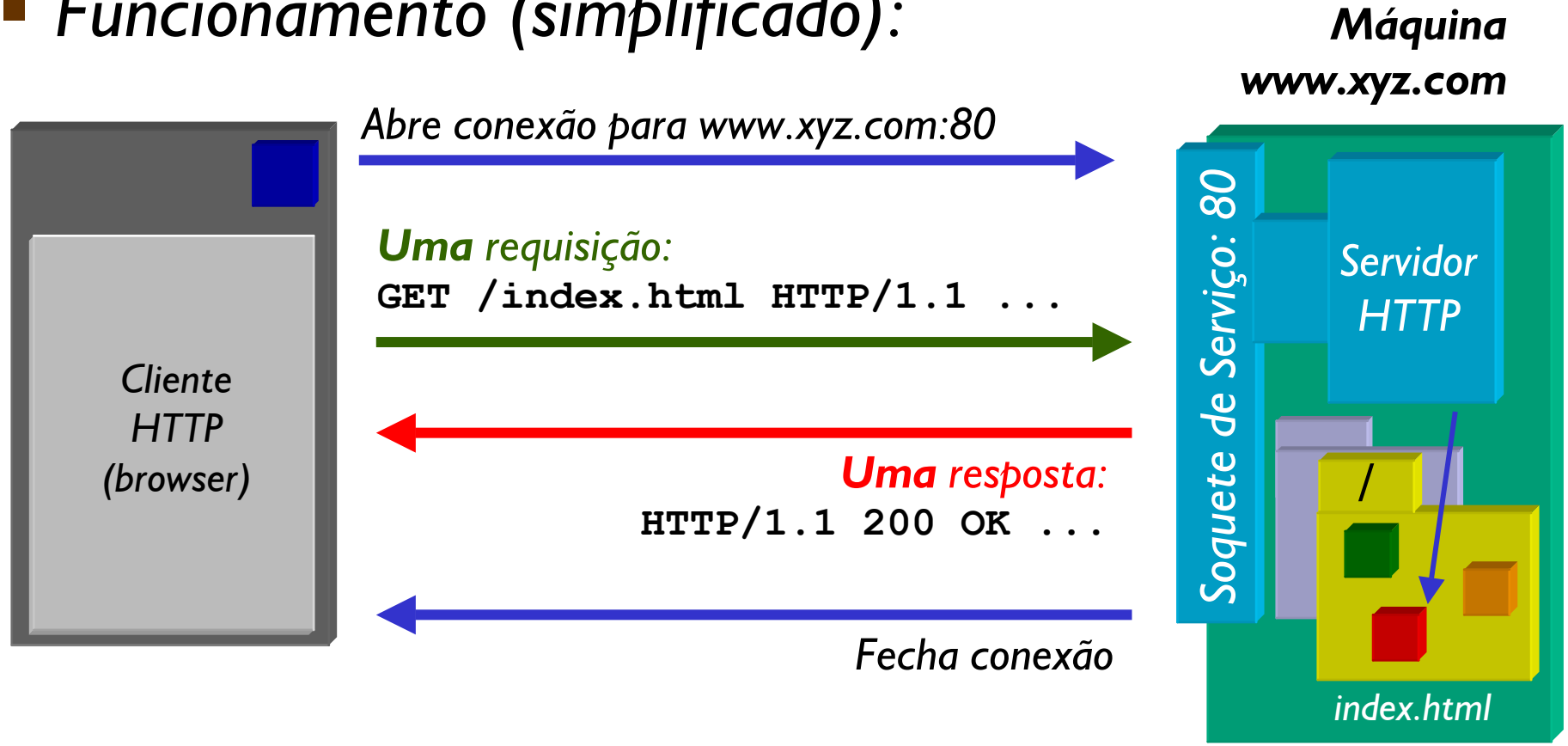
Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Objetivos

- *Este módulo apresenta uma visão geral da plataforma Web*
  - *Lado-cliente*
  - *Lado-servidor*
  - *Protocolo de comunicação HTTP*
- *Descreve o funcionamento de HTTP e as principais tecnologias utilizadas na Web*
- *Apresenta tecnologias Java para a Web: servlets e JSP*
- *Introduz o ambiente de desenvolvimento: Tomcat*

A plataforma Web

- *Baseada em HTTP (RFC 2068)*
  - *Protocolo simples de transferência de arquivos*
  - *Sem estado (não mantém sessão aberta)*
- *Funcionamento (simplificado):*



Cliente e servidor HTTP

- Servidor HTTP
  - Gerencia sistema virtual de arquivos e diretórios
  - Mapeia pastas do sistema de arquivos local (ex: c:\htdocs) a diretórios virtuais (ex: /) acessíveis remotamente (notação de URI)
- Papel do servidor HTTP
  - Interpretar requisições HTTP do cliente (métodos GET, POST, ...)
  - Devolver resposta HTTP à saída padrão (código de resposta 200, 404, etc., cabeçalho RFC 822* e dados
- Papel do cliente HTTP
  - Enviar requisições HTTP (GET, POST, HEAD, ...) a um servidor. Requisições contém URI do recurso remoto, cabeçalhos RFC 822 e opcionalmente, dados (se método HTTP for POST)
  - Processar respostas HTTP recebidas (interpretar cabeçalhos, identificar tipo de dados, interpretar dados ou repassá-los.

* Padrão Internet para construção de cabeçalhos de e-mail

Principais métodos HTTP (requisição)

- ***GET*** *- pede ao servidor um arquivo (informado sua URI) absoluta (relativa à raiz do servidor)*

```
GET <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores> (RFC 822)
<linha em branco>
```

  - *GET pode enviar dados através da URI (tamanho limitado)*

```
<uri>?dados
```

  - *Método HEAD é idêntico ao GET mas servidor não devolve página (devolve apenas o cabeçalho)*

- ***POST*** *- envia dados ao servidor (como fluxo de bytes)*

```
POST <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores>
<linha em branco>
<dados>
```

Cabeçalhos HTTP

- *Na **requisição**, passam informações do cliente ao servidor*
  - *Fabricante e nome do browser, data da cópia em cache, cookies válidos para o domínio e caminho da URL da requisição, etc.*
- *Exemplos:*

```
User-Agent: Mozilla 5.5 (Compatible; MSIE 6.0; MacOS X)
If-Modified-Since: Thu, 23-Jun-1999 00:34:25 GMT
Cookies: id=344; user=Jack; flv=yes; mis=no
```

- *Na **resposta**: passam informações do servidor ao cliente*
  - *Tipo de dados do conteúdo (text/xml, image/gif) e tamanho, cookies que devem ser criados. endereço para redirecionamento, etc.*
- *Exemplos:*

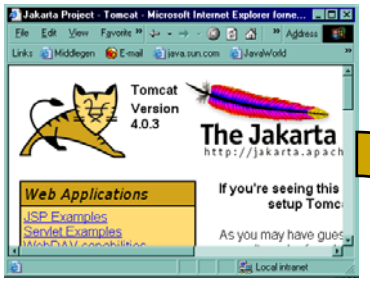
```
Content-type: text/html; charset-iso-8859-1
Refresh: 15; url=/pags/novaPag.html
Content-length: 246
Set-Cookie: nome=valor; expires=Mon, 12-03-2001 13:03:00 GMT
```

Comunicação HTTP: detalhes

- *1. Página HTML*

```
<img src="tomcat.gif" />
```

Interpreta HTML



Gera requisição GET

- *2. Requisição: browser solicita imagem*

```
GET tomcat.gif HTTP/1.1
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
Cookies: querty=uiop; SessionID=D236S11943245
→
```

Linha em branco termina cabeçalhos

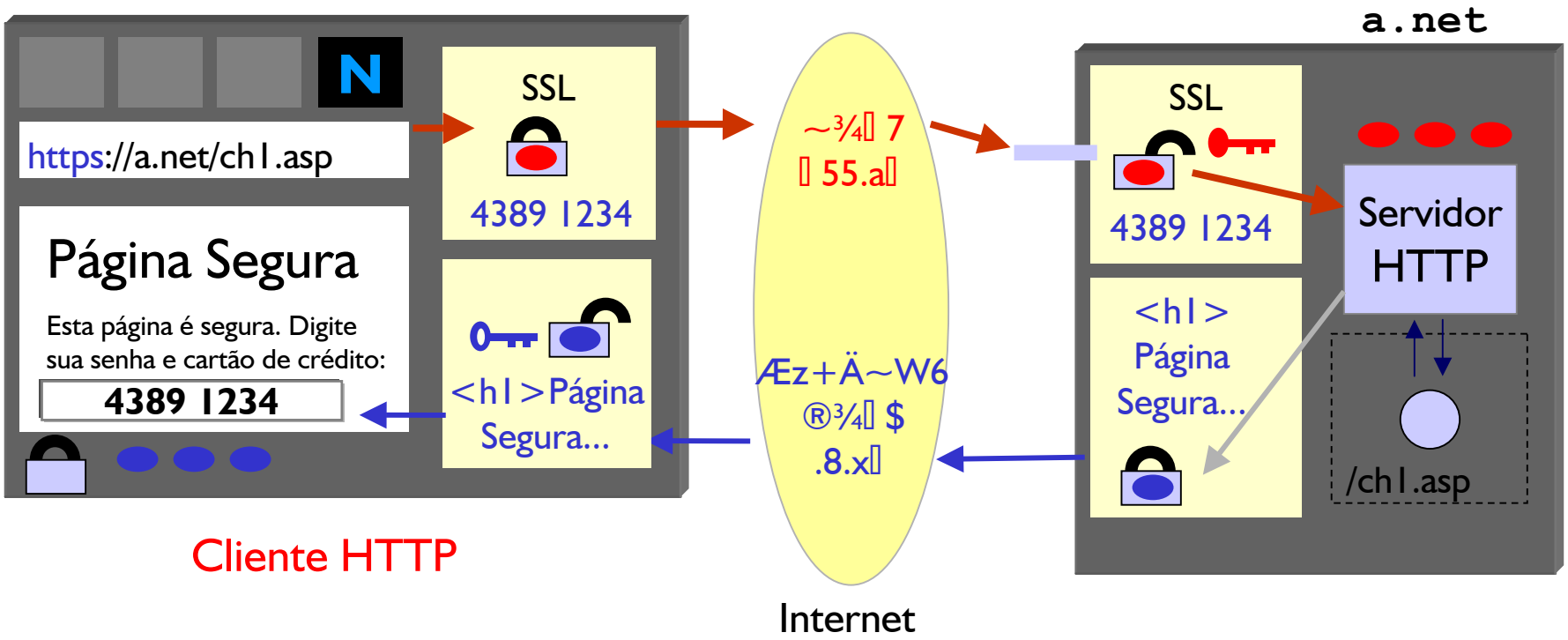
- *3. Resposta: servidor devolve cabeçalho + stream*

```
HTTP 1.1 200 OK
Server: Apache 1.32
Date: Friday, August 13, 2003 03:12:56 GMT-03
Content-type: image/gif
Content-length: 23779
→
!#GIF89~¾ 7
.55.a  6¤Ü4  ...
```

tomcat.gif

SSL - Secure Sockets Layer

- *Camada adicional na comunicação HTTP que introduz criptografia da comunicação*
- *Tanto o browser quanto o servidor têm que suportar o recurso para que uma transação seja segura*
- *Porta default : 443*



Serviço Web: funções

- Serviço de informações
  - finalidade: publicação de informações, multimídia
  - interatividade: limitada a hipertexto
  - tecnologias (passivas): HTML, folhas de estilo
- Serviço de aplicações locais (rodam no cliente)
  - finalidade: oferecer mais recursos interativos ao cliente
  - interatividade: limitada pelo cliente
  - tecnologias (ativas): JavaScript, applets Java, Flash, ActiveX
- Serviço de aplicações cliente/servidor
  - finalidade: oferecer interface para aplicações no servidor
  - interatividade: limitada pela aplicação e servidor Web
  - tecnologias (ativas): CGI, ASP, ISAPI, Servlets, JSP

Serviço de informações: Tecnologias de apresentação

- *HTML 4.0 (HyperText Markup Language)*
  - *Coleção de marcadores (SGML) usados para formatar texto:*
    - *<H2>Cabeçalho de Nível 2</H2>*
    - *<P>Primeiro parágrafo</P>*
  - *Nada diz sobre aparência (browser é quem decide). Define apenas estrutura e conteúdo.*
- *CSS 2.0 (Cascading Style Sheets)*
  - *Lista de regras de apresentação para uma página ou todo um site (linguagem declarativa)*
  - *Depende da estrutura do HTML. Define forma.*
- *Padrões W3C (http://www.w3.org)*

HTML - HyperText Markup Language

- Define a interface do usuário na Web
- Pode ser usada para
  - Definir a estrutura do texto de uma página (que o browser posteriormente formatará com uma folha de estilos)
  - Incluir imagens numa página
  - Incluir vínculos a outras páginas
  - Construir uma interface com formulários para envio de dados ao servidor
  - Servir de base para aplicações rodarem dentro do browser (applets Java, plug-ins, vídeos, etc.)

CSS - Cascading Style Sheets

- *Linguagem usada para definir folhas de estilo que podem ser aplicadas a todo o site.*
  - *Cuida exclusivamente da aparência (forma) da página*
  - *Permite posicionamento absoluto de textos e imagens, manipulação com fontes, cores, etc.*
- *Regras são colocadas em arquivo de texto .css:*

```
H1, H2     { color: rgb(91.5%, 13%, 19%);
                font-size: 72pt; margin-top: 0.6cm}
P.citacao    {font-family: Garamond, serif;
               font-weight:: 300}
```

Limitações

- *HTML e CSS são linguagens declarativas, interpretadas pelo browser, que definem apenas como a informação será organizada e apresentada.*
- *Não oferecem recursos de programação.*
- *Os formulários criados com HTML não fazem nada (eles precisam ser vinculados a uma aplicação)*
- *Não é possível construir aplicações Web interativas utilizando apenas CSS e HTML*
- *Dynamic HTML: solução para alguns problemas*
  - *apresentação + estrutura + interatividade*

Serviço de aplicações: Tecnologias interativas

- Lado-cliente
  - Linguagens de extensão: JavaScript, VBScript
  - Plug-ins e componentes (applets, activeX)
  - Soluções integradas: DHTML
- Persistência de sessão cliente-servidor
  - Cookies
- Lado-servidor
  - CGI, plug-ins do servidor e componentes
  - Linguagens de extensão: SAPIs, ASP, JSP, PHP

Tecnologias lado-cliente

- Estendem a funcionalidade básica do browser (que é apresentação de informação)
- Permitem criar uma interface do usuário dinâmica
  - Tratamento de eventos
  - Alteração dinâmica do conteúdo ou da apresentação
  - Realização de cálculos e computação
  - Disposição de recursos não disponíveis no browser
- Principais tecnologias
  - Extensões do HTML (scripts): JavaScript, VBScript, linguagens proprietárias
  - Extensões do browser (componentes): Applets, ActiveX, Plug-ins

Scripts : extensões do HTML

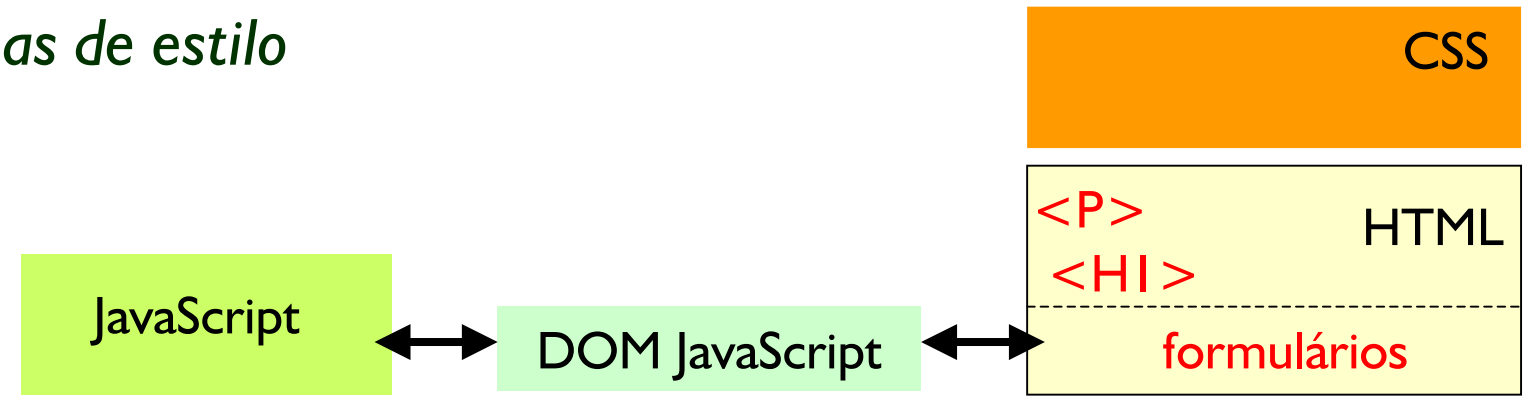
- Forma mais flexível de estender o HTML
- Código geralmente fica visível na página:
  - ```
    <head>
    <script language="JavaScript"><!--
      function soma(a, b) {
         return a + b;
      }
    //--></script></head>
    ```
- Linguagens de roteiro (script) mais populares
  - VBScript: baseado na sintaxe do Visual Basic. MSIE-only.
  - JavaScript/JScript: sintaxe semelhante a de Java
- Código é interpretado diretamente pelo browser (e não por uma máquina virtual, como ocorre com os applets)
    ```

JavaScript e ECMAScript

- JavaScript não é uma versão limitada de Java.
  - Possui sintaxe procedural semelhante, mas é interpretada, baseada em objetos e bem menor.
- JavaScript pode ser usada no browser ou no servidor. A linguagem possui duas partes
  - Núcleo (padrão ECMA chamado de ECMAScript)
  - Modelo de objetos do documento (quando usada no browser) ou do servidor (quando usada no servidor em tecnologias ASP, Livewire, etc.) - implementa padrão DOM
- O núcleo da linguagem define as estruturas de programação, sintaxe e objetos de propósito geral.
- Quando usada no browser, várias estruturas do HTML são acessíveis como 'objetos' JavaScript, permitindo que a linguagem os manipule.

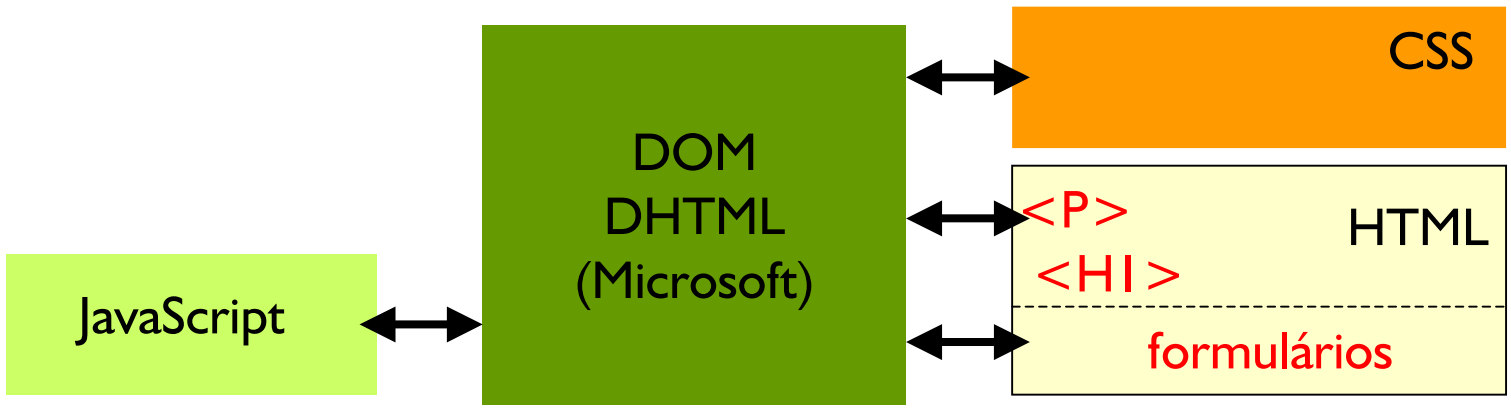
JavaScript e DOM

- O Document Object Model do JavaScript mapeia algumas estruturas do HTML a objetos (variáveis) da linguagem
  - Propriedades dos objetos (e conseqüentemente dos elementos da página) poderão ser alteradas em tempo de execução
  - Mapeamento restringe-se a elementos de formulário, vínculos, imagens e atributos da janela do browser.
  - Permite validação de campos dos formulários, cálculos locais, imagens dinâmicas, abertura de novas janelas, controle de frames, etc.
  - Não é completa. Não mapeia parágrafos, títulos ou folhas de estilo



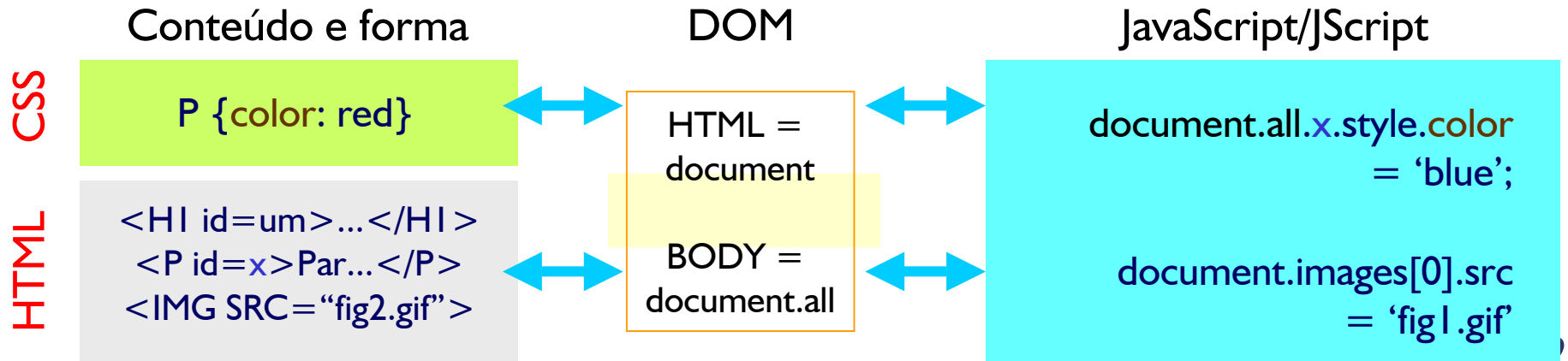
DOM do W3C

- *Document Object Model do W3C*
  - *Mapeia todos os elementos do HTML e folha de estilos, tornando-os acessíveis como objetos JavaScript*
- *Desvantagem: compatibilidade*
  - *A Microsoft utiliza DOM diferente da Netscape*
  - *A W3C tenta padronizar outro*
  - *Na prática W3C DOM funciona bem com XML, mas é problemático com HTML*



DHTML - Dynamic HTML

- *Combinação de uma linguagem de programação (geralmente JavaScript) com HTML e CSS*
  - *Permite tratar eventos em qualquer lugar da página*
  - *Permite grande interatividade*
  - *Permite alteração dinâmica de conteúdo, estrutura e aparência*
- *DOM - Document Object Model é a ponte entre o HTML/CSS e a linguagem baseada em objetos*



Tecnologias lado-servidor

- ***Estendem*** *as funções básicas de servidor HTTP:*
  - *CGI - Common Gateway Interface*
  - *APIs: ISAPI, NSAPI, Apache API, Servlet API, ...*
  - *Scripts: ASP, JSP, LiveWire (SSJS), Cold Fusion, PHP, ...*
- *Rodam do lado do servidor, portanto, não dependem de suporte por parte dos browsers*
  - *Browsers fornecem apenas a interface do usuário*
- *Interceptam o curso normal da comunicação*
  - *Recebem dados via requisições HTTP (GET e POST)*
  - *Devolvem dados através de respostas HTTP*

CGI - Common Gateway Interface

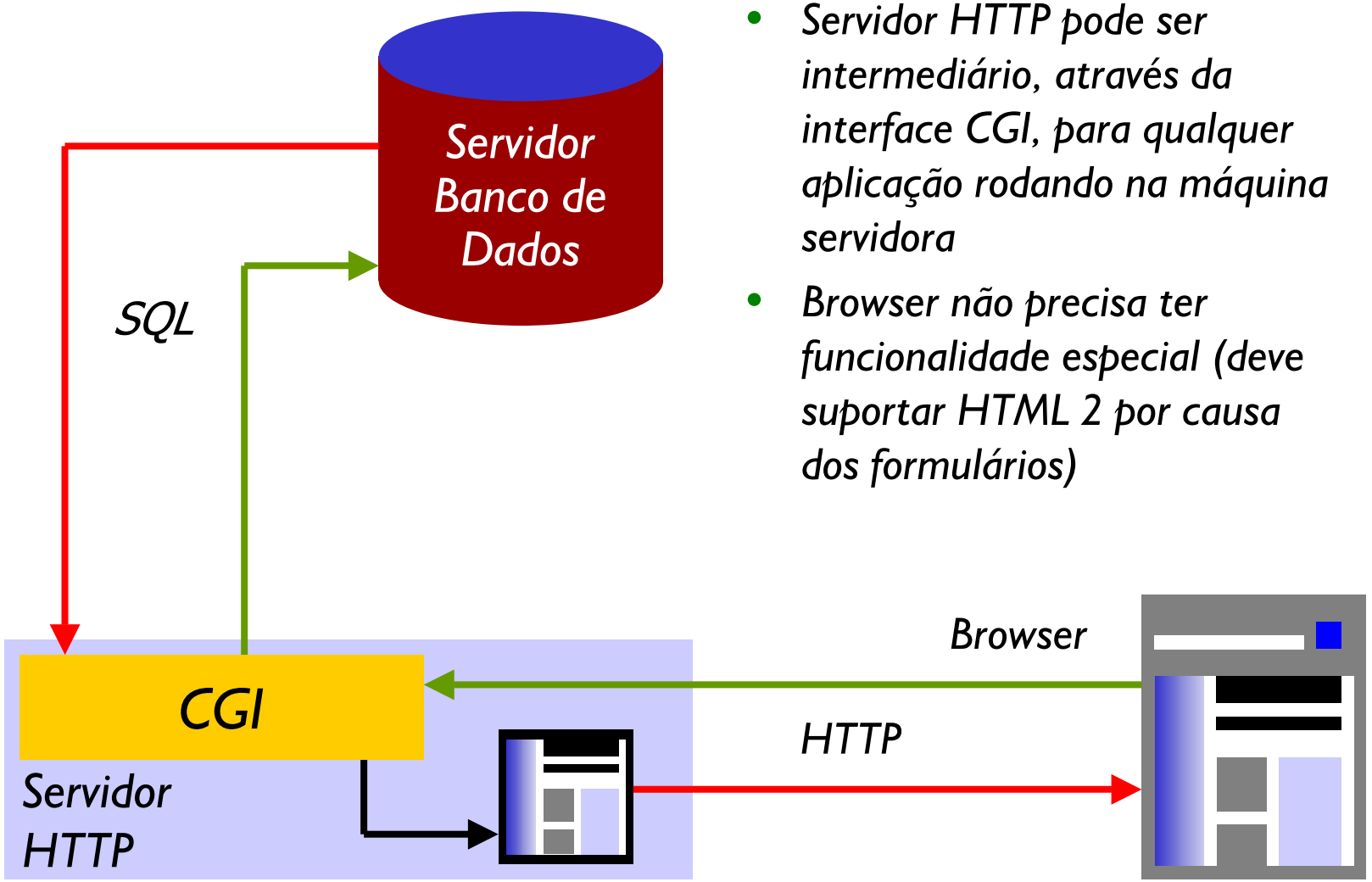
- **Especificação** que determina como construir uma aplicação que será executada pelo servidor Web
- Programas CGI podem ser escritos em qualquer linguagem de programação. A especificação limita-se a determinar os formatos de entrada e saída dos dados (HTTP).
- O que interessa é que o programa seja capaz de
  - Obter dados de entrada a partir de uma requisição HTTP
  - Gerar uma resposta HTTP incluindo os dados e parte do cabeçalho

- Escopo: camada do servidor
  - Não requer quaisquer funções adicionais do cliente ou do HTTP

Aplicações CGI

- Programas CGI podem ser escritos em qualquer linguagem. As linguagens mais populares são C e Perl.
- A linguagem usada deve ter facilidades para
  - Ler variáveis de ambiente (onde o servidor armazena informações passadas no cabeçalho da requisição).
  - Imprimir dados de 8 bits (e não apenas texto ASCII)
- Linguagens não recomendadas
  - Java: dificuldade de ler propriedades do sistema
  - MS-DOS: impossibilidade de gerar HTML
- Segurança depende do servidor e do código

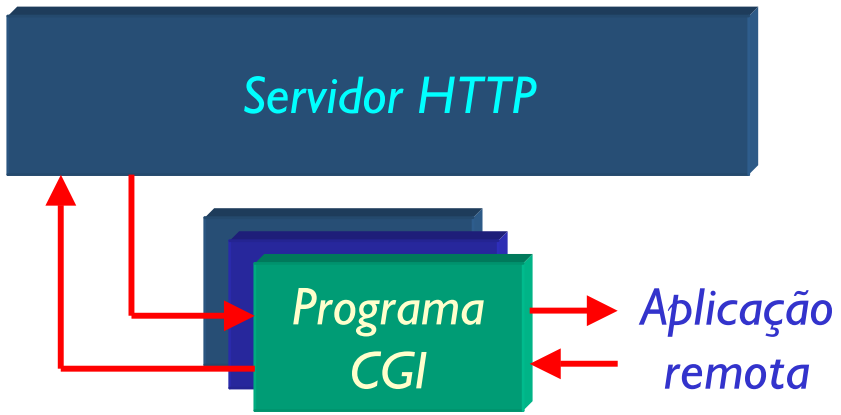
CGI: Gateway para aplicações



- Servidor HTTP pode ser intermediário, através da interface CGI, para qualquer aplicação rodando na máquina servidora
- Browser não precisa ter funcionalidade especial (deve suportar HTML 2 por causa dos formulários)

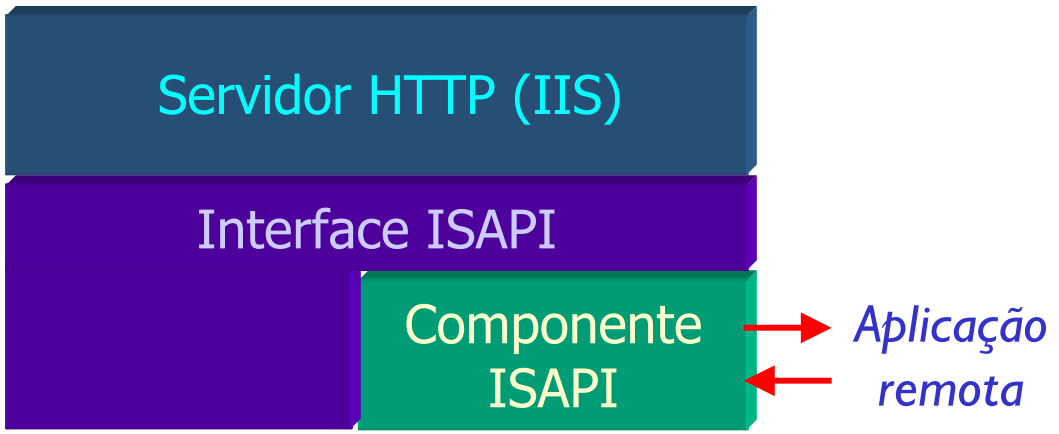
CGI é prático... Mas ineficiente!

- A interface CGI requer que o servidor sempre execute um programa
  - Um novo processo do S.O. rodando o programa CGI é criado para cada cliente remoto que o requisita.
  - Novos processos consomem muitos recursos, portanto, o desempenho do servidor diminui por cliente conectado.
- CGI roda como um processo externo, logo, não tem acesso a recursos do servidor
  - A comunicação com o servidor resume-se à entrada e saída.
  - É difícil o compartilhamento de dados entre processos

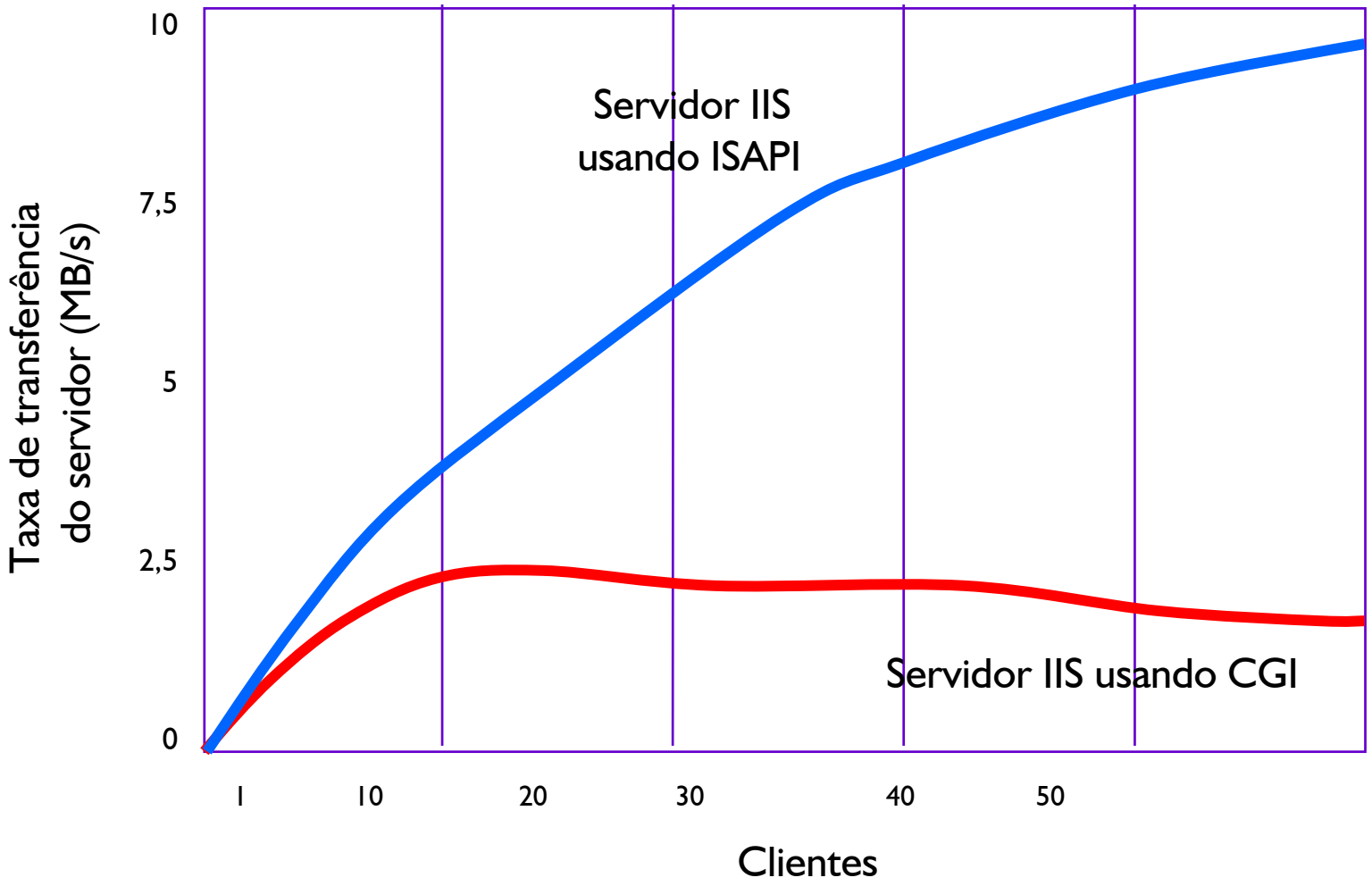


APIs do servidor

- *Podem substituir totalmente o CGI, com vantagens:*
  - *Toda a funcionalidade do servidor pode ser usada*
  - *Múltiplos clientes em processos internos (threads)*
  - *Muito mais rápidas e eficientes (menos overhead)*
- *Desvantagens:*
  - *Em geral dependem de plataforma, fabricante e linguagem*
  - *Soluções proprietárias*
- *Exemplos*
  - *ISAPI (Microsoft)*
  - *NSAPI (Netscape)*
  - *Apache Server API*
  - *??SAPI*



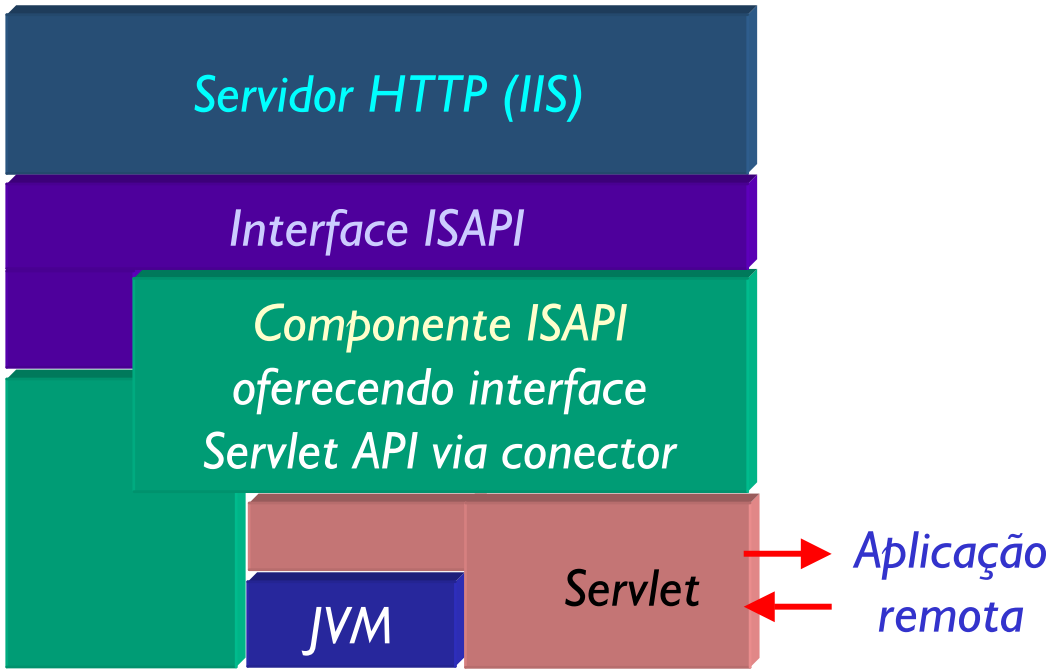
ISAPI vs. CGI



Teste realizado pela PCMagazine Labs com aplicações similares

Servlet API

- *API independente de plataforma e praticamente independente de fabricante*
- *Componentes são escritos em Java e se chamam servlets*
- *Como os componentes SAPI proprietários, rodam dentro do servidor, mas através de uma Máquina Virtual Java*
- *Disponível como 'plug-in' ou conector para servidores que não o suportam diretamente*
  - *Desenho ao lado mostra solução antiga de conexão com IIS*
- *Nativo em servidores Sun, IBM, ...*



Vantagens dos servlets...

- ... sobre CGI
  - Rodam como parte do servidor (cada nova requisição inicia um novo thread mas não um novo processo)
  - Mais integrados ao servidor: mais facilidade para compartilhar informações, recuperar e decodificar dados enviados pelo cliente, etc.
- ... sobre APIs proprietárias
  - Não dependem de único servidor ou sistema operacional
  - Têm toda a API Java à disposição (JDBC, RMI, etc.)
  - Não comprometem a estabilidade do servidor em caso de falha (na pior hipótese, um erro poderia derrubar o JVM)

Problemas dos servlets, CGI e APIs

- *Para gerar páginas dinâmicas (99% das aplicações), é preciso embutir o HTML ou XML dentro de instruções de uma linguagem de programação:*

```
out.print("<h1>Servlet</h1>");
for (int num = 1; num <= 5; i++) {
    out.print("<p>Parágrafo " + num + "</p>");
}
out.print("<table><tr><td> ... </tr></table>");
```

- *Maior parte da informação da página é estática, no entanto, precisa ser embutida no código*
- *Afasta o Web designer do processo*
  - *Muito mais complicado programar que usar HTML e JavaScript*
  - *O design de páginas geradas dinamicamente acaba ficando nas mãos do programador (e não do Web designer)*

Solução: scripts de servidor

- *Coloca a linguagem de programação dentro do HTML (e não o contrário)*

```
<h1>Servlet</h1>
 <% for (int num = 1; num <= 5; i++) { %>
   <p>Parágrafo <%= num %></p>
 <%}%>
 <table><tr><td> ... </tr></table>
```

- *Permite o controle da aparência e estrutura da página em softwares de design (DreamWeaver, FrontPage)*
- *Página fica mais legível*
- *Quando houver muita programação, código pode ser escondido em servlets, JavaBeans, componentes (por exemplo: componentes ActiveX, no caso do ASP)*

Scripts de servidor

- Alguns dos mais populares:
  - Microsoft Active Server Pages (ASP)
  - Sun JavaServer Pages (JSP)
  - Macromedia Cold Fusion
  - PHP
- A página geralmente possui uma extensão de nome de arquivo diferente para que o servidor a identifique como um programa
- As página ASP, PHP, JSP, etc. são processadas e os roteiros são executados pelo servidor, que os consome
  - No browser, chega apenas a saída do programa: página HTML
  - Comandos <% .. %> ou similares nunca chegam no browser
  - Servidor envia cabeçalho Content-type: text/html (default) ou algum outro tipo texto (text/xml, text/plain)

Controle de sessão

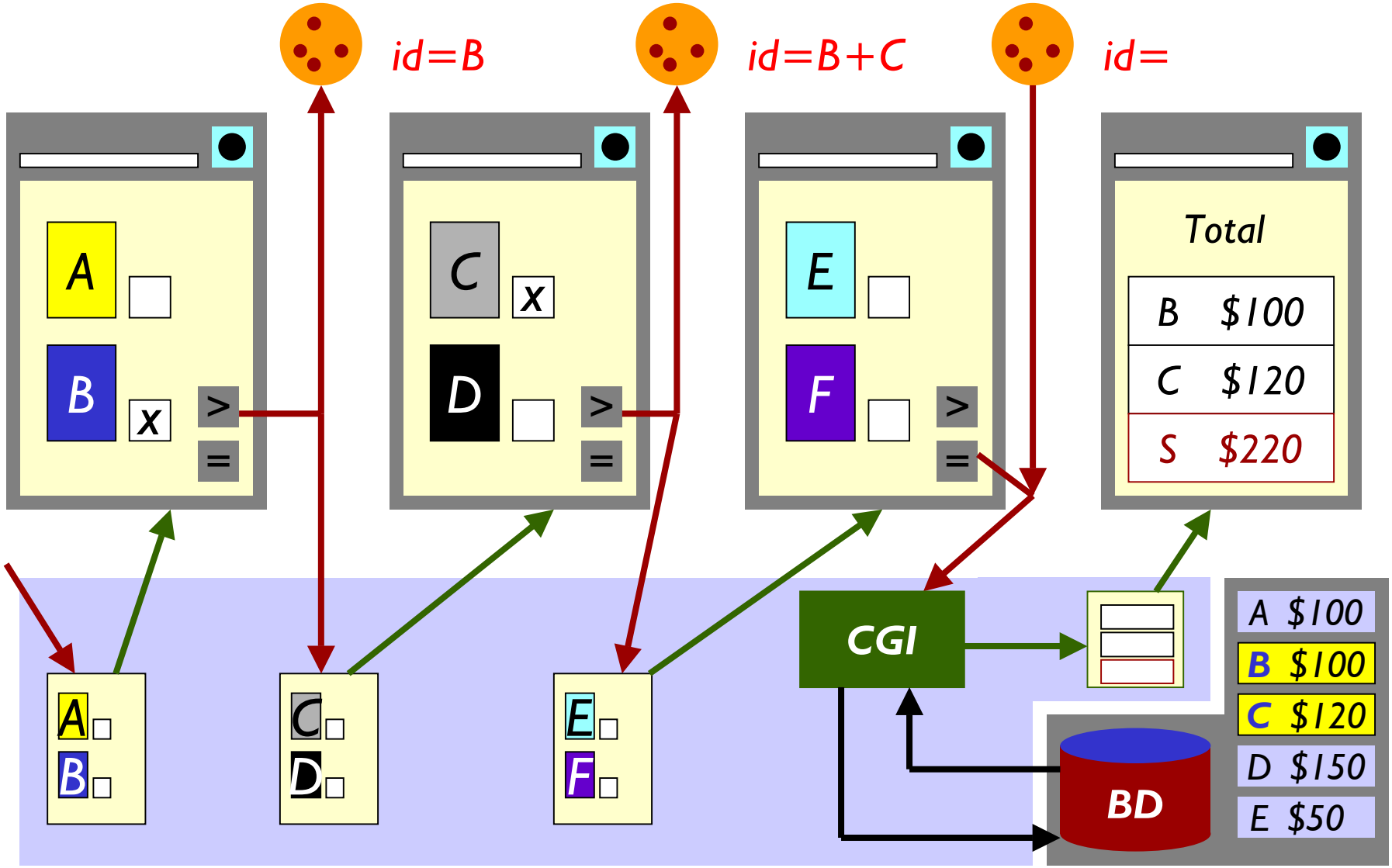
- *HTTP não preserva o estado de uma sessão. É preciso usar mecanismos artificiais com CGI (ou qualquer outra tecnologia Web)*
  - *Seqüência de páginas/aplicações: desvantagens: seqüência não pode ser quebrada; mesmo que página só contenha HTML simples, precisará ser gerada por aplicação*
  - *Inclusão de dados na URL: desvantagens: pouca flexibilidade e exposição de informações*
  - *Cookies (informação armazenada no cliente): desvantagens: espaço e quantidade de dados reduzidos; browser precisa suportar a tecnologia*

Cookies

- *Padrão Internet (RFC) para persistência de informações entre requisições HTTP*
- *Um cookie é uma pequena quantidade de informação que o servidor armazena no cliente*
  - *Par nome=valor. Exemplos: usuario=paulo, num=123*
  - *Escopo no servidor: domínio e caminho da página*
  - *Pode ser seguro*
  - *Escopo no cliente: browser (sessão)*
  - *Duração: uma sessão ou tempo determinado (cookies persistentes)*
- *Cookies são criados através de cabeçalhos HTTP*

```
Content-type: text/html
Content-length: 34432
Set-Cookie: usuario=ax343
Set-Cookie: lastlogin=12%2610%2699
```

Exemplo com cookies: Loja virtual



> Guarda cookie e chama próxima página

= Lê cookie, apaga-o e envia dados para CGI

Exercícios

- *1. Conecte-se via Telnet na porta HTTP de um servidor conhecido. Ex: telnet servidor 80*
- *2. Envie o comando GET abaixo, digite <ENTER> duas vezes e veja o resultado*

```
GET / HTTP/1.0
```

- *3. Envie o comando POST abaixo. Observe o momento do envio*

```
POST /servlet/TestPost HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 10

abcde01234
```

- *4. Rode o programa SendGet.class*
- *3. Execute o programa Listen.class (veja cap01/) em uma janela. Ele escuta a porta 8088. Conecte-se via browser à porta 8088 usando http://localhost:8088*
  - *O programa imprime a requisição recebida pelo browser*

Aplicaçoes Web e Java

- *Servlets e JavaServer Pages (JSP) são as soluções Java para estender o servidor HTTP*
  - *Suportam os métodos de requisição padrão HTTP (GET, POST, HEAD,PUT, DELETE, OPTIONS, TRACE)*
  - *Geram respostas compatíveis com HTTP (códigos de status, cabeçalhos RFC 822)*
  - *Interagem com Cookies*
- *Além dessas tarefas básicas, também*
  - *Suportam filtros, que podem ser chamados em cascata para tratamento de dados durante a transferência*
  - *Suportam controle de sessão transparentemente através de cookies ou rescrita de URLs (automática)*
- *É preciso usar um servidor que suporte as especificações de servlets e JSP*

Exemplo de um servlet

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

public class SimpleServlet extends HttpServlet {
    public void doGet (HttpServletRequest  request,
                       HttpServletResponse response)
                          throws ServletException, IOException {
       PrintWriter  out;
       response.setContentType("text/html");
       out = response.getWriter();
       String user = request.getParameter("usuario");
       if (user == null)
           user = "World";

       out.println("<HTML><HEAD><TITLE>");
       out.println("Simple Servlet Output");
       out.println("</TITLE></HEAD><BODY>");
       out.println("<H1>Simple Servlet Output</H1>");
       out.println("<P>Hello, " + user);
       out.println("</BODY></HTML>");
       out.close();
    }
}
```

Exemplo de um JSP equivalente

```
<HTML><HEAD>
<TITLE>Simple Servlet Output</TITLE>
</HEAD><BODY>
<%
      String user =
         request.getParameter("usuario");
      if (user == null)
          user = "World";
%>
<H1>Simple Servlet Output</H1>
<P>Hello, <%= user %>
</BODY></HTML>
```

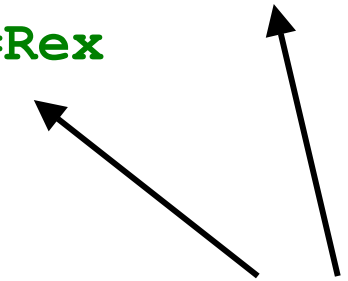
Página recebida no browser

- *Url da requisição*

```
http://servidor/servlet/SimpleServlet?usuario=Rex
http://servidor/hello.jsp?usuario=Rex
```

- *Código fonte visto no cliente*

```
<HTML><HEAD>
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, Rex
</BODY></HTML>
```



Usando contexto default ROOT no TOMCAT

Um simples JavaBean

```
package beans;

public class HelloBean implements
                        java.io.Serializable {

      private String msg;

      public HelloBean() {
            this.msg = "World";
      }

      public String getMensagem() {
            return msg;
      }

      public void setMensagem(String msg) {
            this.msg = msg;
      }
}
```

JSP usando JavaBeans

- *Página JSP que usa HelloBean.class*

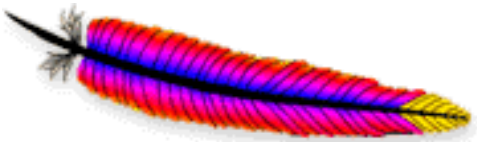
```
<HTML><HEAD>
<jsp:useBean id="hello" class="beans.HelloBean" />
<jsp:setProperty name="hello" property="mensagem"
                 param="usuario" />
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, <jsp:getProperty name="hello"
                           property="mensagem" />
</BODY></HTML>
```

Como executar servlets e JSP

- Para executar servlets e JSP é preciso implantá-los em um Web Container
- Um Web Container pode estar executando como parte de um servidor HTTP que o repassa as requisições destinadas a servlets e JSP
- Neste curso, usaremos o Tomcat Web Container, que pode tanto funcionar conectado a outro servidor como usar seu próprio servidor Web
- O Tomcat ocupará a porta 8080

Jakarta Tomcat

- O Apache Jakarta Tomcat é a implementação de referência para aplicações Web
  - Tomcat 3.x - I.R. para servlets 2.2 e JSP 1.1
  - Tomcat 4.x - I.R. para servlets 2.3 e JSP 1.2
- Em produção, geralmente é acoplado a um servidor de páginas estáticas eficiente (Apache, ou outro)
- Em desenvolvimento, pode-se usar o servidor distribuído com o Tomcat
- Instale o Tomcat
  - Use c:\tomcat-4.0 ou /usr/jakarta/tomcat e defina as variáveis CATALINA_HOME e TOMCAT_HOME apontando para o mesmo lugar



The Jakarta Project
http://jakarta.apache.org

Estrutura do Tomcat

bin	Executáveis. Para iniciar o Tomcat, rode startup.bat
webapps	Contém pastas de contextos (aplicações Web)
└ ROOT	Contexto raiz (coloque suas páginas Web aqui)
common	Classpaths do Tomcat (valem para todas as aplicações)
├ classes	Classpath (coloque classes aqui)
└ lib	Classpath de JARs (o servlet.jar está aqui)
classes	Classpath do servidor (não use)
lib	Classpath do servidor para JARs (não use)
server	Executáveis do Tomcat
conf	Arquivos de configuração (server.xml e outros)
logs	Logs para todas as aplicações
work	Contém servlets gerados a partir de JSPs
temp	Diretório temporário

- $TOMCAT_HOME/conf/server.xml: configuração do servidor (onde se pode configurar novos contextos)
- $TOMCAT_HOME/common/lib/*.jar: Classpath para todas as aplicações que rodam no container (use com cuidado para evitar conflitos)

Como iniciar e parar o Tomcat

- No diretório bin/ há vários arquivos executáveis
- Para iniciar o Tomcat, use, a partir do bin/
  - ./startup.sh ou
  - startup.bat
- Para encerrar o Tomcat, use
  - ./shutdown.sh ou
  - shutdown.bat
- Crie atalhos na sua área de trabalho para esses arquivos ou para o diretório bin (se eles já não existirem). Eles serão usados freqüentemente
- Refira-se aos arquivos no diretório logs/ para realizar depuração de suas aplicações. Crie um atalho para ele também

Como implantar uma aplicação no Tomcat

- Há três maneiras
  - Transferir os arquivos da aplicação (JSP, servlets) para contextos já reconhecidos pelo servidor
  - Configurar o servidor para que reconheça um novo contexto onde os arquivos da aplicação residem (server.xml)
  - Implantar a aplicação como um WebArchive (WAR)
- Contextos são diretórios devidamente configurados que o Tomcat reconhece como aplicações Web
- O contexto raiz chama-se ROOT.
  - Arquivos copiados para `$TOMCAT_HOME/webapps/ROOT/` podem ser acessados via `http://servidor:8080/`
  - Servlets em `webapps/ROOT/WEB-INF/classes` podem ser acessados via `http://servidor:8080/servlet/`

Outros contextos existentes

- Os exemplos do Tomcat rodam em um contexto diferente de ROOT: no contexto /examples/
- Para usar /examples:
  - Coloque páginas Web, JSPs, imagens, etc. em `$TOMCAT_HOME/webapps/examples/`
  - Coloque beans, classes e servlets em `$TOMCAT_HOME/webapps/examples/WEB-INF/classes/`
- Acesse as páginas e JSP usando:
  - http://servidor/examples/pagina.html
- Acesse os servlets usando
  - http://servidor/examples/servlet/pacote.Classe
- Não precisa reiniciar o servidor

Exercícios

- 1. Copie o arquivo hello.jsp para o contexto ROOT
  - a) Copie para `$CATALINA_HOME/webapps/ROOT` (`%CATALINA_HOME%` aponta para `c:\tomcat-4.0` na nossa instalação Windows)
  - b) Acesse via `http://localhost:8080/hello.jsp`
- 2. Implante o SimpleServlet no servidor
  - a) Compile usando o servlet.jar encontrável em common/lib e copie para webapps/ROOT/WEB-INF/classes
  - b) Se classes não existir, crie o diretório e reinicie o servidor

```
javac -d $CATALINA_HOME/webapps/ROOT/WEB-INF/classes
      -classpath  $CATALINA_HOME/common/lib/servlet.jar
      SimpleServlet.java
```

  - c) Acesse via `http://localhost:8080/servlet/SimpleServlet`

Exercícios (2)

- 3. Implante a aplicação hellobean.jsp. Ela tem duas partes: uma página JSP e uma classe Java (JavaBean)
  - a) Compile o JavaBean (observe que ele está em um pacote) e copie-o para o `webapps/ROOT/WEB-INF/classes`
    `src/` contém diretório `beans/` (pacote), que contém `HelloBean.java`

```
> cd src
> javac -d $CATALINA_HOME/webapps/ROOT/WEB-INF/classes
        beans/HelloBean.java
```

  - b) Copie `hellobean.jsp` para `webapps/ROOT`
  - c) Acesse via `http://localhost:8080/hellobean.jsp`

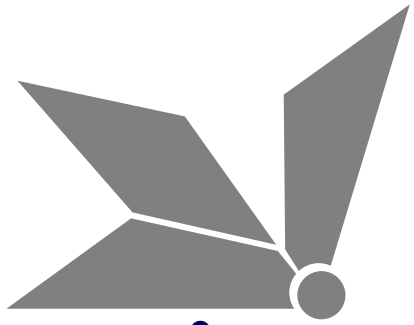
Arquiteturas de aplicações Web

- Grandes aplicações Web geralmente consistem de várias páginas JSP, HTML, imagens misturadas com classes Java comuns e servlets
- Procura-se separar responsabilidades
  - Controle de requisição, resposta, repasse de dados
  - Processamento de lógica de negócio
  - Processamento de resposta e geração de páginas
- Aplicações que dividem-se em camadas de acordo com as responsabilidades acima são aplicações MVC (Model-View-Controller)
  - Mais fáceis de manter e de reutilizar por equipes heterogêneas (web designers, programadores, etc.)

J2EE

- Java 2 Enterprise Edition é uma especificação que inclui JSP e servlets
- J2EE define uma arquitetura em camadas independentes formadas por componentes reutilizáveis
  - Páginas HTML ou outras tecnologias no cliente
  - Servlets e JSP na camada do servidor Web
  - Enterprise JavaBeans na camada de negócios
  - Conectores na camada de integração
  - Sistemas de informação na camada de dados
- Servlets e JSP podem ser usados em servidores J2EE
  - Precisam aderir a novas restrições do ambiente

helder@acm.org



argonavis.com.br

Servlets 2

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Sobre este módulo

- Neste módulo serão apresentados os fundamentos de servlets
  - Como escrever um servlet
  - Como compilar um servlet
  - Como implantar um servlet no servidor
  - Como executar
- Também serão exploradas as formas de interação do servlet com a requisição e resposta HTTP
  - Como ler parâmetros da entrada
  - Como gerar uma página de resposta
  - Como extrair dados de um formulário HTML

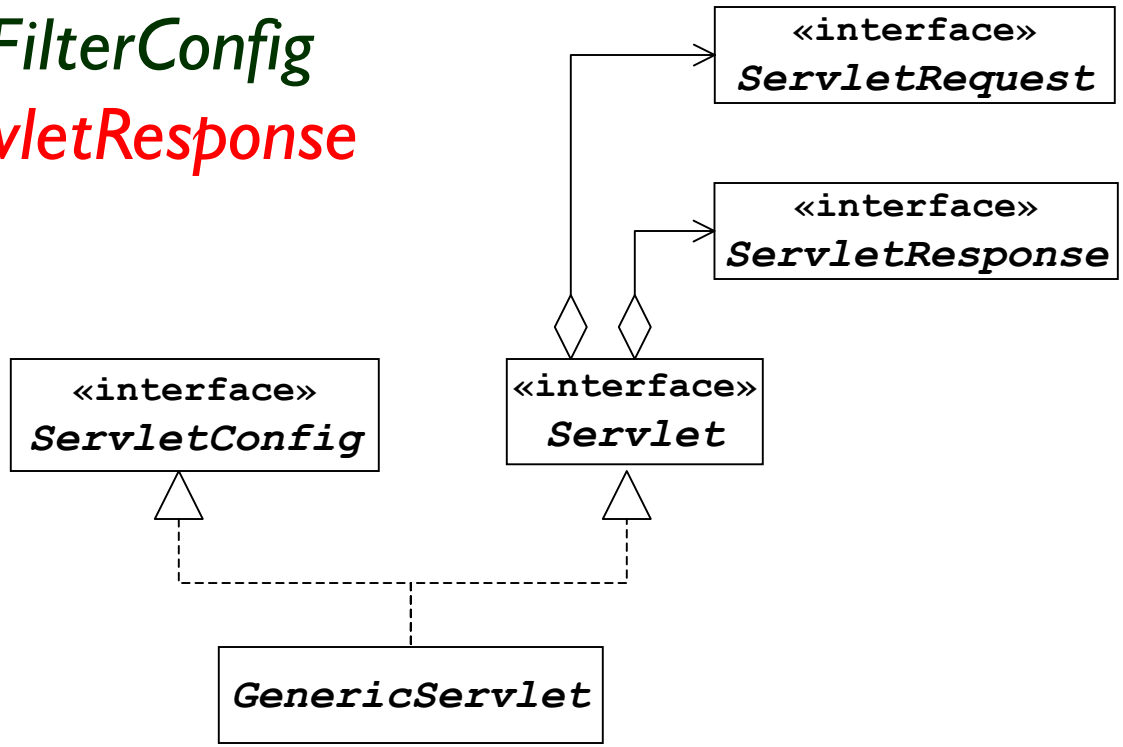
O que são servlets

- Extensão de servidor escrita em Java
  - Servlets são "applets" (pequenas aplicações) de servidor
  - Podem ser usados para estender qualquer tipo de aplicação do modelo requisição-resposta
  - Todo servlet implementa a interface javax.servlet.Servlet (tipicamente estende GenericServlet)
- Servlets HTTP
  - Extensões para servidores Web
  - Estendem javax.servlet.http.HttpServlet
  - Lidam com características típicas do HTTP como métodos GET, POST, Cookies, etc.

API: Servlets

Principais classes e interfaces de javax.servlet

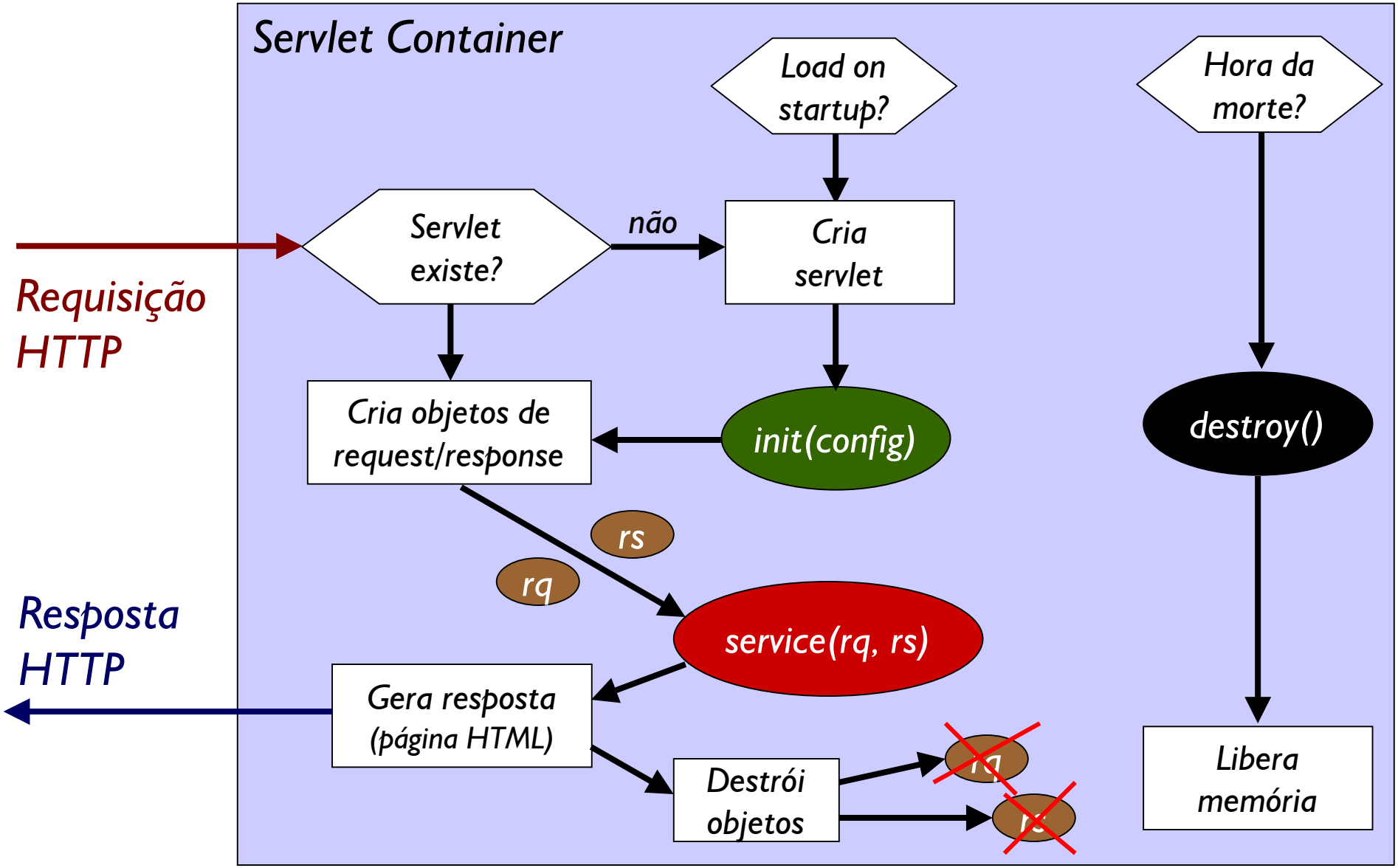
- *Interfaces*
  - *Servlet, ServletConfig, ServletContext*
  - *Filter, FilterChain, FilterConfig*
  - *ServletRequest, ServletResponse*
  - *SingleThreadModel*
  - *RequestDispatcher*
- *Classes abstratas*
  - *GenericServlet*
- *Classes concretas*
  - *ServletException*
  - *UnavailableException*
  - *ServletInputStream e ServletOutputStream*



Ciclo de vida

- O ciclo de vida de um servlet é controlado pelo container
- Quando o servidor recebe uma requisição, ela é repassada para o container que a delega a um servlet. O container
  1. Carrega a classe na memória
  2. Cria uma instância da classe do servlet
  3. Inicializa a instância chamando o método init()
- Depois aue o servlet foi inicializado, cada requisição é executada em um método service()
  - O container cria um objeto de requisição (ServletRequest) e de resposta (ServletResponse) e depois chama service() passando os objetos como parâmetros
  - Quando a resposta é enviada, os objetos são destruídos
- Quando o container decidir remover o servlet da memória, ele o finaliza chamando destroy()

Ciclo de vida



Como escrever um Servlet genérico

- *Um servlet genérico deve estender GenericServlet e implementar seu método service()*

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

public class Generico extends GenericServlet {

   public void service (ServletRequest request,
                        ServletResponse response)
                           throws IOException {

      PrintWriter out = response.getWriter();
      out.println("Hello, World!");
      out.close();
   }

}
```

Inicialização de um servlet

- *Tarefa realizada uma vez*
- *Deve-se sobrepor init(config) com instruções que serão realizadas para inicializar um servlet*
  - *Carregar parâmetros de inicialização, dados de configuração*
  - *Obter outros recursos*
- *Falha na inicialização deve provocar UnavailableException (subclasse de ServletException)*

```
public void init(ServletConfig config)
                              throws ServletException {
    String dirImagens =
                config.getInitParameter("imagens");
    if (dirImagens == null) {
        throw new UnavailableException
                          ("Configuração incorreta!");
    }
}
```

Finalização

- *Quando um servlet container decide remover um servlet da memória, ele chama o seu método destroy()*
  - *Use destroy() para liberar recursos (como conexões de banco de dados, por exemplo) e fazer outras tarefas de "limpeza".*

```
public void destroy() {
   banco.close();
   banco = null;
}
```

- *O servlet geralmente só é destruído quando todos os seus métodos service() terminaram (ou depois de um timeout)*
  - *Se sua aplicação tem métodos service() que demoram para terminar, você deve garantir um shutdown limpo.*

Métodos de serviço

- São os métodos que implementam operações de resposta executadas quando o cliente envia uma requisição
- Todos os métodos de serviço recebem dois parâmetros: um objeto ServletRequest e outro ServletResponse
- Tarefas usuais de um método de serviço
  - extrair informações da requisição
  - acessar recursos externos
  - preencher a resposta (no caso de HTTP isto consiste de preencher os cabeçalhos de resposta, obter um stream de resposta e escrever os dados no stream)

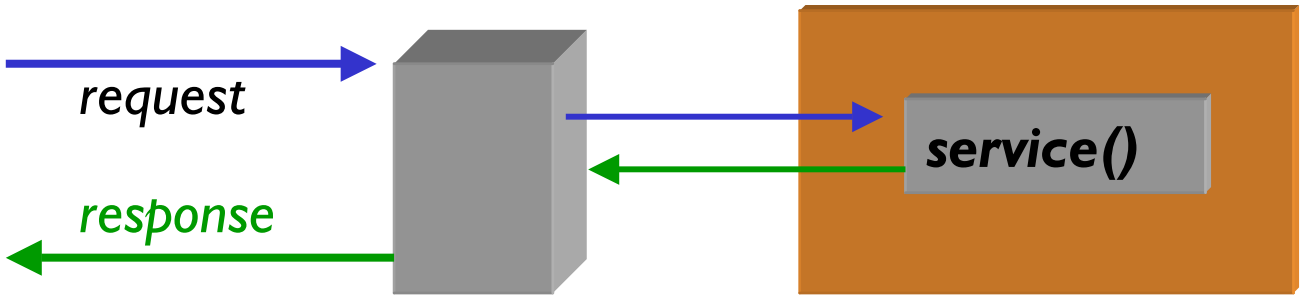
Métodos de serviço (2)

- *O método de serviço de um servlet genérico é o método abstrato service()*

```
public void service(ServletRequest, ServletResponse)
```

definido em javax.servlet.Servlet.

- *Sempre que um servidor repassar uma requisição a um servlet, ele chamará o método service(request, response).*



- *Um servlet genérico deverá sobrepor este método e utilizar os objetos ServletRequest e ServletResponse recebidos para ler os dados da requisição e compor os dados da resposta, respectivamente*

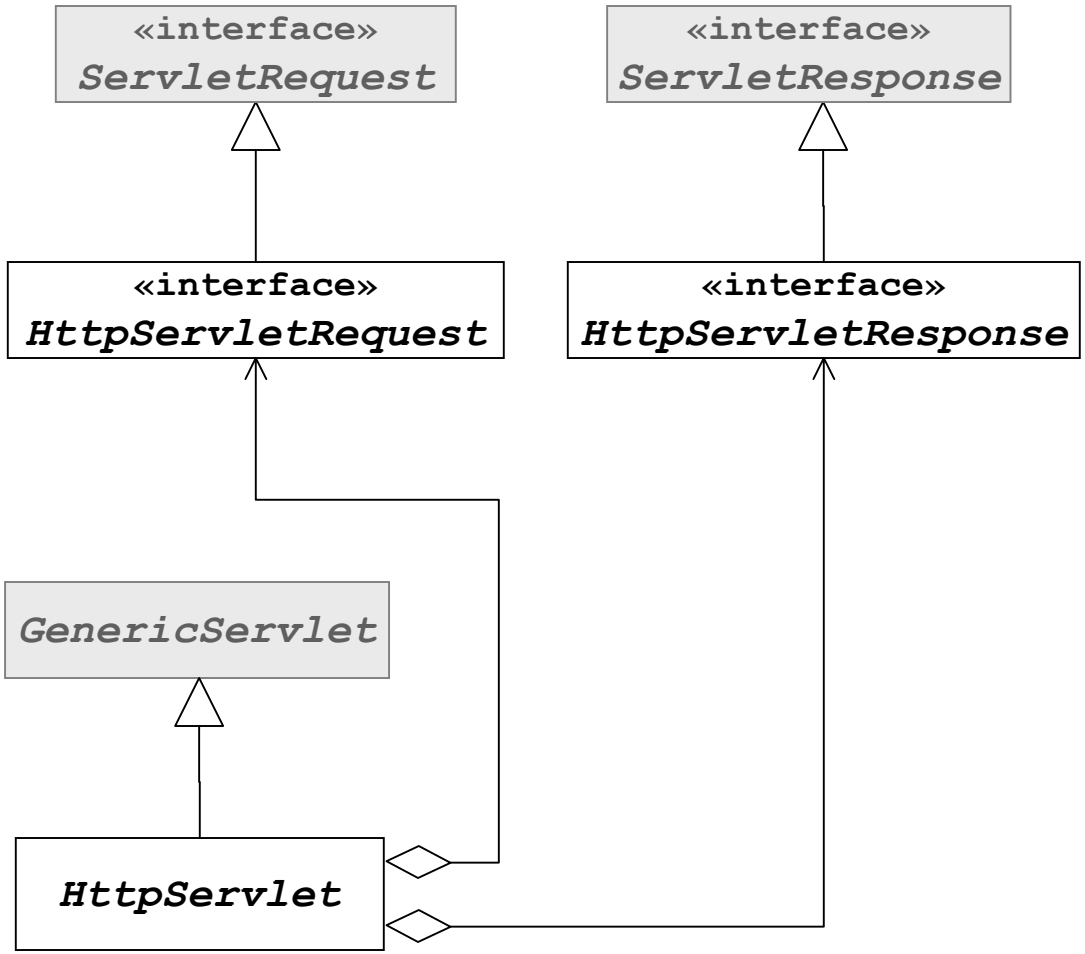
Servlets genéricos

- Servlets genéricos servem como componentes para serviços tipo requisição-resposta em geral
  - Não se limitam a serviços HTTP
  - Podem ser usados para estender, com componentes reutilizáveis, um serviço existente: é preciso implementar um "container" para rodar o servlet
- Para serviços Web deve-se usar Servlets HTTP
  - API criada especificamente para lidar com características próprias do HTTP
  - Método service() dividido em métodos específicos para tratar os diferentes métodos do HTTP

API: Servlets HTTP

Classes e interfaces mais importantes do pacote javax.servlet.http

- *Interfaces*
  - *HttpServletRequest*
  - *HttpServletResponse*
  - *HttpSession*
- *Classes abstratas*
  - *HttpServlet*
- *Classes concretas*
  - *Cookie*



Como escrever um servlet HTTP

- *Para escrever um servlet HTTP, deve-se estender HttpServlet e implementar um ou mais de seus métodos de serviço, tipicamente: doPost() e/ou doGet()*

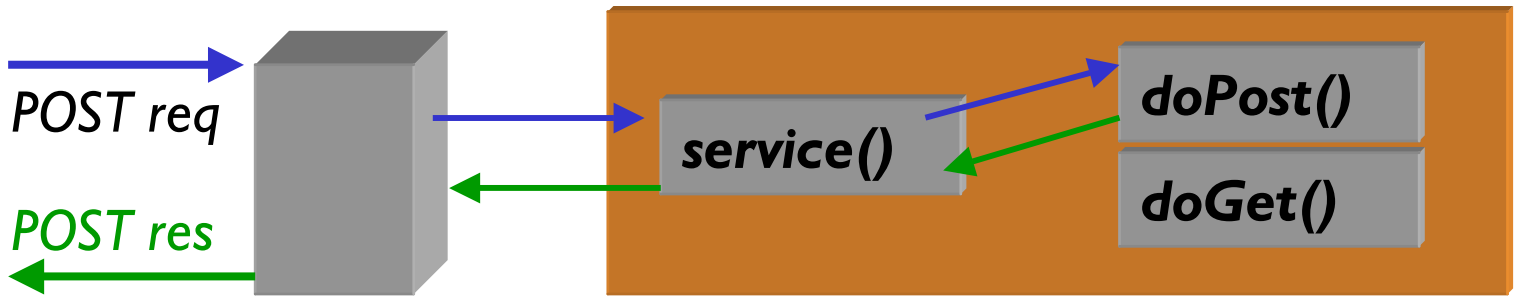
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ServletWeb extends HttpServlet {
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
                           throws IOException {

      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      out.println("<h1>Hello, World!</h1>");
      out.close();
   }
}
```

Métodos de serviço HTTP

- *A classe HttpServlet redireciona os pedidos encaminhados para service() para métodos que refletem os métodos HTTP (GET, POST, etc.):*
  - *public void doGet(HttpServletRequest, HttpServletResponse)*
  - *public void doPost(HttpServletRequest, HttpServletResponse)*
  - *… **



- *Um servlet HTTP genérico deverá estender HTTPServlet e implementar pelo menos um dos métodos doGet() ou doPost()*

** doDelete(), doTrace(), doPut(), doOptions() - Método HEAD é implementado em doGet()*

Inicialização

- *A inicialização de um GenericServlet, como o HttpServlet, pode (e deve) ser feita com a versão de init() sem argumentos (e não init(config))*
- *Todos os métodos de config estão no servlet, pois GenericServlet implementa ServletConfig*

```
public void init() throws ServletException {
    String dirImagens =
                getInitParameter("imagens");
    if (dirImagens == null) {
        throw new UnavailableException
                        ("Configuração incorreta!");
    }
}
```

A requisição HTTP

- *Uma requisição HTTP feita pelo browser tipicamente contém vários cabeçalhos RFC822**

```
GET /docs/index.html HTTP/1.0
Connection: Keep-Alive
Host: localhost:8080
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
Accept: image/gif, image/x-bitmap, image/jpg, image/png, */*
Accept-Charset: iso-8859-1, *
Cookies: jsessionid=G3472TS9382903
```

- *Os métodos de HttpServletRequest permitem extrair informações de qualquer um deles*
  - *Pode-se também identificar o método e URL*

* *Especificação de cabeçalho para e-mail*

Obtenção de dados de requisições

- Alguns métodos de HttpServletRequest
  - Enumeration getHeaderNames() - obtém nomes dos cabeçalhos
  - String getHeader("nome") - obtém primeiro valor do cabeçalho
  - Enumeration getHeaders("nome") - todos os valores do cabeçalho
  - String getParameter(param) - obtém parâmetro HTTP
  - String[] getParameterValues(param) - obtém parâmetros repetidos
  - Enumeration getParameterNames() - obtém nomes dos parâmetros
  - Cookie[] getCookies() - recebe cookies do cliente
  - HttpSession getSession() - retorna a sessão
  - setAttribute("nome", obj) - define um atributo obj chamado "nome".
  - Object getAttribute("nome") - recupera atributo chamado nome
  - String getRemoteUser() - obtém usuário remoto (se autenticado, caso contrátio devolve null)

A resposta HTTP

- *Uma resposta HTTP é enviada pelo servidor ao browser e contém informações sobre os dados anexados*

```
HTTP/1.0 200 OK
Content-type: text/html
Date: Mon, 7 Apr 2003 04:33:59 GMT-03
Server: Apache Tomcat/4.0.4 (HTTP/1.1 Connector)
Connection: close
Set-Cookie: jsessionid=G3472TS9382903

<HTML>
   <h1>Hello World!</h1>
</HTML>
```

- *Os métodos de HttpServletResponse permitem construir um cabeçalho*

Preenchimento de uma resposta

- Alguns métodos de HttpServletResponse
  - addHeader(String nome, String valor) - adiciona cabeçalho HTTP
  - setContentType(tipo MIME) - define o tipo MIME que será usado para gerar a saída (text/html, image/gif, etc.)
  - sendRedirect(String location) - envia informação de redirecionamento para o cliente (Location: url)
  - Writer getWriter() - obtém um Writer para gerar a saída. Ideal para saída de texto.
  - OutputStream getOutputStream() - obtém um OutputStream. Ideal para gerar formatos diferentes de texto (imagens, etc.)
  - addCookie(Cookie c) - adiciona um novo cookie
  - encodeURL(String url) - envia como anexo da URL a informação de identificador de sessão (sessionid)
  - reset() - limpa toda a saída inclusive os cabeçalhos
  - resetBuffer() - limpa toda a saída, exceto cabeçalhos

Como implementar doGet() e doPost()

- Use *doGet()* para receber requisições GET
  - Links clicados ou URL digitadas diretamente
  - Alguns formulários que usam GET
- Use *doPost()* para receber dados de formulários
- Se quiser usar ambos os métodos, não sobrepoha service() mas implemente tanto doGet() como doPost()

```
public class ServletWeb extends HttpServlet {
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response) {
       processar(request, response);
   }
   public void doPost (HttpServletRequest request,
                       HttpServletResponse response) {
       processar(request, response);
   }
   public void processar(HttpServletRequest request,
                         HttpServletResponse response) {
       ...
   }
}
```

Parâmetros da requisição

- Parâmetros são pares nome=valor que são enviados pelo cliente concatenados em strings separados por &:

```
nome=Jo%E3o+Grand%E3o&id=agente007&acesso=3
```

- Parâmetros podem ser passados na requisição de duas formas
  - Se o método for GET, os parâmetros são passados em uma única linha no query string, que estende a URL após um "?"

```
GET /servlet/Teste?id=agente007&acesso=3 HTTP/1.0
```

  - Se o método for POST, os parâmetros são passados como um stream no corpo na mensagem (o cabeçalho Content-length, presente em requisições POST informa o tamanho

```
POST /servlet/Teste HTTP/1.0
Content-length: 21
Content-type: x-www-form-urlencoded

id=agente007&acesso=3
```

Como ler parâmetros da requisição

- *Caracteres reservados e maiores que ASCII-7bit são codificados em URLs:*
  - *Ex:* `ã = %E3`
- *Formulários HTML codificam o texto ao enviar os dados automaticamente*
- *Seja o método POST ou GET, os valores dos parâmetros podem ser recuperados pelo método getParameter() de ServletRequest, que recebe seu nome*

```
String parametro = request.getParameter("nome");
```

- *Parâmetros de mesmo nome podem ser repetidos. Neste caso getParameter() retornará apenas a primeira ocorrência. Para obter todas use String[] getParameterValues()*

```
String[] params = request.getParameterValues("nome");
```

Como gerar uma resposta

- *Para gerar uma resposta, primeiro é necessário obter, do objeto HttpServletResponse, um fluxo de saída, que pode ser de caracteres (Writer) ou de bytes (OutputStream)*

```
Writer out = response.getWriter();    // ou
OutputStream out = response.getOutputStream();
```

- *Apenas um deve ser usado. Os objetos correspondem ao mesmo stream de dados*
- *Deve-se também definir o tipo de dados a ser gerado. Isto é importante para que o cabeçalho Content-type seja gerado corretamente e o browser saiba exibir as informações*

```
response.setContentType("text/html");
```

- *Depois, pode-se gerar os dados, imprimindo-os no objeto de saída (out) obtido anteriormente*

```
out.println("<h1>Hello</h1>");
```

Criação de servlets simples

- São necessárias quatro etapas para construir e usar um servlet
  - Codificar o servlet, usando a Servlet API
  - Compilar o servlet, usando o JAR que contém as classes da API (distribuído pelo software do Web Container)
  - Implantar o servlet no servidor (Web Container)
  - Executar o servlet, chamando-o pelo browser
- Code - Compile - Deploy - Run

Compilação e Implantação

- Para compilar, use qualquer distribuição da API
  - O servlet.jar distribuído pelo Tomcat em common/lib/
  - O j2ee.jar distribuído no pacote J2EE da Sun (em lib/)
  - O javax.servlet.jar do JBoss (server/default/lib/)
- Inclua o JAR no seu CLASSPATH ao compilar

```
> javac -classpath ../servlet.jar;. MeuServlet.java
```

- Para implantar, copie as classes compiladas para um contexto existente no servidor
  - Jakarta-Tomcat (webapps/ROOT/WEB-INF/classes)
  - JBoss: (server/default/deploy/)

Usando o Ant

- Pode-se usar o Ant para fazer a compilação e deployment de uma aplicação Web
  - Defina um <classpath> adicional nas tarefas <javac> que inclua o caminho do servlet.jar
  - Use <copy> para copiar os arquivos para os contextos corretos
  - Use <property environment="env" /> e as propriedades `env.TOMCAT_HOME` ou `env.CATALINA_HOME` para ler as variáveis de ambiente do sistema e tornar seu build file independente da localização do servidor
- O Ant também pode ser usado para gerar o JAR que encapsula uma aplicação Web (WAR) usando a tarefa <war>

Execução

- Se você instalou os servlets em um contexto raiz, execute-os através da URL
  - http://localhost:8080/servlet/nome.do.Servlet
- Se você instalou os servlets em outro contexto use
  - http://localhost:8080/contexto/servlet/nome.do.Servlet
- Para passar parâmetros
  - Escreva um formulário HTML, ou
  - Passe-os via URL, acrescentando um ? seguido dos pares nome=valor:

```
http://localhost:8080/servlet/Servlet?id=3&nome=Ze
```

Exercícios

- 1. Crie um servlet (j550.cap02.ParameterList) que imprima, em uma tabela, todos os nomes de parâmetros enviados e seus valores
  - O servlet deve suportar tanto GET como POST
  - Use request.getParameterNames() e getParameterValues(String)
  - Use o formulário de exemplo AllForms.html fornecido para testá-lo
- 2. Crie um servlet (j550.cap02.HeaderList) que imprima, em uma tabela, todos os nomes de cabeçalhos HTTP da requisição e seus valores
  - O servlet deve suportar tanto GET como POST
  - Use request.getHeaderNames() e getHeaders(String)

Exercícios (2)

- 3. Escreva um servlet simples (j550.cap02.HoraServlet) que devolva uma página contendo o dia, mês, ano e hora
  - Escreva e compile o servlet
  - Copie-o para $TOMCAT_HOME/webapps/ROOT/WEB-INF/classes
  - Rode-o no browser: http://localhost:8080/servlet/j550.HoraServlet
- 4. Escreva um servlet (j550.cap02.FatorialServlet) que gere uma tabela HTML com a lista de fatoriais entre 0 e 10
- 5. Altere o exercício 2 para que o servlet verifique a existência de um parâmetro "maximo". Se ele não for null, converta-o em int e use-o como limite para gerar a tabela de fatoriais.
  - Passe parâmetro com http://../j550.cap02.HoraServlet?maximo=5
- 6. Escreva um servlet que imprima formatado em uma tabela HTML o conteúdo do arquivo produtos.txt (j550.cap02.ProdutosServlet)

Formulários HTML

- *Todo formulário em HTML é construído usando elementos dentro de um bloco <FORM>*
- *O bloco <FORM> define a URL que receberá o formulário e pode definir também o método usado*

```
<FORM ACTION="URL para onde serão enviado os dados"
  METHOD="método HTTP (pode ser GET ou POST)"
  ENCTYPE="formato de codificação"
  TARGET="nome da janela que mostrará a resposta" >
    ... corpo do formulário
      (permite qualquer coisa permitida em <BODY>)
      ...
</FORM>
```

Formulários e links

- Formulários são similares a links.
- Um par formulário-botão tem o mesmo efeito que um link criado com <A HREF>
  - O link está no formulário e o evento no botão
- O bloco

```
<FORM ACTION="/dados/tutorial.html">
   <INPUT TYPE="submit" VALUE="Tutorial">
</FORM>
```

- gera a mesma requisição que

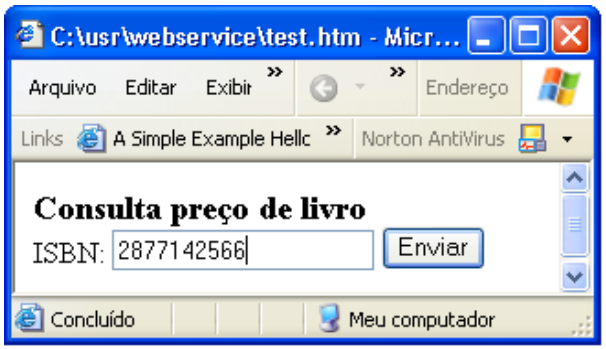
```
<A HREF="/dados/tutorial.html">Tutorial</A>
```

- que é

```
GET /dados/tutorial.html HTTP/1.0
```

Envio de dados com Formulários

- *Vários elementos HTML servem para entrada de dados e são usados dentro de formulários. Todos os elementos de entrada de dados têm um nome e enviam um valor*
- *Exemplo de formulário para entrada de dados*




```
<FORM ACTION="/cgi-bin/catalogo.pl"
      METHOD="POST">
 <H3>Consulta preço de livro</H3>
 <P>ISBN: <INPUT TYPE="text" NAME="isbn">
 <INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

Cabeçalho HTTP

Linha em branco

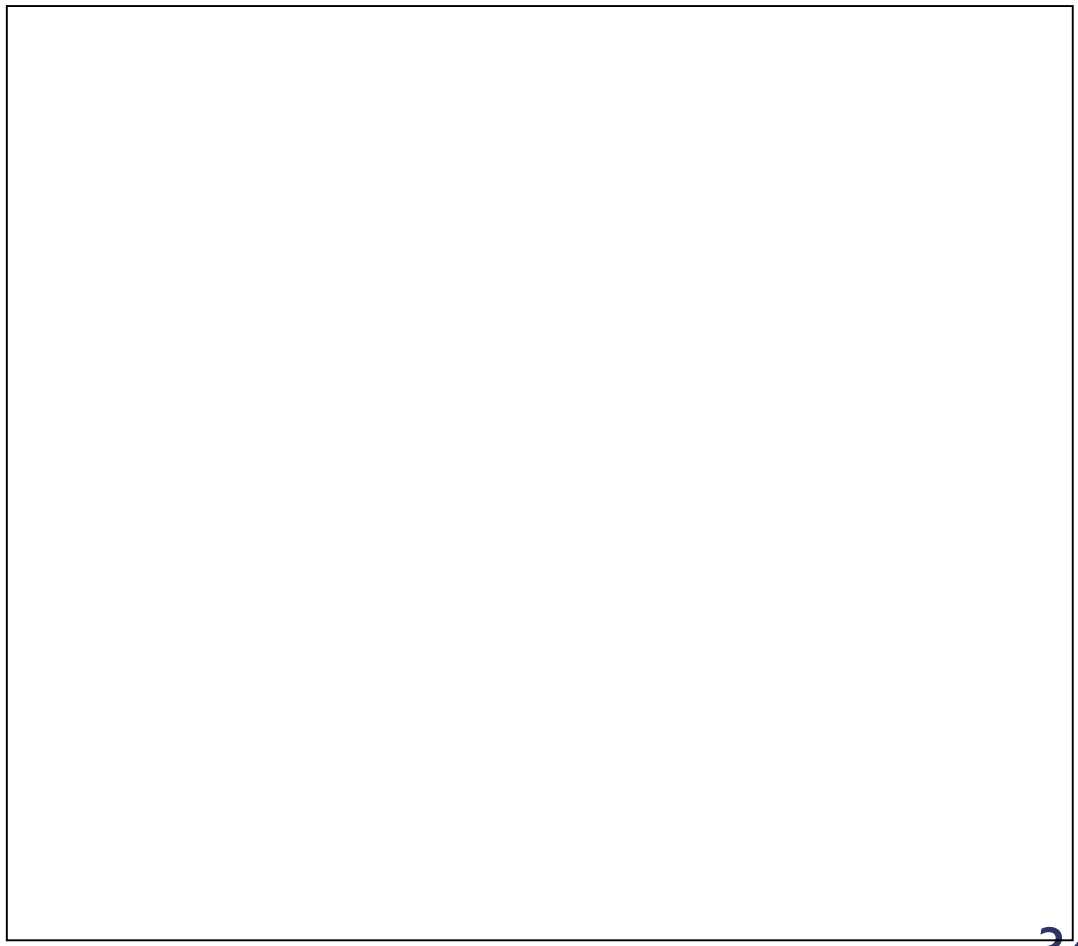
Mensagem (corpo da requisição)

```
POST /cgi-bin/catalogo.pl HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 15

isbn=2877142566
```

Elementos para disparo de eventos

- Os elementos <INPUT> com atributo TYPE Submit, Reset e Button servem para disparar eventos
  - Envio do formulário (Submit)
  - Reinicialização do formulário (Reset)
  - Evento programado por JavaScript (Button)
- O value do botão define o texto que mostrará
- Apenas se o botão contiver um atributo name, o conteúdo de value será enviado ao servidor



Entrada de texto

- Elementos <INPUT> com TYPE="text" podem ser usados para entrada de texto

- Com TYPE="password" o texto digitado é ocultado na tela do browser

Campos ocultos

- Campos ocultos consistem de um par nome/valor embutido no código HTML
- São úteis para que o autor da página possa enviar informações ao servidor
  - Informações sobre configuração da aplicação
  - Comandos, para selecionar comportamentos diferentes da aplicação
  - Parâmetros especiais para controle da aplicação, sessão ou dados que pertencem ao contexto da aplicação
- Sintaxe
  - `<INPUT TYPE="hidden" NAME="nome" VALUE="valor">`

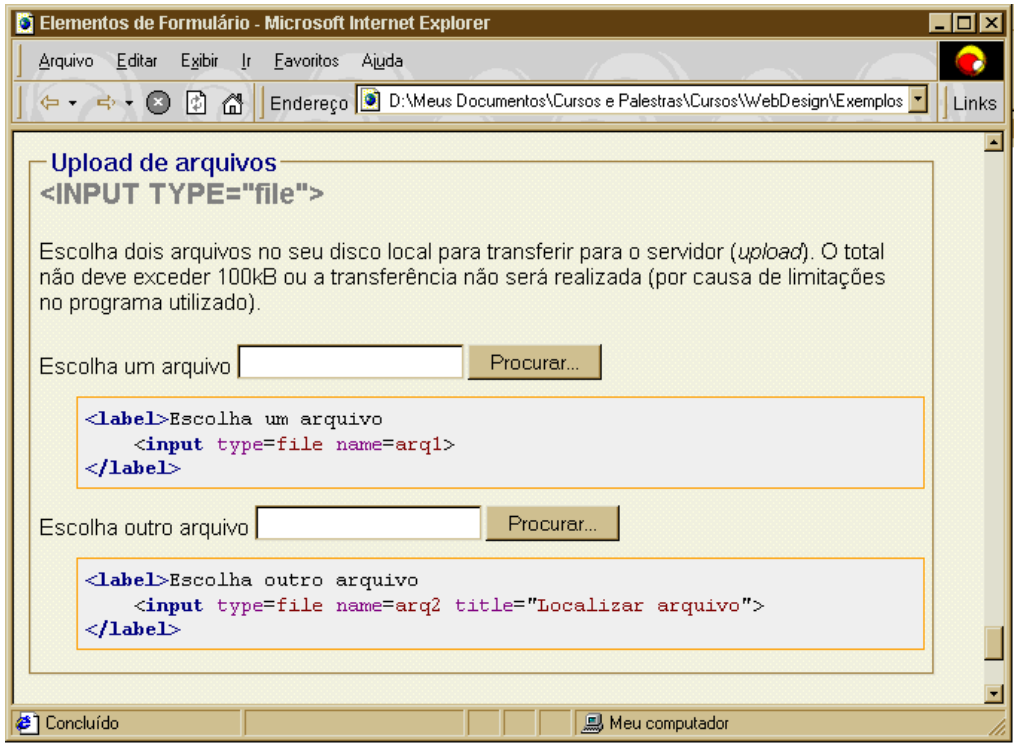
Chaves booleanas

- *Há dois tipos: checkboxes e radio buttons*
- *Checkboxes permitem mais de uma seleção*

- *O código acima enviará nomes repetidos contendo valores diferentes na requisição*
- *Radio Buttons, se tiverem o mesmo nome, formam um grupo. No grupo, apenas uma seleção é aceita*

Upload de arquivos

- *O elemento <INPUT TYPE="file"> cria uma tela que permite o Upload de arquivos para o servidor*
- *Formulário usado deve ter a seguinte sintaxe*




```
<FORM ACTION="/servlet/UploadServlet"
      METHOD="POST"
      ENCTYPE="text/multipart-form-data"> ... </FORM>
```

Área para entrada de texto

- Possibilitam a entrada de texto de múltiplas linhas
- Elemento: <TEXTAREA>

Menus de seleção

- Geram requisições similares a checkboxes e radio buttons
- Consistem de um par de elementos
  - <SELECT> define o nome da coleção
  - <OPTION> define o valor que será enviado

Exercicios

- *7. Crie um formulário onde uma pessoa possa digitar nome, telefone e e-mail. Faça-o chamar um servlet via POST que grave os dados em um arquivo.*
  - *a) Use o nome j550.cap02.RegistroServlet*
  - *b) Guarde um registro por linha*
  - *c) Separe cada campo com o caractere "|"*
  - *d) Use RandomAccessFile. Se o objeto se chamar, por exemplo, raf, use raf.writeUTF() para gravar no arquivo e raf.seek(raf.length()) para achar o final do arquivo*
  - *e) Faça com que a resposta do servlet seja o arquivo formatado em uma tabela HTML*
  - *f) Se o servlet for chamado diretamente (via GET), deve também mostrar os dados do arquivo formatados*
- *8. Embuta o formulário no próprio servlet, para que, após cada chamada, os dados e formulário sejam exibidos*

helder@acm.org

argonavis.com.br

Contextos

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

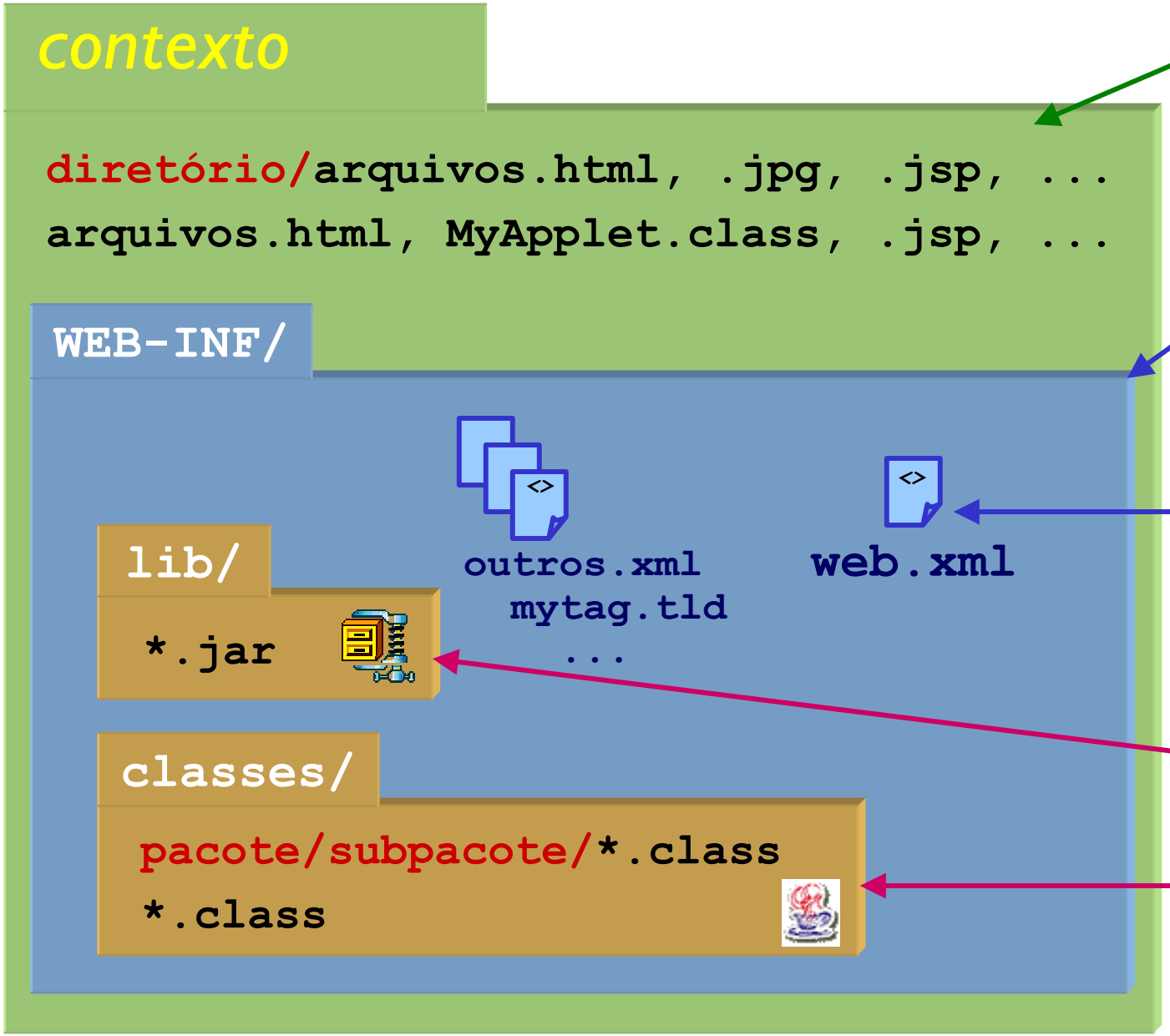
Sobre este módulo

- Neste módulo serão apresentadas aplicações Web configuráveis através de um deployment descriptor
- Aplicações Web são quase a mesma coisa que contextos. Contextos possuem
  - Diretório próprio que define o escopo máximo da aplicação
  - Um CLASSPATH próprio onde rodam os servlets
  - Um arquivo de configuração (web.xml) - o deployment descriptor: usado na implantação da aplicação quando o servidor inicia
- Também serão abordados
  - Passagem de atributos através do contexto e da requisição
  - Logging

Aplicações Web (contextos)

- *Web Containers suportam a implantação de múltiplas aplicações Web*
  - *Definem contextos separados para execução de servlets*
- *No Tomcat, essas aplicações estão na pasta webapps/*
  - *Veja o conteúdo de webapps no seu servidor*
- *Todo diretório de contexto tem uma estrutura definida, que consiste de*
  - *Área de documentos do contexto (/), acessível externamente*
  - *Área inacessível (/WEB-INF), que possui pelo menos um arquivo de configuração padrão (web.xml)*
  - *O WEB-INF pode conter ainda dois diretórios reconhecidos pelo servidor: (1) um diretório que pertence ao CLASSPATH da aplicação (/WEB-INF/classes) e (2) outro onde podem ser colocados JARs para inclusão no CLASSPATH (/WEB-INF/lib)*

Estrutura de uma aplicação Web



Arquivos acessíveis ao cliente a partir da raiz do contexto

Área inaccessível ao cliente

Arquivo de configuração (WebApp deployment descriptor)

Bibliotecas

Classpath (Contém Classes, JavaBeans, Servlets)

Componentes de um contexto

- *A raiz define (geralmente) o nome do contexto.*
  - *Na raiz ficam HTMLs, páginas JSP, imagens, applets e outros objetos para download via HTTP*

{Contexto}/WEB-INF/web.xml

  - *Arquivo de configuração da aplicação*
  - *Define parâmetros iniciais, mapeamentos e outras configurações de servlets e JSPs.*

{Contexto}/WEB-INF/classes/

  - *Classpath da aplicação*

{Contexto}/WEB_INF/lib/

  - *Qualquer JAR incluido aqui será carregado como parte do CLASSPATH da aplicação*

Nome do contexto e URL

- *A não ser que seja configurado externamente, o nome do contexto aparece na URL após o nome/porta do servidor*

```
http://serv:8080/contexto/subdir/pagina.html
http://serv:8080/contexto/servlet/pacote.Servlet
```

- *Para os documentos no servidor (links em páginas HTML e formulários), a raiz de referência é a raiz de documentos do servidor, ou* `DOCUMENT_ROOT`: `http://serv:8080/`
- *Documentos podem ser achados relativos ao* `DOCUMENT_ROOT`

```
/contexto/subdir/pagina.html
/contexto/servlet/pacote.Servlet
```

- *Para a configuração do contexto (web.xml), a raiz de referência é a raiz de documentos do contexto:* `http://serv:8080/contexto/`
- *Componentes são identificados relativos ao contexto*

```
/subdir/pagina.html
/servlet/pacote.Servlet
```

`servlet/` *é mapeamento virtual definido no servidor para servlets em* `WEB-INF/classes`

Tipos e fragmentos de URL

- *URL absoluta: identifica recurso na Internet. Usada no campo de entrada de localidade no browser, em páginas fora do servidor, etc.*

  `http://serv:8080/ctx/servlet/pacote.Servlet/cmd/um`

- *Relativa ao servidor (Request URI): identifica o recurso no servidor. Pode ser usada no código interpretado pelo browser nos atributos HTML que aceitam URLs (para documentos residentes no servidor)*

  `/ctx/servlet/pacote.Servlet/cmd/um`

- *Relativa ao contexto: identifica o recurso dentro do contexto. Pode ser usada no código de servlets e JSP interpretados no servidor e web.xml. Não contém o nome do contexto.*

  `/servlet/pacote.Servlet/cmd/um`

- *Relativa ao componente (extra path information): texto anexado na URL após a identificação do componente ou página*

  `/cmd/um`

Criando um contexto válido

- *Para que uma estrutura de diretórios localizada no webapps/ seja reconhecida como contexto pelo Tomcat, na inicialização, deve haver um arquivo web.xml no diretório WEB-INF do contexto*
  - *O arquivo é um arquivo XML e deve obedecer às regras do XML e do DTD definido pela especificação*
  - *O conteúdo mínimo do arquivo é a declaração do DTD e um elemento raiz <web-app/>*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app/>
```

  - *Se houver qualquer erro no web.xml, a aplicação não será carregada durante a inicialização*

Exercício

- 1. Construa um novo contexto exercicio
  - Crie um diretório exercicio dentro de webapps/
  - Monte a estrutura esperada, com diretório WEB-INF, diretório classes para os servlets e web.xml vazio (copie o que está no WEB-INF do ROOT)
  - Copie alguns dos servlets e páginas Web que você fez em exercícios anteriores para ele
  - Reinicie o servidor
  - Teste o contexto chamando servlets e páginas Web localizadas dentro dele

Configuração de instalação

- *Toda aplicação Web possui um arquivo web.xml que configura a implantação de todos os seus componentes*
  - *Configura inicialização de instâncias de servlets*
  - *Define mapeamentos de nomes a servlets (aliases)*
  - *Pode conter instrução para carregar previamente páginas JSP*
  - *Configura inicialização do contexto (aplicação)*
  - *Define permissões e perfis de usuário*
  - *Configura tempo de timeout de sessão*
  - ...
- *Para criar e editar o web.xml, use um editor XML que possa validá-lo através do DTD:* **`web-app_2_3.dtd`**
  - *O DTD é comentado e explica para que serve cada elemento*
  - *Ache-o em:* **`java.sun.com/dtd/web-app_2_3.dtd`**
  - *Instale o DTD em sua máquina local e use-o para validar seu documento caso não esteja conectado à Internet*

XML e DTDs

- *Um DTD (Document Type Definition) declara todos os elementos e atributos de um documento XML*
  - *Define quais elementos e atributos são válidos e em que contexto*
  - *A sintaxe é SGML. Para definir um elemento:*

    ```
    <!ELEMENT nome-do-elemento (modelo de conteudo)>
    ```

  - *O DTD do web.xml define principalmente elementos*
- *Exemplo: DTD para um documento simples*

```
<!ELEMENT pessoa (nome, web?, telefone+)>
<!ELEMENT nome (prenome, inicial*, sobrenome)>
<!ELEMENT prenome (#PCDATA)>
<!ELEMENT inicial (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT web (email|website)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT website (#PCDATA)>
<!ELEMENT telefone (#PCDATA)>
```

pessoa *tem* **nome**, *seguido de zero ou um* **web** *e um ou mais* **telefone**

nome *tem um* **prenome**, *seguido de zero ou mais* **inicial** *e um* **sobrenome**

web *pode conter ou um* **email** *ou um* **website**

Elementos que só podem conter texto

Documentos válidos segundo o DTD

- *Os documentos abaixo são válidos segundo o DTD mostrado na página anterior*

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <telefone>1199343232</telefone>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <web><website>www.site.com</website></web>
  <telefone>1199343232</telefone>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <inicial>F</inicial><inicial>R</inicial>
        <sobrenome>Bruno</sobrenome></nome>
  <web><email>giordano@web.net</email></web>
  <telefone>1199343232</telefone>
  <telefone>1134999992</telefone>
</pessoa>
```

Documentos inválidos segundo o DTD

- *Os documentos abaixo não são válidos de acordo com o DTD.*
- *Por que?*

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <web><website>www.site.com</website>
       <email>giordano@web.net</email></web>
  <telefone>1199343232</telefone>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <telefone>1199343232</telefone>
  <telefone>1134999992</telefone>
  <web><email>giordano@web.net</email></web>
</pessoa>
```

Elemento <!DOCTYPE>

- O elemento <!DOCTYPE> é um elemento do DTD que deve ser usado dentro da página XML
  - Identifica o elemento raiz
  - Associa o arquivo a um DTD através de uma URL e/ou de um identificador formal público
- <!DOCTYPE> deve aparecer no início do documento XML

```
<!DOCTYPE pessoa SYSTEM "/docs/dtd/pessoa.dtd">
<pessoa>
...
</pessoa>
```

Deve ser o mesmo

Onde está o DTD

- Alguns DTDs possuem um identificador formal público (FPI)
  - Neste caso, são declarados com a palavra PUBLIC e duas strings: o identificador seguido de uma URL onde pode ser encontrado

```
<!DOCTYPE pessoa  PUBLIC  "-//ACME, Inc.//DTD Pessoa//PT"
                          "http://localhost/pessoa.dtd">
```

Arquivo web.xml

- O arquivo web.xml necessita de declaração <!DOCTYPE> pública, que tem a seguinte sintaxe

```
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
```

- O identificador formal deve ser sempre o mesmo. A URL pode ser alterada para apontar para um caminho local ou outro endereço, se necessário
- Uma aplicação Web sempre tem um arquivo web.xml. Se não for necessária configuração alguma em seus servlets e JSPs, pode-se usar o web.xml mínimo:

```
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app></web-app>
```

Elementos XML do web.xml

- Consulte o DTD (faz parte da especificação) para maiores detalhes sobre os elementos que podem ser usados no web.xml
  - A ordem dos elementos é importante: se eles estiverem na ordem incorreta, o XML não validará com o DTD
  - Existem elementos que exigem a presença de outros (e isto não é validado pelo DTD - leia os comentários no DTD)
- Use um editor XML
  - O JEdit habilitado com plug-in XML valida qualquer XML com seu DTD desde que se informe a localização do DTD em sua declaração <!DOCTYPE> (no web.xml, altere, se necessário, o segundo argumento, que representa a URL onde o DTD pode ser achado)
  - Se estiver online, o JEdit achará o DTD no site da Sun. Se não estiver, copie-o e use-o localmente
  - Não tente instalar um contexto sem antes validar o seu web.xml. Se ele não estiver válido a instalação não irá funcionar.

Exemplo de web.xml (1/2)

```
<web-app>
  <context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
  </context-param>

  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>exemplo.pacote.MyServlet</servlet-class>
    <init-param>
      <param-name>datafile</param-name>
      <param-value>data/data.txt</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>myJSP</servlet-name>
    <jsp-file>/myjsp.jsp</jsp-file>
    <load-on-startup>2</load-on-startup>
  </servlet>
                    ...
```

Parâmetro que pode ser lido por todos os componentes

Instância de um Servlet

Parâmetro que pode ser lido pelo servlet

Ordem para carga prévia do servlet

Declaração opcional de página JSP

Ordem para pré-compilar JSP

Exemplo de web.xml (2/2)

```
...
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>404</error-code>
    <location>/notFound.jsp</location>
  </error-page>

</web-app>
```

Servlet examples.MyServlet foi mapeado à URL /myservlet

Sessão do usuário expira expira em 60 minutos

Lista de arquivos que serão carregados automaticamente em URLs terminadas em diretório

Redirecionar para esta página em caso de erro 404

Exercícios

- 2. Use os arquivos XML exemplo mostrados e valide-os com o DTD para comprovar se realmente são válidos ou não
  - Inclua uma declaração
    <!DOCTYPE pessoa SYSTEM "pessoa.dtd">
- 3. Pesquise o DTD do web.xml, descubra para que servem e use os elementos:
  - <welcome-file-list>
  - <description>
  - <error-page>

Instâncias de servlets

- *Uma instância de um servlet pode ser configurada no web.xml através do elemento <servlet>*

```
<servlet>
   <servlet-name>myServlet</servlet-name>
   <servlet-class>exemplo.pacote.MyServlet</servlet-class>
   <!-- elementos de configuração opcionais aqui -->
</servlet>
```

  - *<servlet-name> e <servlet-class> são obrigatórios*
- *É uma boa prática escolher nomes de servlets seguindo as convenções Java*
  - *Use caixa mista, colocando em maiúsculas cada palavra, mas comece com letra minúscula. Ex: banco, pontoDeServico*
- *Pode-se criar múltiplas instâncias da mesma classe definindo blocos servlet com <servlet-name> diferentes*
  - *Não terão muita utilidade a não ser que tenham também configuração diferente e mapeamentos diferentes*

Servlet alias (mapeamento) no web.xml

- É uma boa prática definir aliases para os servlets
  - Nomes grandes são difíceis de digitar e lembrar
  - Expõem detalhes sobre a implementação das aplicações
- Para definir um mapeamento de servlet é necessário usar <servlet> e <servlet-mapping>
- <servlet-mapping> associa o nome do servlet a um padrão de URL relativo ao contexto. A URL pode ser
  - Um caminho relativo ao contexto iniciando por /
  - Uma extensão de arquivo, expresso da forma *.extensao

```
<servlet>
    <servlet-name>umServlet</servlet-name>
    <servlet-class>pacote.subp.MeuServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>umServlet</servlet-name>
    <url-pattern>/programa</url-pattern>
</servlet-mapping>
```

Sintaxe de mapeamentos

- *Mapeamento exato*
  - *Não aceita /nome/ ou /nome/x na requisição*

```
<url-pattern>/nome</url-pattern>
<url-pattern>/nome/subnome</url-pattern>
```

- *Mapeamento para servlet default*
  - *Servlet é chamado se nenhum dos outros mapeamentos existentes combinar com a requisição*

```
<url-pattern>/</url-pattern>
```

- *Mapeamento de caminho*
  - *Aceita texto adicional (path info) após nome do servlet na requisição*

```
<url-pattern>/nome/*</url-pattern>
<url-pattern>/nome/subnome/*</url-pattern>
```

- *Mapeamento de extensão*
  - *Arquivos com a extensão serão redirecionados ao servlet*

```
<url-pattern>*.ext</url-pattern>
```

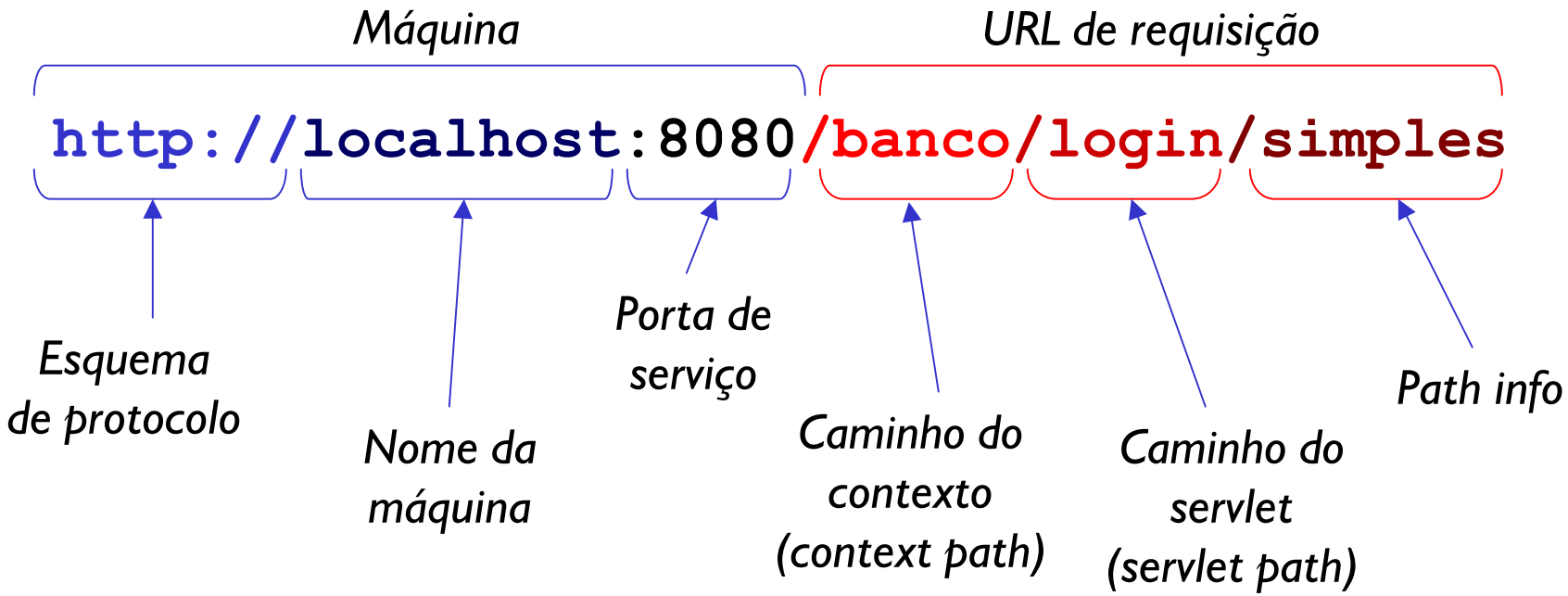
Processamento de URLs e mapeamentos

- O Web container procura dentre os mapeamentos existentes no web.xml, o maior que combine com a URL recebida
  1. Procura primeiro um mapeamento exato
  2. Se não achar, procura entre os caminhos que terminam em `*`.
  3. Por último, procura pela extensão do arquivo, se houver
  4. Não havendo combinação, redireciona ao servlet default se este tiver sido declarado ou exibe um erro, se não houver servlet default
- Após uma combinação, texto adicional à direita na URL recebida será considerado path info
  - Pode ser recuperado com `request.getPathInfo()`
- Considere, por exemplo, o mapeamento `/um/*` e a URL

  `http://localhost:8080/contexto/um/dois/tres/abc.txt`
- Mesmo que exista um mapeamento para `*.txt`, este não será considerado pois antes haverá combinação para `/um`
  - `/dois/tres/abc.txt` é path info!

Anatomia de uma URL

- *Diferentes partes de uma URL usada na requisição podem ser extraídas usando métodos de HttpServletRequest*
  - `getContextPath()`: */banco*, *na URL abaixo*
  - `getServletPath()`: */login*, *na URL abaixo*
  - `getPathInfo()`: */simples*, *na URL abaixo*



Exercícios

- 4. Crie uma aplicação Web chamada miniforum
  - Construa a estrutura com WEB-INF e web.xml
- 5. Escreva um servlet (j550.miniforum.ForumServlet) que receba dois parâmetros: uma mensagem e o e-mail de quem a enviou
  - Crie um formulário HTML novaMensagem.html
  - Grave os dados em um arquivo
  - Mostre na tela a mensagem enviada como resposta
- 6. Escreva um servlet (j550.miniforum.MensagemServlet) que liste todas as mensagens
- 7. Faça mapeamentos no web.xml para que os dois servlets possam ser chamados pelas seguintes URLs no HTML:
  - /forum/gravar - para ForumServlet
  - /forum/listar - para MensagemServlet

ServletConfig

- A interface ServletConfig serve para que um servlet possa ter acesso a informações de configuração definidas no web.xml
- Todo servlet implementa ServletConfig e, portanto, tem acesso aos seus métodos
- Principais métodos de interesse
  - String getInitParameter(String nome): lê um parâmetro de inicialização <init-param> do web.xml
  - Enumeration getInitParameterNames(): obtém os nomes de todos os parâmetros de inicialização disponíveis
- Os métodos de ServletConfig devem ser chamados no método init(), do servlet

Definição de parâmetros de inicialização

- Parâmetros de inicialização podem ser definidos para cada instância de um servlet usando o elemento <init-param> dentro de <servlet>
  - Devem aparecer depois de <servlet-name> e <servlet-class> (lembre-se que a ordem foi definida no DTD)
  - Requer dois sub-elementos que definem o nome do atributo e o seu valor

```
<servlet>
    <servlet-name>umServlet</servlet-name>
    <servlet-class>pacote.subp.MeuServlet</servlet-class>
    <init-param>
      <param-name>dir-imagens</param-name>
      <param-value>c:/imagens</param-value>
    </init-param>
    <init-param> ... </init-param>
  </servlet>
```

Leitura de parâmetros de inicialização

- *Parâmetros de inicialização podem ser lidos no método init() e guardados em variáveis de instância para posterior uso dos métodos de serviço*

```
private java.io.File dirImagens = null;

public void init() throws ServletException {
    String dirImagensStr =
              getInitParameter("dir-imagens");
    if (dirImagens == null) {
        throw new UnavailableException
                    ("Configuração incorreta!");
    }
    dirImagens = new File(dirImagensStr);
    if (!dirImagens.exists()) {
       throw new UnavailableException
                    ("Diretorio de imagens nao existe!");
    }
}
```

Exercícios

- 8. Guarde a cor do texto e a cor de fundo da página que mostra as mensagens como parâmetros de inicialização do servlet j550.cap03.MensagemServlet
  - Use cores HTML (red, blue, #FF0000, ou CSS)
  - Leia os parâmetros na inicialização e guarde-os em variáveis de instância
  - Monte a página HTML com os dados dos parâmetros
- 9. Crie uma segunda instância do mesmo servlet MensagemServlet (use outro nome e outro mapeamento no web.xml)
  - Defina os mesmos parâmetros com valores diferentes
  - Chame o segundo servlet e veja os resultados

Três níveis de configuração

- 1. Nível do servidor (não faz parte da especificação)
  - Configuração definida fora do contexto,em arquivos de configuração do fabricante (ex: server.xml, jboss-web.xml) ou na configuração de um arquivo EAR (J2EE)
  - Permite configuração do nome do contexto raiz
- 2. Nível da aplicação (contexto) - web.xml
  - Aplicada a todos os componentes da aplicação
  - Lista de componentes, mapeamentos, variáveis compartilhadas, recursos e beans compartilhados, serviços compartilhados, timeout da sessão, etc.
- 3. Nivel do componente (servlet, JSP) - <servlet>
  - Configuração de servlets, filtros e páginas individuais
  - Parâmetros iniciais, regras de carga, etc.

ServletContext

- A interface ServletContext encapsula informações sobre o contexto ou aplicação
- Cada servlet possui um método getServletContext() que devolve o contexto atual
  - A partir de uma referência ao contexto atual pode-se interagir com o contexto e compartilhar informações entre servlets
- Principais métodos de interesse de ServletContext
  - String getInitParameter(String): lê parâmetros de inicialização do contexto (não confunda com o método similar de ServletConfig!)
  - Enumeration getInitParameterNames(): lê lista de parâmetros
  - InputStream getResourceAsStream(): lê recurso localizado dentro do contexto como um InputStream
  - setAttribute(String nome, Object): grava um atributo no contexto
  - Object getAttribute(String nome): lê um atributo do contexto
  - log(String mensagem): escreve mensagem no log do contexto

Inicialização de contexto

- No web.xml, <context-param> vem antes de qualquer definição de servlet

```
<context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
</context-param>
```

- No servlet, é preciso primeiro obter uma instância de ServletContext antes de ler o parâmetro

```
ServletContext ctx = getServletContext();
String tempDir = ctx.getInitParameter("tempdir");
if (tempDir == null) {
   throw new UnavailableException("Configuração errada");
}
```

Carregamento de arquivos no contexto

- O método getResourceAsStream() permite que se localize e se carregue qualquer arquivo no contexto sem que seja necessário saber seu caminho completo
  - Isto é importante pois contextos podem ser usados em diferentes servidores e armazenados em arquivos WAR
- Exemplo

```
ServletContext ctx = getServletContext();
String arquivo = "/WEB-INF/usuarios.xml";
InputStream stream = ctx.getResourceAsStream(arquivo);
InputStreamReader reader =
                    new InputStreamReader(stream);
BufferedReader in = new BufferedReader(reader);
String linha = "";
while ( (linha = in.readLine()) != null) {
   // Faz alguma coisa com linha de texto lida
}
```

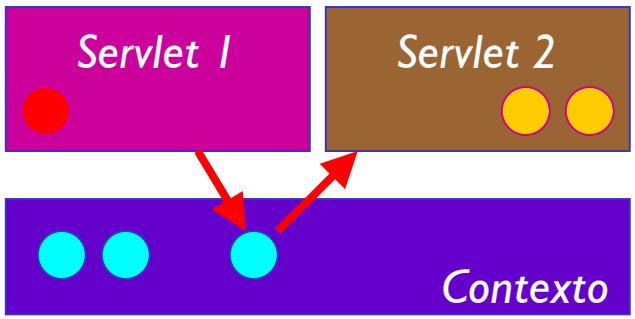
Logging

- *Para cada contexto criado, o servidor fornece um arquivo de log, onde mensagens serão gravadas*
  - *O arquivo só passará a existir quando a primeira mensagem for gravada. Procure em logs/ no Tomcat.*
- *Há dois métodos disponíveis*
  - `log(String mensagem)`: *grava uma mensagem*
  - `log(String mensagem, Throwable exception)`: *grava uma mensagem e o stack-trace de uma exceção. Use nos blocos try-catch.*
- *Log é implementado em GenericServlet também. Pode, portanto, ser chamado tanto do contexto como do servlet*

```
log("Arquivo carregado com sucesso.");
ServletContext ctx = getServletContext();
ctx.log("Contexto obtido!");
```

Gravação de atributos no contexto

- *Servlets podem compartilhar objetos pelo contexto usando*
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`



- *Exemplo de uso*

Servlet 1

```
String[] vetor = {"um", "dois", "tres"};
ServletContext ctx = getServletContext();
ctx.setAttribute("dados", vetor);
```

Servlet 2

```
ServletContext ctx = getServletContext();
String[] dados = (String[])ctx.getAttribute("dados");
```

- *Outros métodos*
  - `removeAttribute(String nome)` - *remove um atributo*
  - `Enumeration getAttributeNames()` - *lê nomes de atributos*

ServletContextListener

- Não existem métodos init() ou destroy() globais para realizar operações de inicialização/destruição de um contexto
  - A forma de controlar o ciclo de vida global para um contexto é através da implementação de um ServletContextListener
- ServletContextListener é uma interface com dois métodos
  - public void contextInitialized(ServletContextEvent e)
  - public void contextDestroyed(ServletContextEvent e)

  que são chamados respectivamente depois que um contexto é criado e antes que ele seja destruído. Para isto é preciso registrá-lo no web.xml usando o elemento <listener>

```
<listener>
   <listener-class>ex01.OuvinteDeContexto</listener-class>
</listener>
```

- ServletContextEvent possui um método getServletContext() que permite obter o contexto associado

Outros listeners de contexto

- É possível saber quando um atributo foi adicionado a um contexto usando ServletContextAttributeListener e ServletContextAttributeEvent
- Métodos a implementar do Listener
  - attributeAdded(ServletContextAttributeEvent e)
  - attributeRemoved(ServletContextAttributeEvent e)
  - attributeReplaced(ServletContextAttributeEvent)
- ServletContextAttributeEvent possui dos métodos para recuperar nome e valor dos atributos
  - String getName()
  - String getValue()
- É preciso registrar o listener no web.xml

Contexto externo no Tomcat

- *Sempre que possível, crie seus contextos no webapps*
- *Se for necessário, é possível fazer o Tomcat criar contextos em outro lugar, alterando a configuração. Acrescente um elemento <Context> em* `$TOMCAT_HOME/conf/server.xml`

```
<Server ...> ...
   <Service ...> ...
      <Engine ...> ...
         <Host ...>
           ...
             <Context path="/sistema"
                      docBase="c:\sistema\build" />
...
```

Específico do Tomcat!

- *docBase também pode conter endereço relativo a webapps (esses são gerados automaticamente) - veja DTD!*
- *É preciso reiniciar o servidor*

Exercício

- 10. Guarde o nome do arquivo compartilhado por ForumServlet e MensagemServlet como um parâmetro de inicialização de contexto
  - Guarde o arquivo dentro de WEB-INF e o caminho no parâmetro de inicialização
  - Recupere o parâmetro no init() de seu servlet e guarde-o em uma variável de instância. Cause uma Unavailable Exception caso o parâmetro seja null.
  - Use getResourceAsStream() para recuperar um stream para o arquivo.
- 11. Guarde, como atributo de contexto, um número, e incremente-o a cada acesso
  - Imprima na página o número de acessos.

Introdução a Model View Controller

- Apesar de servlets não separarem código de resposta do código de requisição explicitamente, isto pode ser feito pelo desenvolvedor
  - Melhor separação de responsabilidades: cada método cuida de uma coisa - ou lógica de negócios, ou controle de requisição ou geração de respostas
  - Maior facilidade para migrar para solução JSP-servlet
- Lógica de negócios deve ficar em classes externas, executadas pelos métodos controladores e pesquisadas pelos métodos de geração de resposta
- Controlador deve selecionar o método de resposta adequado após o processamento
  - Dados podem ser passados através da requisição usando atributos de requisição (não use variáveis de instância)

Exemplo: MVC com servlets

```
public class MVCServlet extends HttpServlet {
   private Servico servico;
   public void init() {
      ... inicializa servico
   }
   public void doGet(...) { processar(...);}
   public void doPost(...) { processar(...);}

   public void processar(... request, ... response) {
      Produto p = new Produto(123, "Livro");
      servico.adicionar("produto", p);
      ...
      if (funcionou) {
         request.setAttribute(p);
         sucesso(request, response);
      } ...
   }
   public void sucesso(... request, ... response) {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      Produto p = (Produto) request.getAttribute("produto"); // ...
      out.println("<td>"+p.getCodigo()+"</td>"); // ...
   }
}
```

*Variável de instância compartilhada contendo objeto do **Model***

Controller

*Objetos do **Model** sendo manipulados no Controller*

*Objetos do **Model** sendo lidos no View*

View

Passagem de atributos pela requisição

- Para compartilhar dados entre métodos de serviço, não use variáveis estáticas ou de instância
  - Elas são compartilhadas por todos os clientes!
- Use atributos de requisição (HttpServletRequest)
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`
- Atributos são destruídos junto com a requisição
  - Não são compartilhados entre clientes
  - É a forma recomendada de comunicação entre métodos de serviço e objetos na mesma requisição
  - Se desejar reter seu valor além da requisição, copie-os para um objeto de persistência maior (por exemplo, um atributo de contexto)

Escopo e threads

- Geralmente, só há uma instância de um servlet rodando para vários clientes
  - Atributos de instância são compartilhados!
- Se não desejar compartilhar dados entre clientes, use sempre objetos thread-safe
  - Atributos guardados no request
  - Variáveis locais
- Quaisquer outros atributos, como atributos de sessão, atributos de instância e de contexto são compartilhados entre requisições
  - Caso deseje compartilhá-los, use synchronized nos blocos de código onde seus valores são alterados.

Exercício

- 12. Altere MensagemServlet e ForumServlet para que contenham apenas lógica de processamento Web, delegando as operações lógicas com mensagens para a classe Mensagem
  - Acesso ao arquivo deve estar em Mensagem
  - Servlets devem criar objeto e manipular métodos de Mensagem: get/setTexto(), get/setEmail() sem se preocupar com arquivos
  - Passe o stream como argumento do construtor de Mensagem
- 13. Separe os métodos relacionados com saída (View) dos métodos relacionados com entrada e processamento (Controller e Model)
  - Métodos Controller controlam a requisição, chamam métodos de processamento no Model (criam objeto, preenchem dados) e escolhem o método de View desejado
  - Métodos View lêem o Model (getXXX()) e geram a resposta
  - Passe objetos entre os métodos usando atributos de requisição

Repasse de requisição

- Objetos *RequestDispatcher* servem para repassar requisições para outra página ou servlet. Seus dois principais métodos são

```
include(request, response)
```

```
forward(request, response)
```

- Esses métodos não podem definir cabeçalhos
  - *forward()* repassa a requisição para um recurso
  - *include()* inclui a saída e processamento de um recurso no servlet
- Para obter um RequestDispatcher use o ServletRequest

```
RequestDispatcher dispatcher =
  request.getRequestDispatcher("url");
```

- Para repassar a requisição para outra máquina use

```
dispatcher.forward(request, response);
```

- No repasse de requisição, o controle não volta para o browser.
  - Todos os parâmetros e atributos da requisição são preservados

Redirecionamento x Repasse

- Pode-se enviar um cabeçalho de redirecionamento para o browser usando

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

- Isto é o mesmo que fazer

```
response.setHeader("Location", "url");
```

- Location é um cabeçalho HTTP que instrui o browser para redirecionar para outro lugar
- Sempre que o controle volta ao browser, a primeira requisição terminou e outra foi iniciada
  - Os objetos HttpServletResponse e HttpServletRequest e todos seus atributos e parâmetros foram destruídos
- Com repasse de requisições, usando RequestDispatcher, o controle não volta ao browser mas continua em outro servlet (com forward())ou no mesmo servlet (com include())

Exercícios

- 14. Crie j550.miniforum.PortalServlet que redirecione a resposta para MensagemServlet ou formulário para entrada de mensagens dependendo do comando recebido como parâmetro
  - Implemente dois links para o mesmo servlet: Listar Mensagens e Nova Mensagem
  - comando=listar - redirecione para /forum/listar
  - comando=criar - redirecione para /forum/gravar (ou para forum/novaMensagem.html)
  - Crie um mapeamento para o servlet: /miniforum/portal
- 15. Use repasse de requisição para refazer o exercício 14.

helder@acm.org

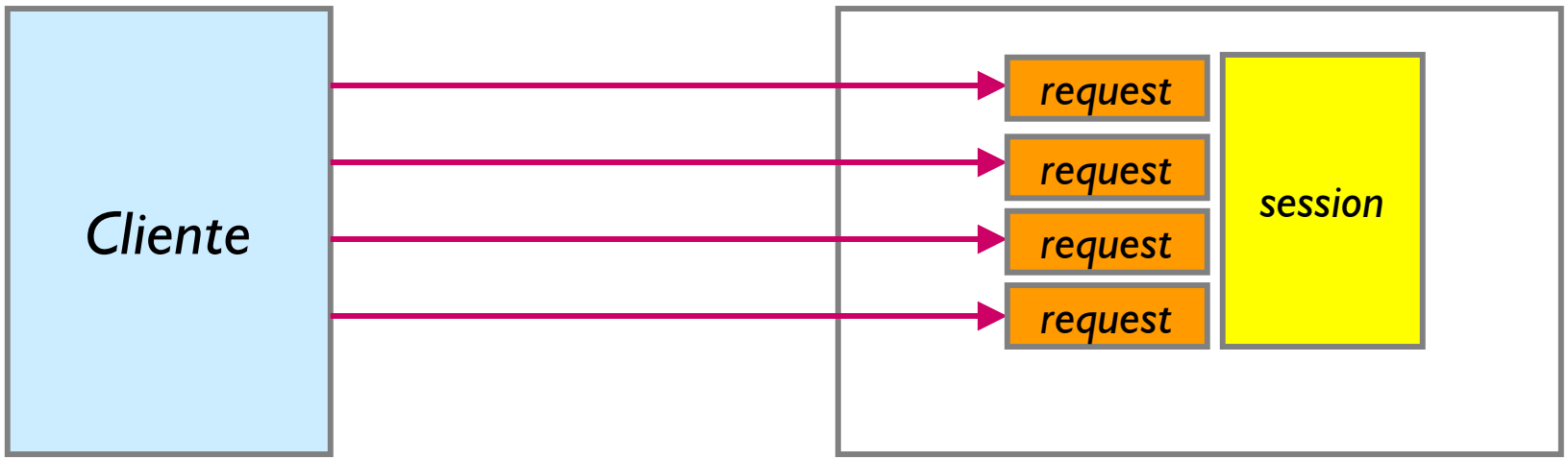
argonavis.com.br

4 Sessões

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Sessões

- *Como o HTTP não mantém estado de sessão, são as aplicações Web que precisam cuidar de mantê-lo quando necessário*
- *Sessões representam um cliente*
  - *A sessão é única para cada cliente e persiste através de várias requisições*



Sessões

- Sessões são representados por objetos *HttpSession* e são obtidas a partir de uma requisição
- Dois métodos podem ser usados

```
HttpSession session = request.getSession(false);
```

  - Se a sessão não existir, retorna null, caso contrário retorna sessão.

```
HttpSession session = request.getSession();
```

  - Retorna a sessão ou cria uma nova. Mesmo que *getSession(true)*
- Para saber se uma sessão é nova, use o método *isNew()*

```
if (session.isNew()) {
  myObject = new BusinessObject();
} else {
  myObject = (BusinessObject) session.getAttribute("obj");
}
```

- *getSession()* deve ser chamado antes de *getOutputStream()**
  - Sessões podem ser implementadas com cookies, e cookies são definidos no cabeçalho HTTP (que é montado antes do texto)

*ou qualquer método que obtenha o stream de saída, como getWriter()

Compartilhamento de objetos na sessão

- *Dois métodos*
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`

  permitem o compartilhamento de objetos na sessão. Ex:

  Requisição 1

```
String[] vetor = {"um", "dois", "tres"};
HttpSession session = request.getSession();
session.setAttribute("dados", vetor);
```

  Requisição 2

```
HttpSession session = request.getSession();
String[] dados = (String[])session.getAttribute("dados");
```

- *Como a sessão pode persistir além do tempo de uma requisição, é possível que a persistência de alguns objetos não sejam desejáveis*
  - *Use* `removeAttribute("nome")` *para remover objetos da sessão*

Gerência de sessões

- Não há como saber que cliente não precisa mais da sessão
  - Pode-se definir um timeout em minutos para a duração de uma sessão desde a última requisição do cliente

    `setMaxInactiveInterval(int)` define novo valor para timeout

    `int getMaxInactiveInterval()` recupera valor de timeout
  - Timeout default pode ser definido no web.xml para todas as sessões
  - Outros métodos úteis: getLastAccessedTime() e getCreationTime()
- Para destruir uma sessão use

  ```
  session.invalidate();
  ```
- Eventos de ligação e ativação de uma sessão podem ser controlados com implementações das interfaces HttpSessionBindingListener e HttpSessionActivationListener
  - Consulte a documentação. A abordagem dessas interfaces não faz parte do escopo deste curso

Timeout default no web.xml

- O elemento <session-config> permite definir a configuração da sessão
  - Deve aparecer depois dos elementos <servlet-mapping>
- O trecho abaixo redefine o tempo de duração default da sessão em 15 minutos para todas as sessões

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

- Uma sessão específica pode ter uma duração diferente se especificar usando *setMaxInactiveInterval()*

Sessão à prova de clientes

- A sessão é implementada com cookies se o cliente suportá-los
  - Caso o cliente não suporte cookies, o servidor precisa usar outro meio de manter a sessão
- Solução: sempre que uma página contiver uma URL para outra página da aplicação, a URL deve estar dentro do método encodeURL() de HttpServletResponse

```
out.print("<a href='" +
                response.encodeURL("caixa.jsp") + "'>");
```

- Se cliente suportar cookies, URL passa inalterada (o identificador da sessão será guardado em um cookie)
- Se cliente não suportar cookies, o identificador será passado como parâmetro da requisição

  ex: http://localhost:8080/servlet/Teste;jsessionid=A424JX08S99

Captura de eventos de atributos

- É possível saber quando um atributo foi adicionado a uma sessão usando HttpSessionAttributeListener e HttpSessionBindingEvent
- Métodos a implementar do Listener
  - attributeAdded(ServletContextAttributeEvent e)
  - attributeRemoved(ServletContextAttributeEvent e)
  - attributeReplaced(ServletContextAttributeEvent)
- HttpSessionBindingEvent possui três métodos para recuperar sessão e nome e valor dos atributos
  - String getName()
  - String getValue()
  - HttpSession getSession()
- É preciso registrar o listener no web.xml

Captura de eventos do ciclo de vida

- Pode-se saber quando uma sessão foi criada, invalidada ou expirada usando HttpSessionListener:
  - Métodos sessionCreated() e sessionDestroyed()
- Para saber quando uma sessão existente foi ativada ou está para ser passivada usa-se HttpSessionActivationListener:
  - Métodos sessionDidActivate() e sessionWillPassivate()
- Para controlar quando objetos são associados a uma sessão e quando deixam a sessão (por qualquer razão) deve-se implementar um HttpSessionBindingListener
  - Métodos valueBound() e valueUnbound()
- Cada listener tem um evento correspondente, que é recebido em cada método. Para maiores detalhes, consulte a documentação e exemplos no Tomcat
  - Maiores detalhes sobre este assunto fogem ao escopo deste curso

Escopo de objetos em servlets

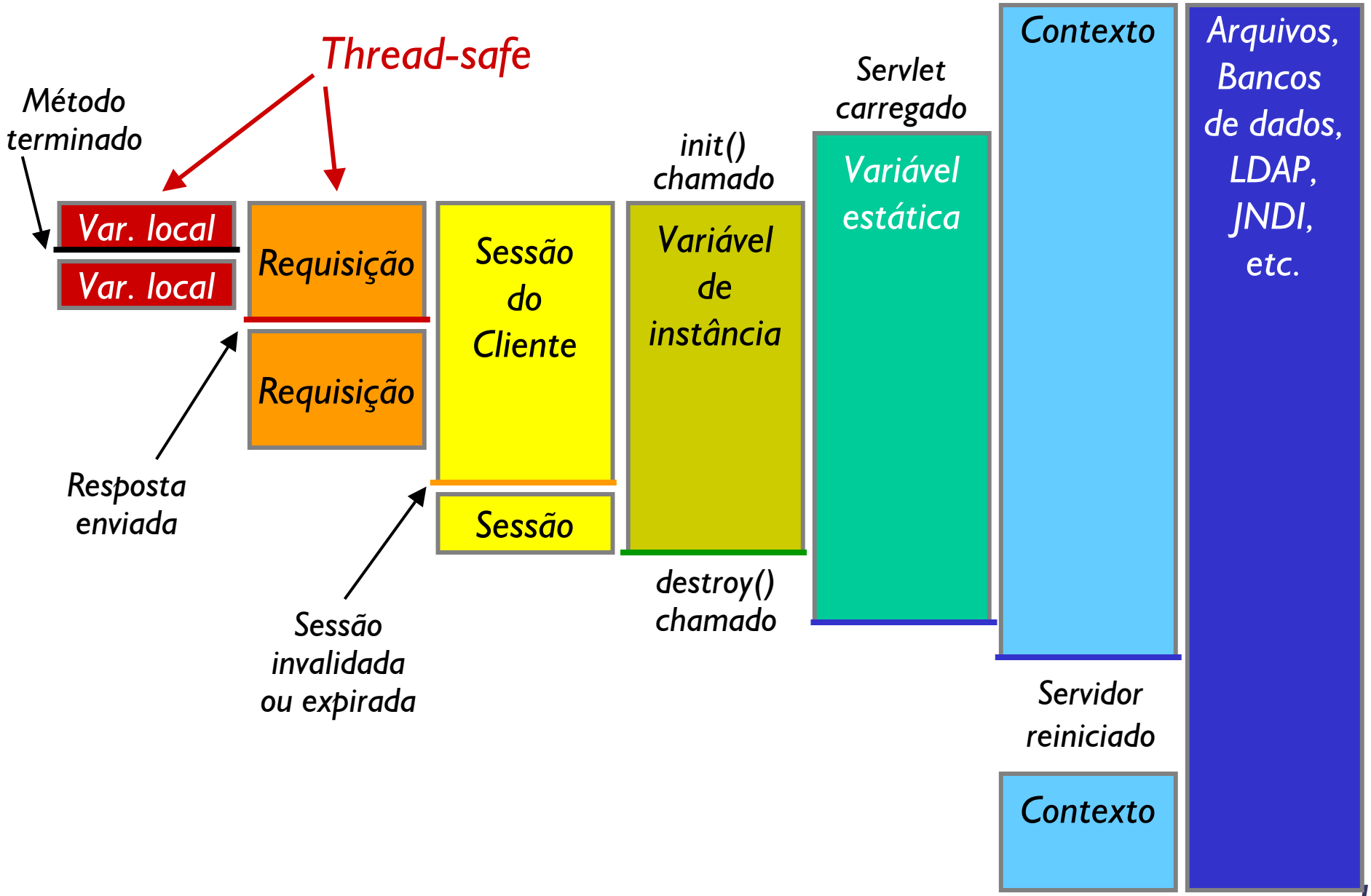
- Servlets podem compartilhar informações de várias maneiras
  - Usando meios persistentes (bancos de dados, arquivos, etc)
  - Usando objetos na memória por escopo (requisição, sessão, contexto)
  - Usando variáveis estáticas ou de instância
- Servlets oferecem três níveis diferentes de persistência na memória (ordem decrescente de duração)
  - Contexto da aplicação: vale enquanto aplicação estiver na memória (javax.servlet.ServletContext)
  - Sessão: dura uma sessão do cliente (javax.servlet.http.HttpSession)
  - Requisição: dura uma requisição (javax.servlet.ServletRequest)
- Para gravar dados em um objeto de persistência na memória

```
objeto.setAttribute("nome", dados);
```

- Para recuperar ou remover os dados

```
Object dados = objeto.getAttribute("nome");
objeto.removeAttribute("nome");
```

Escopo de objetos em servlets: resumo



Lidando com recursos compartilhados

- Há vários cenários de acesso concorrente
  - Componentes compartilhando sessão ou contexto
  - Threads acessando variáveis compartilhadas
- Servlets são automaticamente multithreaded
  - O container cria um thread na instância para cada requisição
  - É preciso sincronizar blocos críticos para evitar problemas decorrentes do acesso paralelo
- Exemplo: protegendo definição de atributo de contexto:

```
synchronized(this) {
    context.setAttribute("nome", objeto);
}
```

- Para situações onde multithreading é inaceitável, servlet deve implementar a interface SingleThreadModel (só um thread estará presente no método service() ao mesmo tempo)
  - Evite isto a todo custo: muito ineficiente!

Exercício

1. Criar uma aplicação Web usando os objetos de negócio
   - j550.cap04.Produto. Atributos (métodos get/set): int id, String nome, String preco
   - j550.cap04.Carrinho. Métodos: addProduto(Produto), removeProduto(id), Produto getProduto(id), Produto[] getProdutos()
   - Veja como usá-los na classe TestaProdutos.java
- a. Crie um servlet j550.cap04.web.AdminLojaServlet
  - LojaServlet recebe parâmetros para adicionar um produto e lista os produtos existentes como resposta
- b. Crie um servlet j550.cap04.web.ComprasServlet e classe
  - ComprasServlet lista todos os produtos disponíveis com um botão Adicionar ao lado de cada um. O botão deve adicionar o produto correspondente no objeto Carrinho.
  - A resposta deve mostrar cada item incluído com um botão Remover. Deve haver também botão Comprar Mais e Encerrar
  - O Carrinho deve persistir entre requisições

Cookies

- Como já podemos manipular sessões de maneira transparente com HttpSession, usamos cookies principalmente para definir preferências que irão durar além do tempo da sessão
  - Servidor irá criar cabeçalho que irá instruir o browser a criar um arquivo guardando as informações do cookie
- Para criar cookies que duram mais que uma sessão (cookies persistentes no disco do cliente) é preciso
  - Criar um novo objeto Cookie
  - Definir a duração do cookie com o método setMaxAge()
  - Definir outros métodos se necessário
  - Adicionar o cookie à resposta

Como usar Cookies

- *Exemplo de gravação: 1) definir um cookie que contenha o nome do usuário recebido como parâmetro na requisição*

```
String nome = request.getParameter("nome");
Cookie c = new Cookie("usuario", nome);
```

- *2) Definir a duração do cookie em segundos*

```
c.setMaxAge(1000 * 24 * 3600 * 60); // 60 dias
```

- *3) Adicionar o cookie à resposta*

```
response.addCookie(c);
```

- *Exemplo de leitura: 1) recuperar o cookie da requisição*

```
Cookie[] cookies = request.getCookies();
```

- *2) Extrair cookie para um objeto local*

```
for (int i = 0; i < cookies.length; i++) {
   if (cookies[i].getName().equals("nome") {
        usuario = cookies[i].getValue();
    }
}
```

Exercícios

- 2. Crie uma tela de entrada na loja LojaServlet com links para os servlets.
  - Ela deve requisitar um e-mail. Grave o e-mail como um Cookie com duração de 30 dias. "Lembre-se" do e-mail na próxima requisição e mostre-o no text-field

helder@acm.org

argonavis.com.br

Componentes Web J2EE

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Sobre este módulo

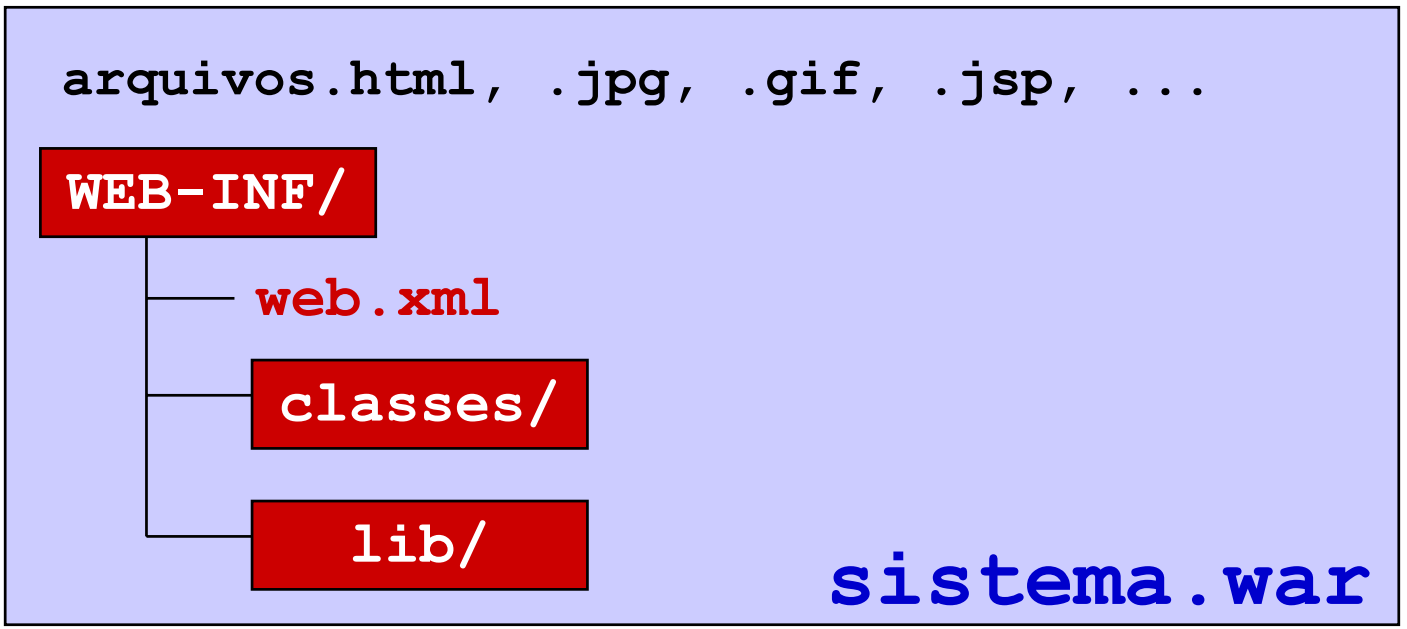
- Neste módulo conheceremos uma nova maneira de fazer deployment: arquivos WAR
  - Com arquivos WAR pode-se colocar uma aplicação no ar, em muitos casos, simplesmente copiando o arquivo para um determinado local do servidor
- A criação de arquivos WAR é basicamente uma tarefa de copiar e compactar
  - Usaremos o Ant para implantar as operações
- Reimplantar um arquivo WAR é simples no JBoss, mas é trabalhoso no Tomcat
  - Requer a remoção do WAR e do diretório criado
  - Requer o reinício do servidor

  por este motivo, continuaremos a copiar os arquivos para o servidor nos próximos módulos, apesar de criar o WAR

Web Archive

- *Utilizável no Tomcat e também em servidores J2EE*
- *Permite criação de novo contexto automaticamente*
- *Coloque JAR contendo estrutura de um contexto no diretório de deployment (webapps, no Tomcat)*
  - *O JAR deve ter a extensão .WAR*
  - *O JAR deve conter WEB-INF/web.xml válido*

Exemplo - aplicação: `http://servidor/sistema/`



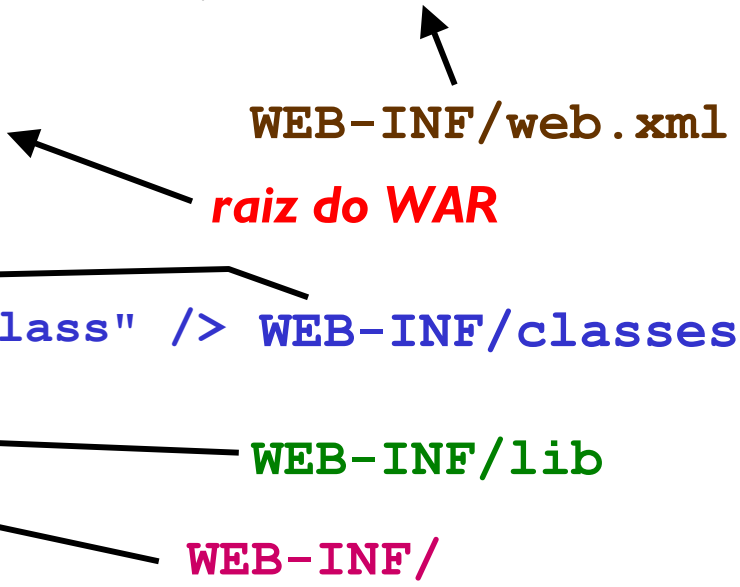
Como criar um WAR

- O mesmo WAR que serve para o Tomcat, serve para o JBoss, Weblogic, WebSphere, etc.
  - Todos aderem à mesma especificação
- Há várias formas de criar um WAR
  - Usando o deploytool do Tomcat.
  - Usando um aplicativo tipo WinZip
  - Usando uma ferramenta JAR:
    jar -cf arquivo.war -C diretorio_base .
  - Usando a ferramenta packager do kit J2EE:
    packager -webArchive <opções>
  - Usando a tarefa <jar> ou <war> no Ant

WAR criado pelo Ant

- Pode-se criar WARs usando a tarefa <jar> ou <war>
  - Com <jar> você precisa explicitamente definir seus diretórios WEB-INF, classes e lib (usando um <zipfileset>, por exemplo) e copiar os arquivo web.xml, suas classes e libs.
  - Com <war> você pode usar o atributo webxml, que já coloca o arquivo web.xml no lugar certo, e outros elementos de um war:

```
<war warfile="bookstore.war" webxml="etc/metainf.xml">
   <fileset dir="web" >
      <include name="*.html />
      <include name="*.jsp" />
   </fileset>
   <classes dir="${build}" >
      <include name="database/*.class" />
   </classes>
   <lib dir="${lib.dir}" />
   <webinf dir="${etc.dir}" />
</war>
```



  - Veja o manual do Ant para outros exemplos e detalhes

Configuração externa do WAR (servidores J2EE)

- Configuração externa ao WAR pode ser feita quando WAR é acrescentado em um arquivo EAR e funciona em servidores J2EE (JBoss, por exemplo)
  - EAR é JAR comum com arquivo application.xml no seu META-INF
- O arquivo application.xml do EAR deve conter

```
<application>
  <module>
    <web>
      <web-uri>mywebapp.war</web-uri>
      <context-root>/myroot</context-root>
    </web>
  </module>
</application>
```

- A aplicação agora é acessada via

```
http://servidor/myroot
```

Deployment e execução

- Depende do servidor
  - No JBoss, copie o WAR ou EAR para o diretório deploy do servidor para implantar o serviço. Remova o WAR para tirar o serviço do ar
  - No Tomcat, copie o WAR para o diretorio webapps e reinicie o servidor. Para remover o serviço, desligue o Tomcat, apague o diretório criado e o WAR.
- Para executar, as regras são as mesmas que uma aplicação comum. Acesse o contexto raiz via Web
  - http://servidor/nome-do-contexto/
  - http://servidor/nome-do-contexto/index.jsp
  - http://servidor/nome-do-contexto/subcontexto/aplicacao
  - http://servidor/nome-do-contexto/servlet/pacote.Classe

Exercícios

- 1. Crie um alvo no Ant que gere WARs para todas as aplicações que você já criou até o momento
- 2. Crie outro alvo que faça o deployment automático, copiando o WAR para o diretório correto
  - Guarde as propriedades de deployment em um arquivo externo build.properties para fácil alteração
- 3. Crie um alvo que faça o undeploy
  - Remova o diretório (com mesmo nome de contexto)
  - Remova o WAR

  Esta última tarefa pode falhar caso o Tomcat esteja no ar. Será preciso pará-lo antes (mas não crie uma tarefa para isto)

helder@acm.org

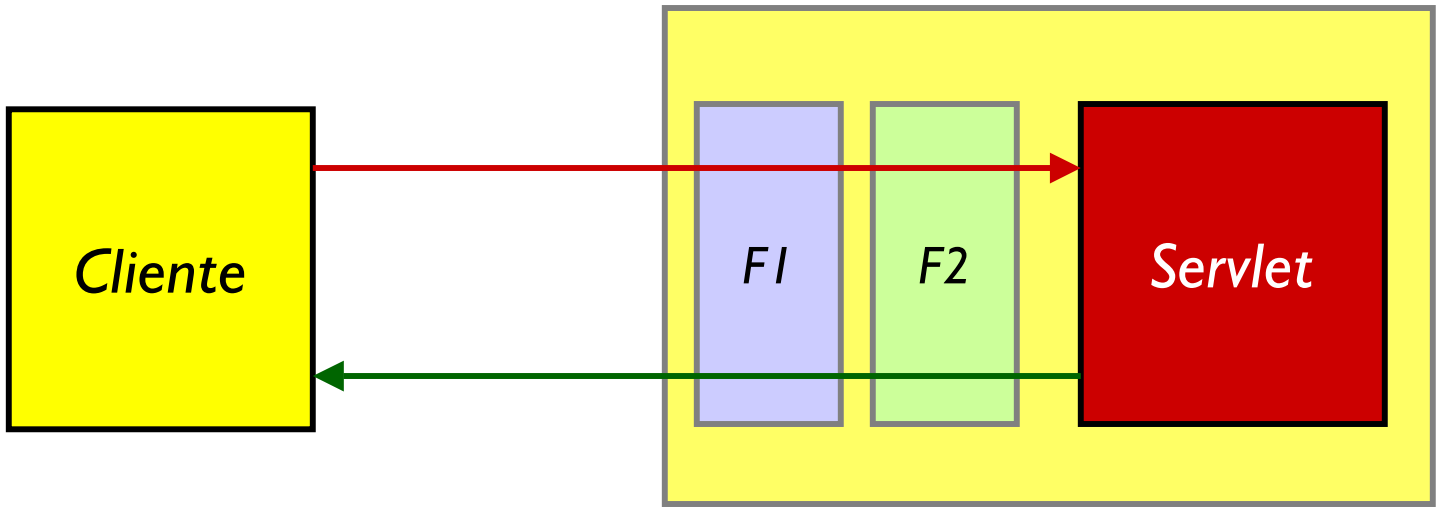
argonavis.com.br

6 Filtros

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

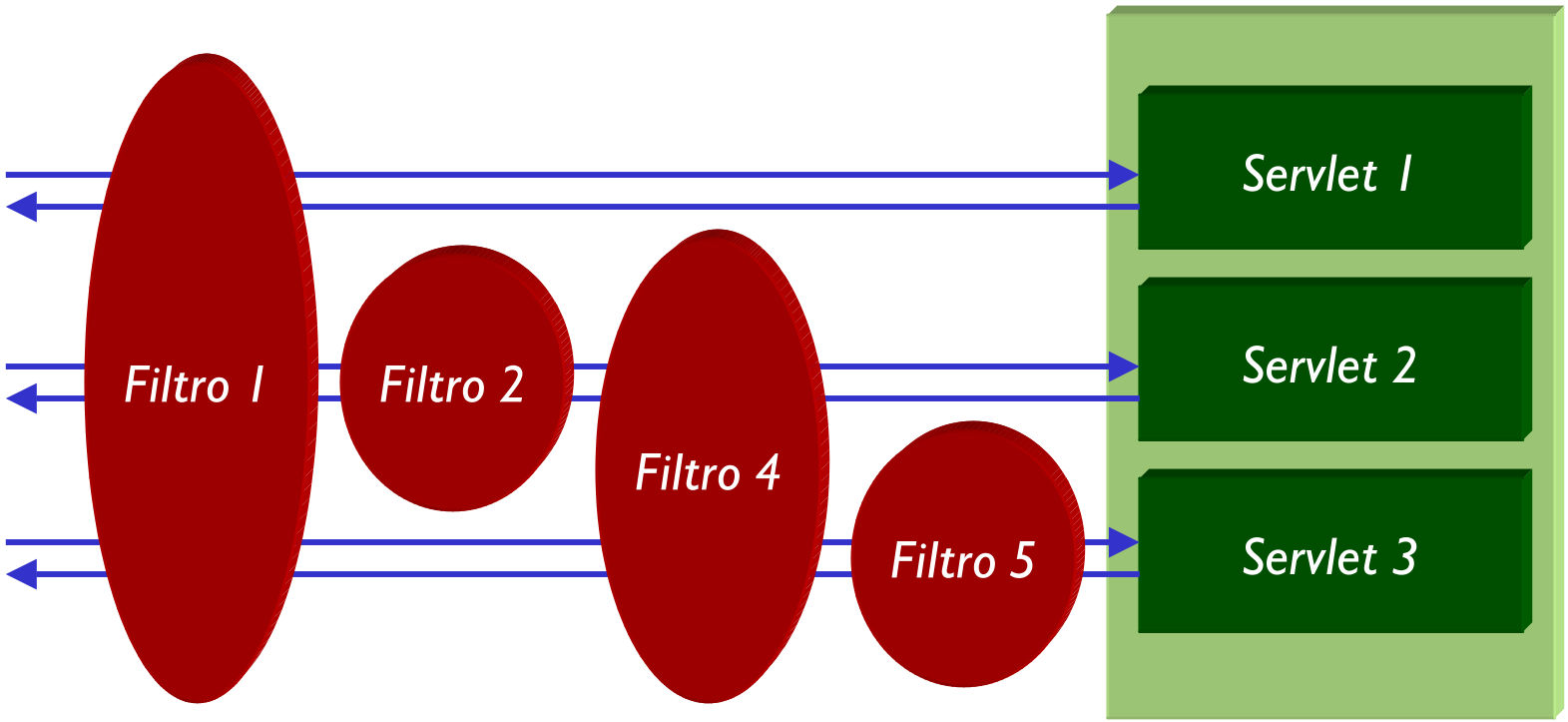
O que são Filtros

- Um *filtro* é um *componente Web* que reside no servidor
  - *Intercepta* as requisições e respostas no seu caminho até o servlet e de volta ao cliente
  - Sua existência é ignorada por ambos. É totalmente *transparente* tanto para o cliente quanto para o servlet
  - Suportado desde a versão 2.3 da especificação de Servlets
- Filtros podem ser concatenados em uma *corrente*
  - Neste cenário, as requisições são interceptadas em uma ordem e as respostas em ordem inversa



Filtros

- *Um filtro pode realizar diversas transformações, tanto na resposta como na requisição antes de passar esses objetos adiante (se o fizer)*
- *Filtros podem ser reutilizados em vários servlets*



Para que servem?

- Filtros permitem
  - Tomada de decisões: podem decidir se repassam uma requisição adiante, se redirecionam ou se enviam uma resposta interrompendo o caminho normal da requisição
  - Tratamento de requisições e respostas: podem empacotar uma requisição (ou resposta) em outra, alterando os dados e o conteúdo dos cabeçalhos
- Aplicaçoes típicas
  - Autenticação
  - Conversão de caracteres, MIME types, tokenizing
  - Conversão de imagens, compressão e decompressão
  - Criptografia
  - Transformação XSLT

Como funcionam?

- Quando o container recebe uma requisição, ele verifica se há um filtro associado ao recurso solicitado. Se houver, a requisição é roteada ao filtro
- O filtro, então, pode
  1. Gerar sua própria resposta para o cliente
  2. Repassar a requisição, modificada ou não, ao próximo filtro da corrente, se houver, ou ao recurso final, se ele for o último filtro
     - Rotear a requisição para outro recurso
- Na volta para o cliente, a resposta passa pelo mesmo conjunto de filtros em ordem inversa

API: Interfaces Filter, FilterConfig, FilterChain

- javax.servlet.Filter
  - void init(FilterConfig),
  - void doFilter(ServletRequest, ServletResponse, FilterChain)
  - void destroy()
- javax.servlet.FilterConfig
  - String getFilterName()
    String getInitParameter(String name)
  - Enumeration getInitParameterNames()
  - ServletContext getServletContext()
- javax.servlet.FilterChain
  - void doFilter(ServletRequest, ServletResponse)

API: Classes empacotadoras

- Úteis para que filtros possam trocar uma requisição por outra
  - Uma subclasse dessas classes empacotadoras pode ser passada em uma corrente de filtros no lugar da requisição ou resposta original
  - Métodos como *getParameter()* e *getHeader()* podem ser sobrepostos para alterar parâmetros e cabeçalhos
- No pacote *javax.servlet*
  - *ServletRequestWrapper implements ServletRequest*: implementa todos os métodos de ServletRequest e pode ser sobreposta para alterar o request em um filtro
  - *ServletResponseWrapper implements ServletResponse*: implementa todos os métodos de ServletResponse
- No pacote *javax.servlet.http*
  - *HttpServletRequestWrapper* e *HttpServletResponseWrapper*: implementam todos os métodos das interfaces correspondentes, facilitando a sobreposição para alteração de cabeçalhos, etc.

Como escrever um filtro simples

1. Escreva uma classe implementando a interface Filter e todos os seus métodos
   - init(FilterConfig)
   - doFilter(ServletRequest, ServletResponse, FilterChain)
   - destroy()
2. Compile usando o JAR da Servlet API
3. Configure o filtro no deployment descriptor (web.xml) usando os elementos <filter> e <filter-mapping>
   - Podem ser mapeados a URLs, como servlets
   - Podem ser mapeados a servlets, para interceptá-los
   - A ordem dos mapeamentos é significativa
4. Implante o filtro da maneira usual no servidor

Filtro simples que substitui servlet

```
package j550.filtros;
import java.io.*;
import javax.servlet.*;

public class HelloFilter implements Filter {
  private String texto;
  public void init(FilterConfig config) {
    texto = config.getInitParameter("texto");
  }
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain filterChain)
                        throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.println("<HTML><HEAD><TITLE>Filter Response");
      out.println("</TITLE></HEAD><BODY>");
      out.println("<H1>Filter Response</H1>");
      out.println("<P>" + texto);
      out.println("</BODY></HTML>");
      out.close();
   }
   public void destroy() {}
}
```

Configuração

- *Os elementos <filter> e <filter-mapping> são quase idênticos aos equivalentes para <servlet>*
  - *A diferença é que <filter-mapping> é usado também para associar filtros a servlets, na ordem em que aparecem*
- *Filtro simples, que substitui um servlet*

```
<filter>
    <filter-name>umFiltro</filter-name>
    <filter-class>j550.filtros.HelloFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>umFiltro</filter-name>
    <url-pattern>/filtro</url-pattern>
</filter-mapping>
```

- *Filtro que intercepta um servlet*

```
<filter-mapping>
    <filter-name>umFiltro</filter-name>
    <servlet-name>umServlet</servlet-name>
</filter-mapping>
```

Filtros "de verdade"

- *Filtros úteis podem ser encadeados em uma corrente. Para que isto seja possível, devem chamar doFilter() no objeto FilterChain - parâmetro no seu próprio doFilter()*

```
public void doFilter(...req,...res, FilterChain chain) {
    ...
    chain.doFilter(req, res);
    ...
}
```

- *Antes da chamada ao doFilter(), o filtro pode processar a requisição e alterar ou substituir os objetos ServletRequest e ServletResponse ao passá-los adiante*

```
ServletRequest newReq = new ModifiedRequest(...);
chain.doFilter(newReq, res);
```

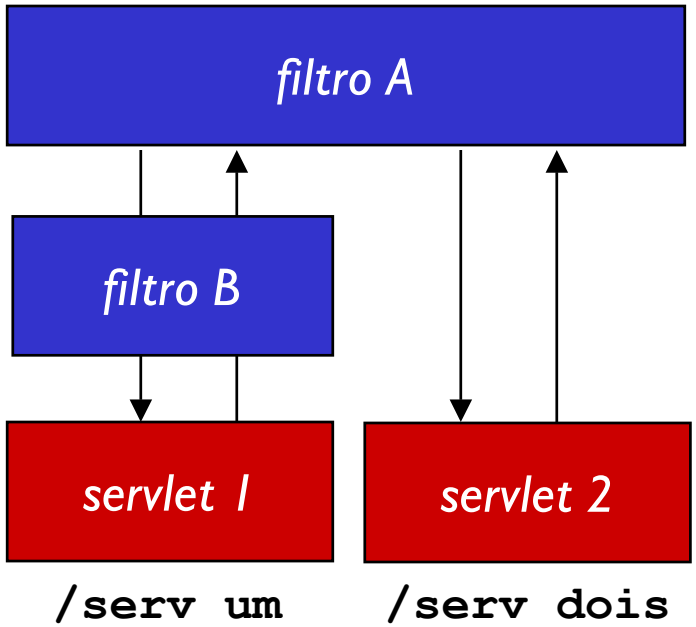
Estende ServletRequestWrapper

- *Na volta, opera sobre a resposta e pode alterá-la*

Configuração da corrente

- *A corrente pode ser configurada com definição das instâncias de filtros e mapeamentos em ordem*

```
<filter>
    <filter-name>filtroA</filter-name>
    <filter-class>j550.filtros.FilterA</filter-class>
</filter>
<filter>
    <filter-name>filtroB</filter-name>
    <filter-class>j550.filtros.FilterB</filter-class>
</filter>
<filter-mapping>
    <filter-name>filtroA</filter-name>
    <url-pattern>/serv_um</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>filtroA</filter-name>
    <servlet-name>servlet2</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>filtroB</filter-name>
    <servlet-name>servlet1</servlet-name>
</filter-mapping>
```



Filtros que tomam decisões

- *Um filtro pode ler a requisição e tomar decisões como transformá-la, passá-la adiante ou retorná-la*

```
public void doFilter(.. request, ... response, ... chain) {
   String param = request.getParameter("saudacao");
   if (param == null) {
      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      out.println("<h1>Erro!</h1>");
   } else if (param.equals("Bonjour!")) {
      class MyWrapper extends ServletRequestWrapper { // ...
        public String getParameter(String name) {
          if (name.equals("saudacao")) {
             return "Bom Dia";
          }
        }
      }
      ServletRequest myRequest = new MyWrapper(request);
      chain.doFilter(myRequest, response);
   } else { chain.doFilter(request, response); }
}
```

Classe interna (o construtor (abaixo) foi omitido da declaração por falta de espaço)

Wrappers

- *Sobrepondo um HttpServletRequest*

```
public class MyServletWrapper
                              extends HttpServletRequestWrapper {
   public MyServletWrapper(HttpServletRequest req) {
      super(req);
   }
   public String getParameter(String name) {
      return super.getParameter(name).toUpperCase();
   }
}
```

- *Usando Wrappers em servlets HTTP*

```
HttpServletRequest req = (HttpServletRequest)request;
HttpServletResponse res = (HttpServletResponse)response;
HttpServletRequest fakeReq = new MyServletWrapper(req);
HttpServletResponse fakeRes = new TestResponse(res);

chain.doFilter(fakeReq, fakeRes);
```

Observações importantes

- Para filtros usados com servlets HTTP, o request e response passados são HttpServletRequest e HttpServletResponse
  - Wrappers devem estender as classes que implementam essas interfaces
- Filtros não são chamados quando o recurso que interceptam for chamado através de um RequestDispatcher
  - O recurso é acessado diretamente sem filtragem
  - Isto ocorre para evitar loops infinitos
- Filtros associados a páginas de erro também não são chamados

Exercícios

1. Escreva um filtro simples que leia a requisição e verifique se ela contém os parâmetros usuario e senha
   - Se não tiver, repasse a requisição para a página erro.html
   - Se tiver, abra o arquivo usuarios.txt usando a classe Properties. Ele possui uma lista de nome=senha, um por linha. Veja se o usuário coincide com a senha. Se sim, chame o próximo filtro. Se não, redirecione para acessoNegado.html
   - Associe o filtro a um servlet qualquer (o SimpleServlet, por exemplo)
   - Acesse o servlet e verifique que ele passa pelo filtro
- 2. Escreva dois RequestWrappers que encapsulam HttpServletRequest e sobrepõem o método getParameter()
   - Use o primeiro em um filtro chamado UpperCaseFilter, que coloque os valores de todos os parâmetros em caixa-alta.
   - O segundo, ReverseFilter, deve inverter o texto dos parâmetros
   - Coloque os dois em cascata apontando para um servlet simples.

Exercícios (2)

- 3. Escreva um filtro chamado StyleFilter para alterar o estilo do texto passado como parâmetro
  - a) O `estilo` será definido como parâmetro inicial (<init-param>) do filtro em linguagem CSS. Se for recebido um texto original
    `texto`
    o filtro deve devolver
    `<span style='estilo'>texto</span>`
- 4. Configure duas instâncias do filtro (boldFilter e blueFilter)
  - a) Uma passando o estilo: "`font-weight: bold`"
  - b) Outra passando o estilo: "`color: blue`"
- 5. Crie duas instâncias e mapeamentos no web.xml para SimpleServlet e associe-os às instâncias de filtros
  - a) Mapeie uma instância à URL /bluebold e outra a /blue
  - b) Associe boldFilter e blueFilter, nesta ordem, a /bluebold
  - c) Associe blueFilter e UpperCaseFilter (exercício 2) a /blue
  - d) Verifique a ordem de chamada no código-fonte gerado no browser

helder@acm.org

argonavis.com.br

Segurança e Controle de erros

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Assuntos abordados

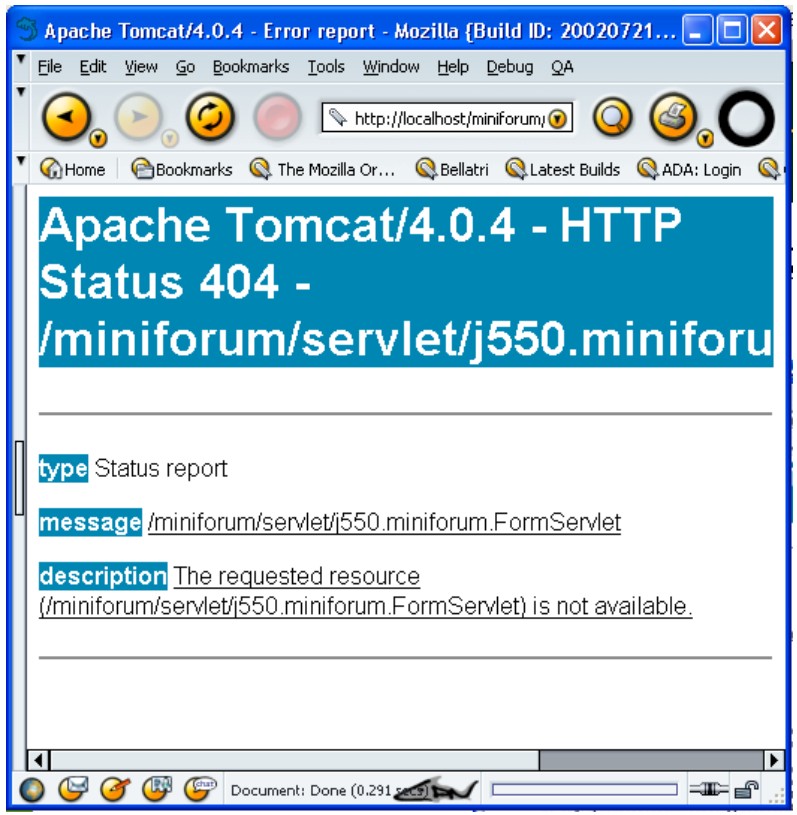
- Este módulo trata de dois assuntos
  - Como mapear erros HTTP e exceções Java a servlets ou páginas HTML e JSP, criando respostas personalizadas e servlets que possam lidar com situações de erro
  - Como implementar autenticação e autorização baseada em perfis do usuário usando servlets, permitindo que determinados recursos fiquem protegidos por senha e com acesso limitado a usuários que pertençam a determinado perfil

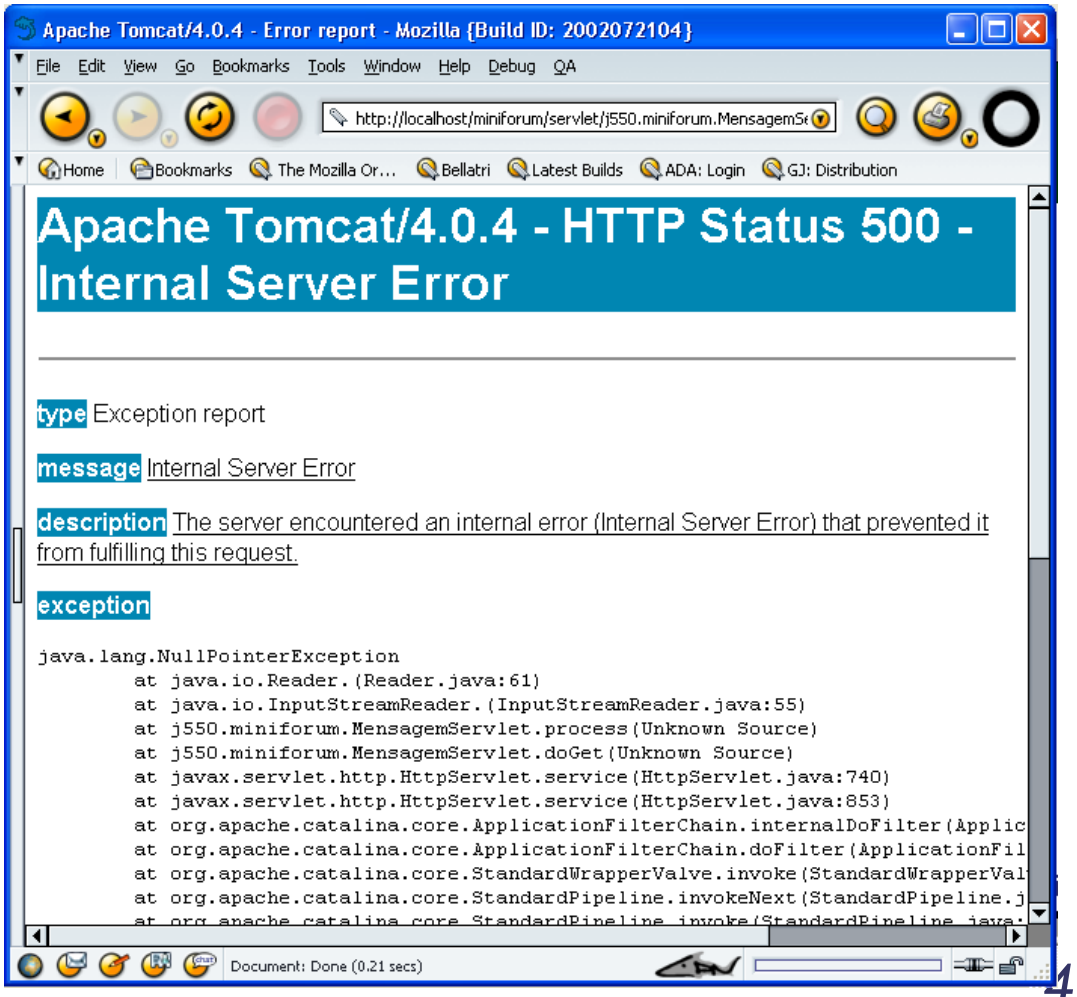
Erros

- Dois tipos de erros podem ocorrer em servlets
  - Erros HTTP
  - Exceções
- Erros HTTP são identificados pelo servidor através de um código de status
  - Códigos de status que começam com 1, 2 ou 3 não indicam erro
  - Códigos que começam com 4 (404, 401, etc.) indicam erro originado no cliente (que, em tese, poderia ser corrigido pelo browser - ex: desviar para outra página)
  - Códigos que começam com 5 (500, 501, etc.) indicam erro no servidor (não adianta o browser tentar evitá-lo)

Comportamento default

- *Quando acontece um erro do servidor, causado ou não por uma exceção, o servidor mostra uma página default*





Páginas de erro

- Configurando-se o web.xml, pode-se estabelecer páginas de erro customizadas para cada código de erro HTTP e cada exceção ocorrida
- O exemplo abaixo usa <error-page> para informar uma página HTML que irá tratar erros 404 e um servlet que irá receber FileNotFoundException

```
<error-page>
  <error-code>404</error-code>
  <location>/notFound.html</location>
</error-page>

<error-page>
  <exception-type>java.io.FileNotFoundException</exception-type>
  <location>/servlet/j550.error.IOExceptionServlet</location>
</error-page>
```

Pode haver qualquer número de elementos <error-page> mas apenas um para cada tipo de exceção específica ou código de erro HTTP

Se você define <error-page> de exceções não defina um <error-page> para o erro HTTP 500: este é o código para todas as exceções!

Atributos especiais

- *Destinos de erro recebem dois atributos na requisição que podem ser lidas se forem páginas geradas dinamicamente (servlet ou JSP) para melhor exibir informações de erro*
  - *Nome: javax.servlet.error.request_uri. Tipo: String*
  - *Nome: javax.servlet.error.exception. Tipo: Throwable*
- *A exceção recebida corresponde à exceção embutida dentro de ServletException (a causa)*

```
public void doGet(... request, ... response) {
  Throwable exception = (Throwable)
    request.getAttribute("javax.servlet.error.exception");
  String urlCausadora = (String)
    request.getAttribute("javax.servlet.error.request_uri");

  // Faça alguma coisa com os dados recebidos

}
```

Recursos de segurança

- A especificação Servlet 2.3 permite a configuração de segurança em dois domínios
  - Autenticação: o processo de verificação da identidade do usuário que solicita um recurso - quem é?
  - Autorização: processo de verificação das permissões que um usuário autenticado possui - o que ele pode fazer?
- Os dois recursos podem ser configurados para cada contexto no web.xml definindo
  - Login Configuration: configuração de métodos de autenticação utilizados
  - Security Roles: perfis de usuário para autorização
  - Security Constraint: URLs e métodos de acesso permitidos e perfis de segurança necessários para acessá-los

Configuração de Login

- *Escolha uma dentre quatro técnicas de autenticação*
  - *BASIC: mostra uma janela do browser que recebe nome e senha. Os dados são enviados em conexão insegura*
  - *FORM: igual a BASIC mas em vez de usar uma janela do browser, pemite o uso de um formulário HTML*
  - *DIGEST: usa criptografia fraca para enviar os dados*
  - *CLIENT-CERT: requer certificado X-509 para funcionar e usa criptografia forte (128-bit)*
- *BASIC, FORM e DIGEST são seguros se usados com SSL*

```
<login-config>
  <auth-method>
    BASIC
  </auth-method>
</login-config>
```

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/form.html</form-login-page>
    <form-error-page>/erro.html</form-error-page>
  </form-login-config>
</login-config>
```

Autorização: perfis de usuário

- *Uma vez definida a forma de autenticação, é preciso definir perfis de usuário habilitados a acessar os recursos*
- *A declaração de perfis de usuários é feita no web.xml usando <security-role>. A associação desses perfis com usuários reais é dependente de servidor*

```
<security-role>
   <role-name>administrador</role-name>
</security-role>

<security-role>
   <role-name>membro</role-name>
</security-role>

<security-role>
   <role-name>visitante</role-name>
</security-role>
```

Associação de perfis com usuário no Tomcat

- *Para definir domínios de segurança* ***no Tomcat*** *veja a documentação do servidor sobre security realms: http://localhost:8080/tomcat-docs/realm-howto.html*
  - *Há três níveis diferentes de configuração. Um grava os dados em banco relacional (JDBC Realm), outro em LDAP via JNDI (JNDI Realm) e o mais simples usa um par de arquivos (Memory Realm)*
- *Para usar Memory Realm, localize o arquivo tomcat-users.xml no diretório conf/ em* ***$TOMCAT_HOME*** *e acrescente os usuários, senhas e perfis desejados*

```
<tomcat-users>
  <user name="einstein" password="e=mc2" roles="visitante" />
  <user name="caesar"   password="rubicon"
                        roles="administrador, membro"  />
  <user name="brutus"   password="tomcat" roles="membro" />
</tomcat-users>
```

Web-Resource Collection

- *A coleção de recursos Web protegidos e os métodos de acesso que podem ser usados para acessá-los é definido em um bloco <web-resource-collection> definido dentro de <security-constraint>*
  - *Inclui URL-patterns <url-pattern> que indicam quais os mapeamentos que abrangem a coleção*
  - *Inclui um <http-method> para cada método permitido*

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Seção de Assinantes</web-resource-name>
    <url-pattern>/membros/*</url-pattern>
    <url-pattern>/preferencias/config.jsp</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  ...
</security-constraint>
```

Security Constraint

- *O Web Resource Collection faz parte do Security Constraint que associa perfis de usuário (papéis lógicos) à coleção*

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Seção de Assinantes</web-resource-name>
    <url-pattern>/membros/*</url-pattern>
    <url-pattern>/preferencias/config.jsp</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>

  <auth-constraint>
    <role-name>administrador</role-name>
  </auth-constraint>
</security-constraint>
...
<security-constraint>
        ... outras coleções e constraint ...
</security-constraint>
```

Form-based Login

- *Para autenticação por formulário, é preciso definir no <login-config> FORM como <auth-method>*
- *Adicionalmente, é informada a página que implementa o formulário. Esta página deve conter um elemento <FORM> HTML com algumas restrições*
  - *Atributo ACTION de <FORM> deve conter: j_security_check*
  - *Campo <INPUT> do nome deve conter: j_username*
  - *Campo <INPUT> da senha deve conter: j_password*

```
<form action="j_security_check" method="POST">
  <p>Digite seu nome de usuário:
  <input type="text" name="j_username">
  <p>Digite sua senha:
  <input type="password" name="j_password">
  <input type="submit">
</form>
```

Exercícios

- 1. Mapeamentos de erro: crie uma página especial para erros 404 na sua aplicação.
- 2. Mapeamentos de exceção: redirecione qualquer Exception para uma página especial que imprima a classe da exceção ocorrida
  - Use getClass() para obter o nome da classe do objeto.
- 3. Configure os usuários e perfis mostrados nos exemplos no seu Tomcat e implemente um login BASIC para que
  - Apenas membros possam mandar mensagens para o forum
  - Apenas usuários visitantes registrados possam vê-las
  - Qualquer um possa ver a página inicial (index.html)
  - Administradores tenham acesso a todo o sistema
- 4. Crie um formulário de login em uma página HTML e altere o programa do exercício anterior para utilizá-lo.

Exercício

- *3. Implemente uma tela de login para a aplicação da Loja Virtual alterando LojaServlet para que peça além do e-mail, uma senha*
  - *Se o usuário for administrador, redirecione-o para AdminLojaServlet*
  - *Se usuário foi registrado, redirecione-o para ComprasServlet*
  - *Se usuário não for registrado, envie-o para página onde possa se registrar**
  - *Use BASIC ou FORM para autenticação*

** Não precisa implementar o registro. Se desejar, manipule o arquivo XML do Tomcat usando a API JAXP (javax.xml) ou use o JDBC Security Realm do Tomcat para guardar as informações em um banco de dados*

helder@acm.org

argonavis.com.br

J550

Integração com Bancos de Dados

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Objetivos

- *Este módulo apresenta estratégias para conectar servlets com a camada de dados usando um DAO - Data Access Object*
- *Um DAO é a alternativa ideal porque isola o servlet do conhecimento dos detalhes da implementação de banco de dados usada*
  - *Pode-se trocar a implementação por outra (usando XML, arquivos, outros bancos, etc.)*
- *Para escrever um DAO para um banco relacional, será necessário usar a interface JDBC, mas o servlet não terá nenhuma linha de código JDBC*

Acesso a bancos de dados

- *Servlets são aplicações Java e, como qualquer outra aplicação Java, podem usar JDBC e integrar-se com um banco de dados relacional*
- *Pode-se usar java.sql.DriverManager e obter a conexão da forma tradicional*

```
Class.forName("nome.do.Driver");
Connection con =
     DriverManager.getConnection("url", "nm", "ps");
```

- *Pode-se obter as conexões de um pool de conexões através de javax.sql.DataSource via JNDI (J2EE)*

```
DataSource ds = (DataSource)ctx.lookup("jbdc/Banco");
Connection con = ds.getConnection();
```

Servlet que faz um SELECT em banco

- *Para tornar um servlet capaz de acessar bancos de dados, basta incluir código JDBC dentro dele*

```
import java.sql.*; // ... outros pacotes
public class JDBCServlet extends HttpServlet {
   String jdbcURL;   String driverClass;
   String nome = ""; String senha = "";  Connection con;
   public void init() {
      // ... obter os dados via initParameter()
      try {
        Class.forName(driverClass);
        con = DriverManager.getConnection(jdbcURL, nome, senha);
     } catch (Exception e) { throw new UnavailableException(e); }
   }
   public void doGet(... request, ... response) ... {
      try {  // ... codigo de inicializacao do response
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM nomes");
        out.println("<table border>");
        while(rs.next()) {
          out.println("<tr><td>"+rs.getString("nome")+"</tr></td>");
        } // ... resto do codigo
      } catch (SQLException e) {
        throw new ServletException(e);
      }
   }
}
```

Servlet que usa DAO

- *Misturar código de servlet com banco de dados não é uma boa idéia. O ideal é disponibilizar uma interface independente de banco para o servlet*

```
public class DataAccessServlet extends HttpServlet {
   DataAccessDAO dao;
   public void init() {
      dao = JDBCDataAccessDAO.getInstance();
   }
   public void doGet(... request, ... response) ... {
      try {  // ... codigo de inicializacao do response
        String[] nomes = dao.listarTodosOsNomes();
        out.println("<table border>");
        for(int i = 0; i < nomes.length; i++) {
          out.println("<tr><td>"+nomes[i]+"</tr></td>");
        } // ... resto do codigo
      } catch (AcessoException e) {
        throw new ServletException(e);
      }
   }
}
```

Exemplo de DAO

- *O DAO é um objeto que isola o servlet da camada de dados, deixando-o à vontade para mudar a implementação*
- *Faça-o sempre implementar uma interface Java*

```
import java.sql.*; // ... outros pacotes
public class JDBCDataAccessDAO implements DataAccessDAO {
   // ...
   public void JDBCDataAccessDAO() {
      try {
        Class.forName(driverClass);
        con = DriverManager.getConnection(jdbcURL, nome, senha);
     } catch (Exception e) { ... }
   }
   public String[] listarTodosOsNomes() throws AcessoException {
      try {
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM nomes");
        List nomesList = new ArrayList();
        while(rs.next()) {
          nomesList.add(rs.getString("nome"));
        }
        return (String[]) nomesList.toArray(new String[nomesList.size()];
      } catch (SQLException e) {
        throw new AcessoException(e);
      }
   }
}
```

Exercicio

- 1. Crie um DAO para isolar a gravação de produtos (atualmente gravados em arquivo) do resto da aplicação
  - Todo o conhecimento sobre gravação deve ficar no DAO
  - Use uma interface ou classe abstrata ProdutosDAO e implemente o DAO em uma classe ArquivoProdutosDAO
- 2. Crie uma tabela Produto no seu banco de dados e implemente as funções do DAO usando JDBC
  - Crie uma classe JDBCProdutosDAO
  - Passe os dados de configuração do banco como parâmetros de contexto e recupere-os no seu init()
- 3. Implemente, na sua aplicação, uma opção administrativa para remover produtos e outra para editar produtos
  - Crie as novas funções no DAO e implemente-as com JDBC
- 4. Implemente a função adicionar autor e adicionar editor do DAO da aplicação Biblioteca disponível no diretório cap08

helder@ibpinet.net

argonavis.com.br

J550

JavaServer Pages

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Objetivos

- Este módulo apresenta tudo o que é necessário para implementar servlets escrevendo JavaServer Pages
  - Sintaxe dos marcadores JSP e objetos
  - Funcionamento
  - Como implantar e depurar
- Tudo o que vale para servlets continua valendo para JavaServer Pages
  - Um JSP é um servlet durante a execução
  - Escrever código em JSP é como escrever código dentro do doPost() ou doGet() de um servlet com os objetos response, request, out, session e outros já definidos
  - Um JSP, depois de carregado, é tão veloz quando um servlet
  - É mais fácil escrever e implantar, mas é mais difícil depurar

Problemas de servlets

- *Servlets forçam o programador a embutir código HTML dentro de código Java*
  - *Desvantagem se a maior parte do que tem que ser gerado é texto ou código HTML estático*
  - *Mistura as coisas: programador tem que ser bom Web Designer e se virar sem ferramentas de Web Design*

```
Date hoje = new Date();
out.println("<body>");
out.println("<p>A data de hoje é "+hoje+".</p>");
out.println("<body>");
```

HojeServlet.java

- *Uma solução inteligente é escrever um arquivo de template*

```
<body>
<p>A data de hoje é <!--#data#-->.</p>
<body>
```

template.html

Usando templates em servlets

- *Tendo-se um template, é preciso criar um mecanismo eficiente para processá-los*
  - *No exemplo mostrado, pode-se ler o arquivo seqüencialmente , jogando tudo na saída até achar a seqüência "<!--#"*
  - *Depois, interpretar o "comando", gera-se o código relacionado com ele e prossegue-se na leitura e impressão do resto do documento*
  - *Há várias formas de implementar. O código abaixo usa o pacote javax.util.regex para localizar os comandos e fazer a substituição*

```
Date hoje = new Date();
String pagina = abreHTML("template.html");
Pattern p = Pattern.compile("<!--#data#-->");
Matcher m = p.matcher(pagina);
m.replaceAll(hoje);
out.println(m.toString());
```

HojeServlet.java

- *Com o tempo, define-se um vocabulário e procura-se fazer o processador de templates cada vez mais reutilizável*

O que são JavaServer Pages (JSP)

- JSP é uma tecnologia padrão, baseada em templates para servlets. O mecanismo que a traduz é embutido no servidor.
- Há várias outras alternativas populares
  - Apache Cocoon XSP: baseado em XML (xml.apache.org/cocoon)
  - Jakarta Velocity (jakarta.apache.org/velocity)
  - WebMacro (www.webmacro.org)
- Solução do problema anterior usando templates JSP

```
<body>
<p>A data de hoje é <%=new Date() %>.</p>
<body>
```

hoje.jsp

- Em um servidor que suporta JSP, processamento de JSP passa por uma camada adicional onde a página é transformada (compilada) em um servlet
- Acesso via URL usa como localizador a própria página

Exemplos de JSP

- *A forma mais simples de criar documentos JSP, é*
  1. *Mudar a extensão de um arquivo HTML para .jsp*
  2. *Colocar o documento em um servidor que suporte JSP*
- *Fazendo isto, a página será transformada em um servlet*
  - *A compilação é feita no primeiro acesso*
  - *Nos acessos subseqüentes, a requisição é redirecionada ao servlet que foi gerado a partir da página*
- *Transformado em um JSP, um arquivo HTML pode conter blocos de código (scriptlets):* `<% ... %>` *e expressões* `<%= ... %>` *que são os elementos mais frequentemente usados*

```
<p>Texto repetido:
<% for (int i = 0; i < 10; i++) { %>
  <p>Esta é a linha <%=i %>
<% }%>
```

Exemplo de JSP

```
<%@ page import="java.util.*" %>
<%@ page import="j2eetut.webhello.MyLocales" %>
<%@ page contentType="text/html; charset=iso-8859-9" %>
<html><head><title>Localized Dates</title></head><body bgcolor="white">
<a href="index.jsp">Home</a>
<h1>Dates</h1>
<jsp:useBean id="locales" scope="application"
                          class="j2eetut.webhello.MyLocales"/>
<form name="localeForm" action="locale.jsp" method="post">
<b>Locale:</b><select name=locale>
<%
   Iterator i = locales.getLocaleNames().iterator();
   String selectedLocale = request.getParameter("locale");
   while (i.hasNext()) {
      String locale = (String)i.next();
      if (selectedLocale != null && selectedLocale.equals(locale) ) {
         out.print("<option selected>" + locale + "</option>");
      } else { %>
         <option><%=locale %></option>
<%    }
   } %>
</select><input type="submit" name="Submit" value="Get Date">
</form>
<p><jsp:include page="date.jsp" flush="true" />
</body></html>
```

diretivas

bean

scriptlet

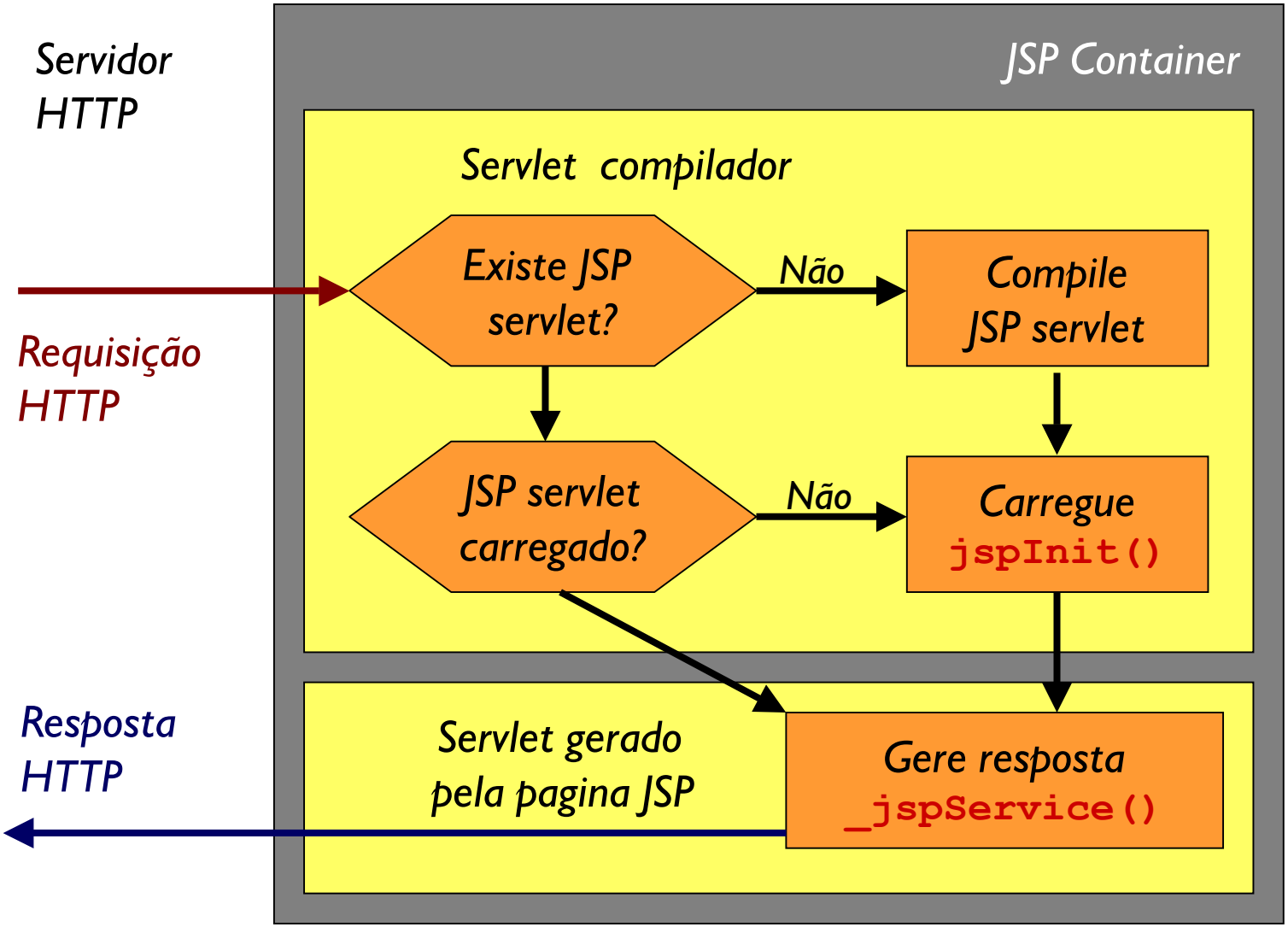
expressão

ação

Ciclo de vida

- Quando uma requisição é mapeada a uma página JSP, o container
  - Verifica se o servlet correspondente à página é mais antigo que a página (ou se não existe)
  - Se o servlet não existe ou é mais antigo, a página JSP será compilada para gerar novo servlet, em seguida, a requisição é repassada ao servlet
  - Se o servlet está atualizado, a requisição é redirecionada para ele
- Deste ponto em diante, o comportamento equivale ao ciclo de vida do servlet, mas os métodos são diferentes
  - Se o servlet ainda não estiver na memória, ele é instanciado, carregado e seu método `jspInit()` é chamado
  - Para cada requisição, seu método `_jspService(req, res)` é chamado. Ele é resultado da compilação do corpo da página JSP
  - No fim da vida, o método `jspDestroy()` é chamado

Como funciona JSP



Sintaxe dos elementos JSP

- *Podem ser usados em documentos de texto (geralmente HTML ou XML)*
- *Todos são interpretados no servidor (jamais chegam ao browser)*
  - *diretivas:* `<%@ ... %>`
  - *declarações:* `<%! ... %>`
  - *expressões:* `<%= ... %>`
  - *scriptlets:* `<% ... %>`
  - *comentários:* `<%-- ... --%>`
  - *ações:* `<jsp:ação ... />`
  - *custom tags:* `<prefixo:elemento ... />`

(a) diretivas

- *Contém informações necessárias ao processamento da classe do servlet que gera a página JSP*
- *Sintaxe :*

```
<%@ diretiva atrib1 atrib2 ... %>
```

- *Principais diretivas:*
  - **page**: *atributos relacionados à página*
  - **include**: *inclui outros arquivos na página*
  - **taglib**: *declara biblioteca de custom tags usada no documento*
- *Exemplos*

```
<%@ page  import="java.net.*, java.io.*"
          session="false"
          errorPage="/erro.jsp" %>
<%@ include file="navbar.jsp" %>
```

(a) diretiva page

- *Atributos de* `<%@page ... %>`

`info="Texto informativo"`	*default: nenhum*
`language="java"`	*(default)*
`contentType="text/html; charset=ISO-8859-1"`	*(default)*
`extends="acme.FonteJsp"`	*default: nenhum*
`import="java.io.*, java.net.*"`	*default: java.lang*
`session="true"`	*(default)*
`buffer="8kb"`	*(default)*
`autoFlush="true"`	*(default)*
`isThreadSafe="true"`	*(default)*
`errorPage="/erros/404.jsp"`	*default: nenhum*
`isErrorPage= "false"`	*(default)*

Alguns atributos de @page

- *session*
  - *Se true, aplicações JSP podem manter sessões do usuário abertas usando HttpSession*
  - *Se uma página declara session=false, ela não terá acesso a objetos gravados na sessão do usuário (objeto HttpSession)*
- *isThreadSafe*
  - *Se true, só um cliente poderá acessar a página ao mesmo tempo*
- *isErrorPage*
  - *Se true, a página possui um objeto exception (Throwable) e pode extrair seus dados quando alvo de redirecionamento devido a erro. Possui também os dois atributos padrão de páginas de erro.*
- *errorPage*
  - *URL da página para o qual o controle será redirecionado na ocorrência de um erro ou exceção. Deve ser uma página com isErrorPage=true.*

Atributos de @page: buffer e autoflush

- Pode-se redirecionar, criar um cookie ou modificar o tipo de dados gerado por uma página JSP em qualquer parte dela
  - Essas operações são realizadas pelo browser e devem ser passadas através do cabeçalho de resposta do servidor
  - Lembre-se que o cabeçalho termina ANTES que os dados comecem
- O servidor JSP armazena os dados da resposta do servidor em um buffer (de 8kB, default) antes de enviar
  - Assim é possível montar o cabeçalho corretamente antes dos dados, e permitir que o programador escolha onde e quando definir informações de cabeçalho
  - O buffer pode ser redefinido por página (diretiva page buffer). Aumente-o se sua página for grande.
  - autoFlush determina se dados serão enviados quando buffer encher ou se o programa lançará uma exceção.

(b) declarações

- Dão acesso ao corpo da classe do servlet. Permitem a declaração de variáveis e métodos em uma página
- Úteis para declarar:
  - Variáveis e métodos de instância (pertencentes ao servlet)
  - variáveis e métodos estáticos (pertencentes à classe do servlet)
  - Classes internas (estáticas e de instância), blocos static, etc.
- Sintaxe

```
<%! declaração %>
```

- Exemplos

```
<%! public final static String[] meses =
      {"jan", "fev", "mar", "abr", "mai", "jun"};
%>
<%! public static String getMes() {
       Calendar cal = new GregorianCalendar();
       return meses[cal.get(Calendar.MONTH)];
    }
%>
```

(b) declarações (métodos especiais)

- *jspInit()* e *jspDestroy()* permitem maior controle sobre o ciclo de vida do servlet
  - Ambos são opcionais
  - Úteis para inicializar conexões, obter recursos via JNDI, ler parâmetros de inicialização do web.xml, etc.
- Inicialização da página (chamado *uma* vez, antes da primeira requisição, após o instanciamento do servlet)

```
<%!
   public void jspInit() { ... }
%>
```

- Destruição da página (ocorre quando o servlet deixa a memória)

```
<%! public void jspDestroy() { ... } %>
```

(c) expressões e (d) scriptlets

- ***Expressões***: *Quando processadas, retornam um valor que é inserido na página no lugar da expressão*
- *Sintaxe:*

```
<%= expressão %>
```

- *Equivale a* `out.print(expressão)`, *portanto, não pode terminar em ponto-e-vírgula*
  - *Todos os valores resultantes das expressões são convertidos em String antes de serem redirecionados à saída padrão*
- ***Scriptlets***: *Blocos de código que são executados sempre que uma página JSP é processada*
- *Correspondem a inserção de seqüências de instruções no método* `_jspService()` *do servlet gerado*
- *Sintaxe:*

```
<% instruções Java; %>
```

(e) comentários

- *Comentários HTML* <!-- --> *não servem para comentar JSP*

```
<!-- Texto ignorado pelo browser mas não pelo
servidor. Tags são processados -->
```

- *Comentários JSP: podem ser usados para comentar blocos JSP*

```
<%--    Texto, código Java, <HTML> ou tags
<%JSP%> ignorados pelo servidor --%>
```

- *Pode-se também usar comentários Java quando dentro de scriptlets, expressões ou declarações:*

```
<%  código JSP ... /* texto ou comandos Java
ignorados pelo servidor */ ... mais código %>
```

(f) ações padronizadas

- *Sintaxe:*

```
<jsp:nome_ação atrib1 atrib2 ... >
  <jsp:param name="xxx" value="yyy"/>
   ...
</jsp:nome_ação>
```

- *Permitem realizar operações (e meta-operações) externas ao servlet (tempo de execução)*
  - *Concatenação de várias páginas em uma única resposta*
    `<jsp:forward>` e **`<jsp:include>`**
  - *Inclusão de JavaBeans*
    `<jsp:useBean>`, **`<jsp:setProperty>`** e **`<jsp:getProperty>`**
  - *Geração de código HTML para Applets*
    `<jsp:plugin>`

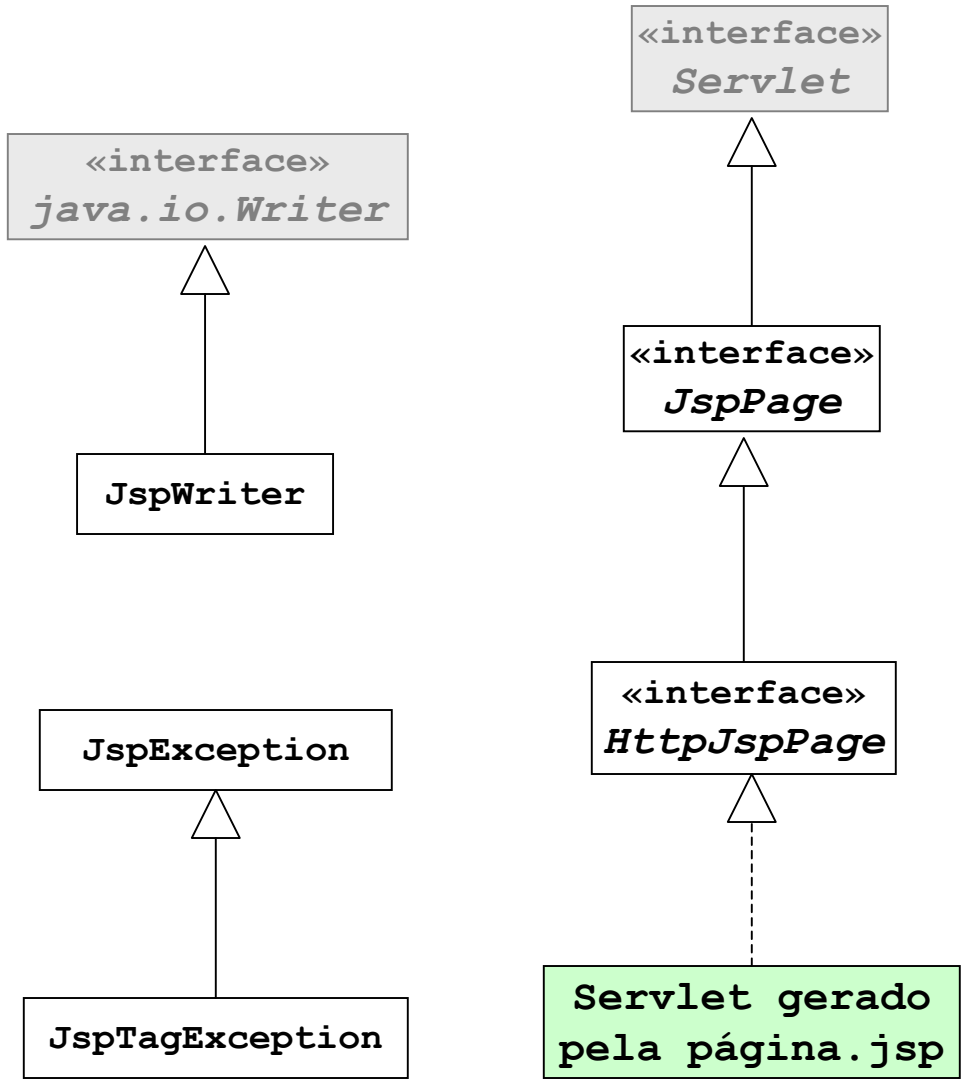
(f) ações (exemplos)

```
<%
if (Integer.parseInt(totalImg) > 0) {
%>
  <jsp:forward page="selecimg.jsp">
     <jsp:param name="totalImg"
                value="<%= totalImg %>"/>
     <jsp:param name="pagExibir" value="1"/>
  </jsp:forward>
<%
} else {
%>
  <p>Nenhuma imagem foi encontrada.
<%
}
%>
```

API: Classes de suporte a JSP

Pacote *javax.servlet.jsp*

- *Interfaces*
  - *JspPage*
  - *HttpJspPage*
- *Classes abstratas:*
  - *JspEngineInfo*
  - *JspFactory*
  - *JspWriter*
  - *PageContext*
- *Classes concretas:*
  - *JspException*
  - *JspTagException*



Objetos implícitos JSP

- São variáveis locais previamente inicializadas
- Disponíveis nos blocos <% . . . %> (scriptlets) de qualquer página (exceto session e exception que dependem de @page para serem ativados/desativados)

- Objetos do servlet
  - page
  - config
- Entrada e saída
  - request
  - response
  - out
- Objetos contextuais
  - session
  - application
  - pageContext
- Controle de exceções
  - exception

(a) page

- Referência para o servlet gerado pela página
  - Equivale a "this" no servlet
- Pode ser usada para chamar qualquer método ou variável do servlet ou superclasses
  - Tem acesso aos métodos da interface *javax.servlet.jsp.JspPage* (ou *HttpJspPage*)
  - Pode ter acesso a mais variáveis e métodos se estender alguma classe usando a diretiva @page extends:

```
<%@ page extends="outra.Classe" %>
```

- Exemplo:

```
<% HttpSession sessionCopy =
                    page.getSession() %>
```

(b) config

- *Referência para os parâmetros de inicialização do servlet (se existirem) através de objeto ServletConfig*
- *Equivale a page.getServletConfig()*
- *Exemplo:*

```
<% String user = config.getInitParameter("nome");
   String pass = config.getInitParameter("pass"); %>
```

- *Parâmetros de inicialização são fornecidos na instalação do servlet no servidor, através de <init-param> de <servlet> em web.xml. É preciso declarar a página no web.xml*

```
<servlet>
    <servlet-name>ServletJSP</servlet-name>
    <jsp-page>/pagina.jsp</jsp-page>
    <init-param>
      <param-name>nome</param-name>
      <param-value>guest</param-value>
    </init-param>
</servlet>
```

(c) request

- Referência para os dados de entrada enviados na requisição do cliente (no GET ou POST, por exemplo, em HTTP)
  - É um objeto do tipo javax.servlet.http.HttpServletRequest
- Usado para
  - Guardar e recuperar atributos que serão usadas enquanto durar a requisição (que pode durar mais de uma página)
  - Recuperar parâmetros passados pelo cliente (dados de um formulário HTML, por exemplo)
  - Recuperar cookies
  - Descobrir o método usado (GET, POST)

```
String method = request.getMethod();
```

(c) exemplos

- *URL no browser:*

```
http://servidor/programa.jsp?nome=Fulano&id
=5
```

- *Recuperação dos parâmetros no programa JSP:*

```
<%
String nome = request.getParameter("nome");
String idStr = request.getParameter("id");
int id = Integer.parseInt(idStr);
%>
<p>Bom dia <%=nome %>! (cod: <%=id %>
```

- *Cookies*

```
Cookie[] c = request.getCookies()
```

(d) response

- Referência aos dados de saída enviados na resposta do servidor enviada ao cliente
  - É um objeto do tipo javax.servlet.http.HttpServletResponse
- Usado para
  - Definir o tipo dos dados retornados (default: text/thml)
  - Criar cookies

```
Cookie c = new Cookie("nome", "valor");
response.addCookie(c);
```

  - Definir cabeçalhos de resposta
  - Redirecionar

```
response.sendRedirect("pagina2.html");
```

(e) out

- *Representa o stream de saída da página (texto que compõe o HTML que chegará ao cliente).*
  - *É instância da classe javax.servlet.jsp.JspWriter (implementação de java.io.Writer)*
- *Equivalente a* `response.getWriter();`
- *Principais métodos*

  `print()` e `println()` - *imprimem Unicode*
- *Os trechos de código abaixo são equivalentes*

```
<% for (int i = 0; i < 10; i++) {
out.print("<p> Linha " + i);
} %>

<% for (int i = 0; i < 10; i++) { %>
<p> Linha <%= i %>
<% } %>
```

(f) session

- Representa a sessão do usuário
  - O objeto é uma instância da classe *javax.servlet.http.HttpSession*
- Útil para armazenar valores que deverão permanecer durante a sessão (set/getAttribute())

```
Date d = new Date();
session.setAttribute("hoje", d);
  ...
Date d = (Date)
          session.getAttribute("hoje");
```

(g) application

- Representa o contexto ao qual a página pertence
  - Instância de javax.servlet.ServletContext
- Útil para guardar valores que devem persistir pelo tempo que durar a aplicação (até que o servlet seja descarregado do servidor)
- Exemplo

```
Date d = new Date();
application.setAttribute("hoje", d);
  ...
Date d = (Date)
  application.getAttribute("hoje");
```

(h) pageContext

- Instância de javax.servlet.jsp.PageContext
- Oferece acesso a todos os outros objetos implícitos.
  Métodos:
  - getPage() - retorna page
  - getRequest() - retorna request
  - getResponse() - retorna response
  - getOut() - retorna out
  - getSession() - retorna session
  - getServletConfig() - retorna config
  - getServletContext() - retorna application
  - getException() - retorna exception
- Constrói a página (mesma resposta) com informações localizadas em outras URLs
  - pageContext.forward(String) - mesmo que ação <jsp:forward>
  - pageContext.include(String) - mesmo que ação <jsp:include>

Escopo dos objetos

- A persistência das informações depende do escopo dos objetos onde elas estão disponíveis
- Constantes da classe javax.servlet.jsp.PageContext identificam escopo de objetos
  - pageContext PageContext.PAGE_SCOPE
  - request PageContext.REQUEST_SCOPE
  - session PageContext.SESSION_SCOPE
  - application PageContext.APPLICATION_SCOPE

- Métodos de pageContext permitem setar ou buscar atributos em qualquer objeto de escopo:
  - `setAttribute(nome, valor, escopo)`
  - `getAttribute(nome, escopo)`

(i) exception

- *Não existe em todas as páginas - apenas em páginas designadas como páginas de erro*

```
<%@ page isErrorPage="true" %>
```

- *Instância de java.lang.Throwable*
- *Exemplo:*

```
<h1>Ocoreu um erro!</h1>
<p>A exceção é
<%= exception %>
Detalhes: <hr>
<% exception.printStackTrace(out); %>
```

Depuração de JSP

- Apesar de ser muito mais fácil escrever JSP, a depuração não é simples
  - A página é compilada no servidor, exigindo que se procure por erros de compilação ou de parsing em uma página HTML remota
  - Nem sempre os erros são claros. Erros de parsing (um tag %> faltando, produzem mensagens esdrúxulas)
  - Os números das linhas, nas mensagens do Tomcat, não correspondem ao local do erro no JSP mas ao número da linha do código Java do servlet que foi gerado: você encontra o servlet no diretório work, do Tomcat
  - O servlet gerado pelo Tomcat não é fácil de entender, usa variáveis pouco explicativas e código repetido.

Servlet gerado de hello.jsp (do Cap 1)

- *Procure-o em $TOMCAT_HOME\work\Standalone\localhost_\hello$jsp.java*

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class hello$jsp extends HttpJspBase {
    static {    }
    public hello$jsp( ) {    }
    private static boolean _jspx_inited = false;
    public final void _jspx_init() throws org.apache.jasper.runtime.JspException {
    }
    public void _jspService(HttpServletRequest request, HttpServletResponse  response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        String  _value = null;
        try {
            if (_jspx_inited == false) {
                synchronized (this) {
                    if (_jspx_inited == false) {
                        _jspx_init();
                        _jspx_inited = true;
                    }
                }
            }
```

Continuação de hello.jsp

$TOMCAT_HOME\work\Standalone\localhost_\hello$jsp.java

```
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;ISO-8859-1");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                                                      "", true, 8192, true);
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            // HTML // begin [file="/hello.jsp";from=(0,0);to=(8,0)]
                out.write("<HTML><HEAD>\r\n<TITLE>Simple Servlet Output</TITLE>\r\n");
                out.write("</HEAD>\r\n\r\n<BODY>\r\n");

            // end
            // begin [file="/hello.jsp";from=(8,2);to=(12,0)]
                    String user = request.getParameter("usuario");
                    if (user == null)
                        user = "World";
            // end
            // HTML // begin [file="/hello.jsp";from=(12,2);to=(14,10)]
                out.write("\r\n<H1>Simple JSP Output</H1>\r\n<P>Hello, ");
            // end
            // begin [file="/hello.jsp";from=(14,13);to=(14,19)]
                out.print( user );
            // end
            // HTML // begin [file="/hello.jsp";from=(14,21);to=(17,0)]
                out.write("\r\n</BODY></HTML>\r\n\r\n");
            // end
        } catch (Throwable t) {
            if (out != null && out.getBufferSize() != 0)
                out.clearBuffer();
            if (pageContext != null) pageContext.handlePageException(t);
        } finally {
            if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
        }
    }
}
```

Números de linha da página JSP

Código e HTML da página

Não precisa montar projeto. Simplesmente copie os arquivos para o contexto ROOT!

Exercícios

- *1. Escreva um JSP data.jsp que imprima a data de hoje.*
  - *Use Calendar e GregorianCalendar*
- *2. Escreva um JSP temperatura.jsp que imprima uma tabela HTML de conversão Celsius-Fahrenheit entre -40 e 100 graus Celsius com incrementos de 10 em 10*
  - *A fórmula é F = 9/5 C + 32*
- *3. Altere o exercício anterior para que a página também apresente um campo de textos para entrada de temperatura em um formulário que envie os dados com POST. Faça com que a própria página JSP receba a mensagem*
  - *a) Identifique, no início, o método com request.getMethod() (retorna POST ou GET, em maiúsculas).*
  - *b) Se o método for POST, mostre, em vermelho, antes da exibição do formulário, o texto: "x graus F = y graus C" onde x é o valor digitado pelo usuário e y é a resposta.*

Faça estes arquivos parte de um contexto
(por exemplo: forumjsp)

Exercícios

- *4. JSP simples usando objeto de sessão*
  - *a. Escreva uma página JSP novaMensagem.jsp que mostre um formulário na tela com dois campos: email e mensagem.*
  - *b. Escreva uma outra página gravarMensagem.jsp que receba dois parâmetros: email e mensagem e grave esses dois parâmetros na sessão do usuário.*
  - *c. Faça com que a primeira página aponte para a segunda.*
  - *d. Crie uma terceira página listarMensagens.jsp que mostre todas as mensagens criadas até o momento.*
- *5. Altere o exercício anterior fazendo com que*
  - *a. A página gravarMensagem.jsp mostre todas as mensagens da sessão como resposta, mas grave a mensagem em disco usando parâmetro de inicialização do web.xml*
  - *b. A página listarMensagens.jsp liste todas as mensagens em disco.*
  - *Obs: garanta uma gravação thread-safe para os dados.*

helder@acm.org

argonavis.com.br

JSP com tags padrão

Helder da Rocha
www.argonavis.com.br

Objetivos

- *Utilizar os marcadores padrão do JSP com o objetivo principal de reduzir a quantidade de código Java nas páginas e promover a separação de responsabilidades*
  - *Marcadores para uso de JavaBeans (View Helper pattern): <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty>*
  - *Marcadores para divisão de páginas em blocos menores que são compostos em tempo de execução ou de compilação (Composite View pattern): <%@include%> e <jsp:include>*
  - *Marcadores para redirecionamento de requisição para outras páginas ou servlets <jsp:forward>*
  - *Marcadores para geração de código HTML de suporte a applets: <jsp:plugin>*

Usando beans

- JavaBeans são objetos escritos de acordo com um determinado padrão que permite tratá-los como componentes de um framework
  - Ótimos para separar os detalhes de implementação de uma aplicação de seus "serviços"
  - Permitem encapsular dados recebidos de outras partes da aplicação e torná-los disponíveis para alteração e leitura através de uma interface uniforme.
- Podem ser usados com JSP para remover grande parte do código Java de uma página JSP
  - Maior facilidade de manutenção e depuração
  - Separação de responsabilidade e reuso de componentes

Como incluir um bean

- *Para que um bean possa ser usado por uma aplicação JSP, ele deve estar compilado e localizado dentro do CLASSPATH reconhecido pelo servidor*
  - *No subdiretório WEB-INF/classes do seu contexto*
- *Para incluir:*

```
<jsp:useBean id="nome_da_referência"
             class="pacote.NomeDaClasse"
             scope="page|session|request|application">
</jsp:useBean>
```

- *O atributo de escopo é opcional e indica o tempo de vida do Java Bean. Se omitido, será page, que o limita à página*
  - *Com escopo de request, o bean pode ser recuperado com outra instrução <jsp:useBean> que esteja em outra página que receber a mesma requisição (via dispatcher)*
  - *Com escopo de session, o bean é recuperável em páginas usadas pelo mesmo cliente, desde que <%@page> não tenha session=false*

Como incluir um bean

- *O nome do bean (atributo id) comporta-se como uma referência a um objeto Java*
- *Incluir o tag*

```
<jsp:useBean id="bean" class="bean.HelloBean"
             scope="request" />
```

é o mesmo que incluir na página

```
<% Object obj = request.getAttribute("bean");
   bean.HelloBean bean = null;
   if (obj == null) {
      bean = new bean.HelloBean();
      request.setAttribute("bean", bean);
   } else {
      bean = (bean.HelloBean) obj;
   } %>
```

- *O id pode ser usado em scriptlets para usar membros do bean*

```
<%  bean.setValor(12); %>
```

Propriedades

- *JavaBeans possuem propriedades que podem ser somente-leitura ou leitura-alteração.*
- *O nome da propriedade é sempre derivada do nome do método getXXX():*

```
public class Bean {
  private String mensagem;
  public void setTexto(String x) {
    mensagem = x;
  }
  public String getTexto() {
    return mensagem;
  }
}
```

- *O bean acima tem uma propriedade (RW) chamada* `texto`

Propriedades

- *Páginas JSP podem ler ou alterar propriedades de um bean usando os tags*

```
<jsp:setProperty name="bean" property="propriedade"
                 value="valor"/>
```

que equivale a `<% bean.setPropriedade(valor); %>` e

```
<jsp:getProperty name="bean" property="propriedade"/>
```

que equivale a `<%=bean.getPropriedade() %>`

- *Observe que o nome do bean é passado através do atributo name, que corresponde ao atributo id em* `<jsp:useBean>`
- *Valores são convertidos de e para String automaticamente*
- *Parâmetros HTTP com mesmo nome que as propriedades têm valores passados automaticamente com <jsp:setProperty>*
  - *Se não tiverem, pode-se usar atributo param de <jsp:setProperty>*
  - *<jsp:setProperty ... property="*"/> lê todos os parâmetros*

Inicialização de beans

- *A tag <jsp:useBean> simplesmente cria um bean chamando seu construtor. Para inicializá-lo, é preciso chamar seus métodos setXXX() ou usar <jsp:setProperty> após a definição*

```
<jsp:useBean id="bean" class="bean.HelloBean" />
<jsp:setProperty name="bean" property="prop" value="valor"/>
```

- *Se um bean já existe, porém, geralmente não se deseja inicializá-lo.*
- *Neste caso, a inicialização pode ser feita dentro do marcador <jsp:useBean> e o sistema só a executará se o bean for novo (se já existir, o código será ignorado)*

```
<jsp:useBean id="bean" class="bean.HelloBean" />
  <jsp:setProperty name="bean" property="prop" value="valor"/>
</jsp:useBean>
```

ou

```
<jsp:useBean id="bean" class="bean.HelloBean" />
  <% bean.setProp(valor); %>
</jsp:useBean>
```

Condicionais e iterações

- *Não é possível usar beans para remover de páginas Web o código Java de expressões condicionais e iterações como for do-while e while*
  - *Para isto, não há tags padrão. É preciso usar Taglibs*
- *Beans, porém, podem ser usados dentro de iterações e condicionais, e ter seus valores alterados a cada repetição ou condição*

```
<jsp:useBean id="mBean" class="MessageBean" scope="session" />
<% MessageBean[] messages = MessagesCollection.getAll();
   for (int i = messages.length -1; i >= 0; i--) {
      mBean = messages[i];
%>
   <tr><td><jsp:getProperty name="mBean" property="time"/></td>
   <td><%=mBean.getHost()%></td>
   <td><%=mBean.getMessage()%></td></tr>
<% } %>
```

Pode-se usar uma forma ou a outra

(MessageBean tem propriedades: time, host, message)

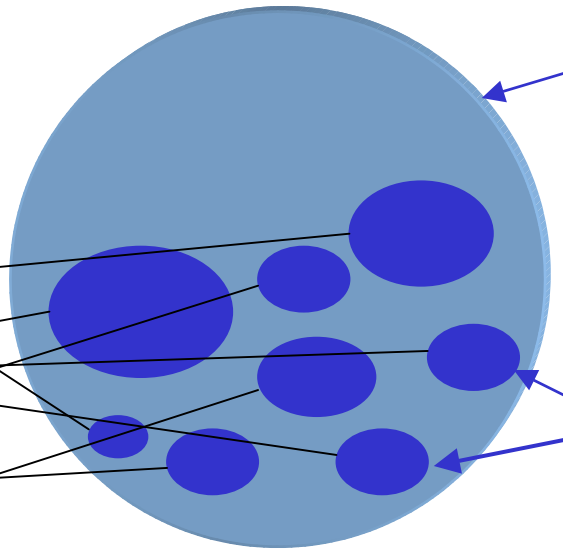
Matando beans

- *Beans são sempre gravados em algum objeto de escopo: page, request, session ou application*
  - *Persistem até que o escopo termine ou expirem devido a um timeout (no caso de sessões)*
- *Para se livrar de beans persistentes, use os métodos removeAttribute(), disponíveis para cada objeto de escopo:*

```
session.removeAttribute(bean);
application.removeAttribute(bean);
request.removeAttribute(bean);
```

Composite View

- *Páginas Web complexas (ex: portais) freqüentemente são divididas em partes independentes*
  - *Algumas partes são altamente dinâmicas, mudando frequentemente até na estrutura interna*
  - *Outras partes mudam apenas o conteúdo*
  - *Outras partes sequer mudam o conteúdo*
- *Gerar uma página dessas usando apenas um template é indesejável*

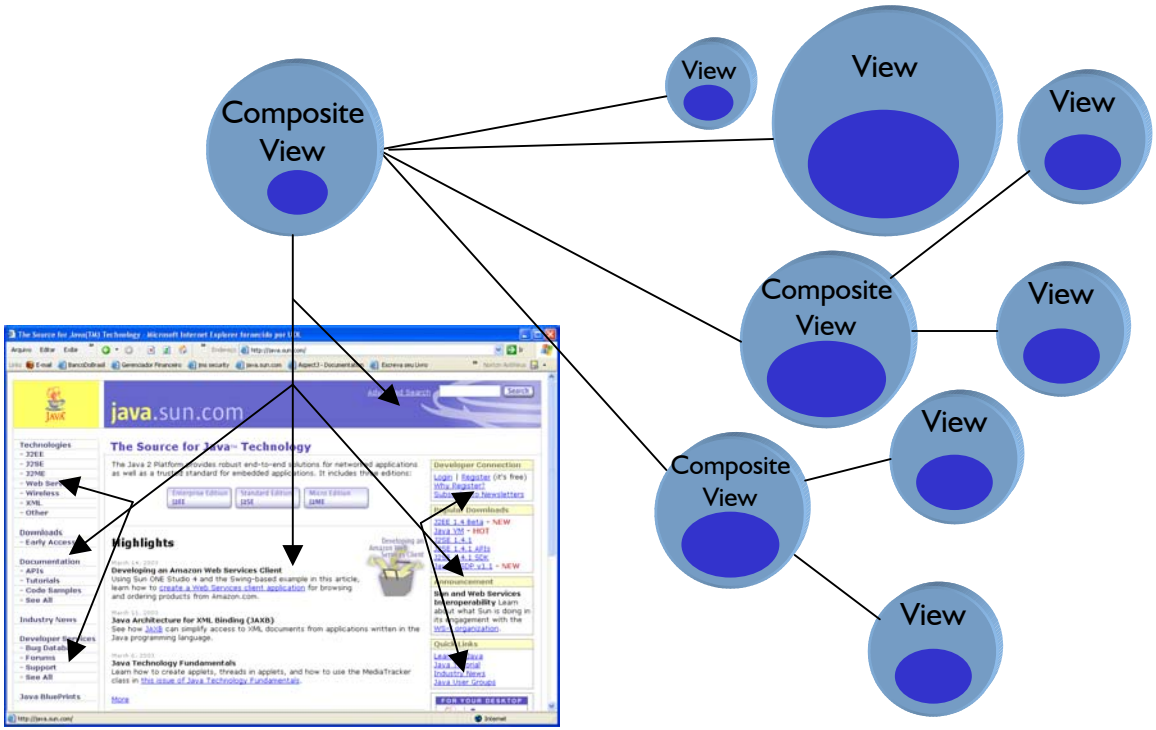


Template JSP

Código espalhado usado para gerar partes de cada seção da página

Composite View (2)

- *O padrão de projeto Composite View sugere que tais páginas sejam separadas em blocos menores, que possam ser alterados individualmente e compostos durante a publicação (deployment) ou exibição*



- *JSP oferece duas soluções para obter esse efeito*
  - *Usando inclusão estática (no momento da compilação do servlet)*
  - *Usando inclusão dinâmica (no momento da requisição)*

Inclusão estática

- *Mais eficiente: fragmentos são incluídos em único servlet*
- *Indicada quando estrutura não muda com freqüência (conteúdo pode mudar)*
  - *Menus, Logotipos e Avisos de copyright*
  - *Telas com miniformulários de busca*
- *Implementada com <%@ include file="fragmento" %>*

```
<!-- Menu superior -->
<table>
<tr><td><%@ include file="menu.jsp" %></td></tr>
</table>
<!-- Fim do menu superior -->
```

Fragmento menu.jsp

```
<a href="link1">Item 1</a></td>
<td><a href="link2">Item 2</a></td>
<a href="link3">Item 3</a>
```

Se tela incluida contiver novos fragmentos, eles serão processados recursivamente

Inclusão dinâmica

- *Mais lento: fragmentos não são incluídos no servlet mas carregados no momento da requisição*
- *Indicada para blocos cuja estrutura muda com freqüência*
  - *Bloco central ou notícias de um portal*
- *Implementada com <jsp:include page="fragmento"/>*
- *Pode-se passar parâmetros em tempo de execução usando <jsp:param> no seu interior*

```
<!-- Texto principal -->
<table>
<tr><td>
<jsp:include page="texto.jsp">
   <jsp:param name="data" value="<%=new Date() %>">
</jsp:include>
</td></tr> </table>
<!-- Fim do texto principal -->
```

Repasse de requisições

- *Uma requisição pode ser repassada de uma página JSP para outra página ou servlet usando RequestDispatcher*

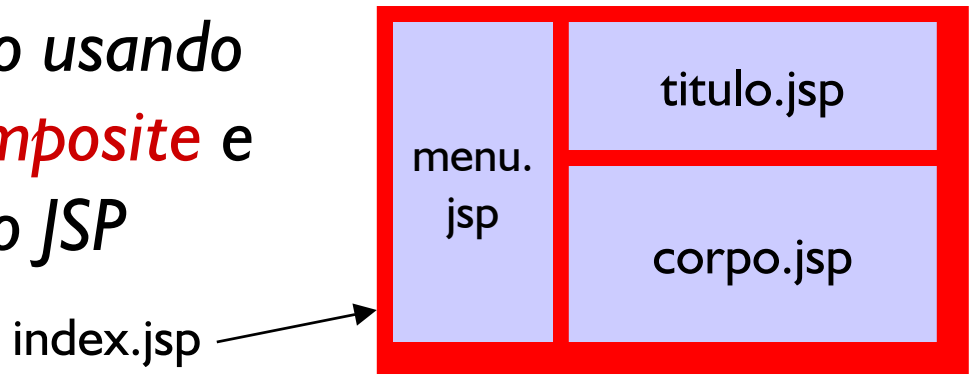
```
<% RequestDispatcher rd =
            request.getRequestDispatcher("url");
   rd.dispatch(request, response);
%>
```

- *O mesmo efeito é possível sem usar scriptlets com a ação padrão <jsp:forward>*
- *Assim como <jsp:include>, pode incluir parâmetros recuperáveis na página que receber a requisição usando request.getParameter() ou <jsp:getProperty> se houver bean*

```
<% if (nome != null) { %>
<jsp:forward page="segunda.jsp">
   <jsp:param name="nome" value="<%=nome %>">
</jsp:forward>
<% } %>
```

Exercícios

- *1. Use um JavaBean Mensagem, com propriedades email e mensagem para implementar o exercício 4 do capítulo anterior*
  - *Substitua todas as chamadas de new Mensagem() por <jsp:useBean> no escopo da sessão*
  - *Use <jsp:getProperty> para exibir os dados*
- *2. Altere gravarMensagens para que use <jsp:forward> para despachar a requisição para uma página erro.jsp, caso o usuário deixe os campos do formulário em branco, e para listarMensagens.jsp se tudo funcionar corretamente*
- *3. Monte a página ao lado usando os arquivos em cap10/composite e os marcadores de inclusão JSP*



helder@acm.org

argonavis.com.br

11 Tag Libraries e JSTL

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Custom tags

- *JSP com JavaBeans fornecem um meio de diminuir código Java da página, mas não totalmente*
  - *Designers de página ainda têm que usar elementos de script para loops e lógica condicional (getProperty e setProperty não bastam)*
  - *Nem sempre os JavaBeans são suficientes para encapsular toda a lógica da aplicação*
- *A especificação prevê a criação de elementos XML personalizados (custom tags) para resolver essas limitações*
  - *Organizados em bibliotecas (taglibs)*
  - *Cada biblioteca tem seu próprio namespace*
- *Taglibs são declaradas no início de cada página ...*

```
<%@taglib uri="http://abc.com/ex" prefix="exemplo"%>
```

namespace

- *... e usadas em qualquer lugar*

```
<exemplo:dataHoje />
```

produz → **Tuesday, May 5, 2002 13:13:13 GMT-03**

Como usar custom tags

- A URI usada para identificar o prefixo de um custom tag não precisa ser real (e apontar para um local)
  - Serve apenas como **identificador**
  - Ligação entre a especificação da biblioteca (arquivo TLD) e o identificador é feito no arquivo web.xml

```
<web-app>
    ...
    <taglib>
        <taglib-uri>http://abc.com/ex</taglib-uri>
        <taglib-location>
        /WEB-INF/mytaglib.tld
        </taglib-location>
    </taglib>
</web-app>
```

Este é o deployment descriptor do Taglib.

Localização real!

Exemplo de arquivo TLD

```
<?xml version="1.0" ?>
<!DOCTYPE taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>exemplo</short-name>
    <uri>http://abc.com/ex</uri>

    <tag>
        <name>dataHoje</name>
        <tag-class>exemplos.DateTag</tag-class>
        <description>Data de hoje</description>
    </tag>

</taglib>
```

Sugestão de prefixo (autor de página pode escolher outro na hora)

URI identifica o prefixo. (autor de página tem que usar exatamente esta URI)

Exemplos de Custom Tags

- *Veja* **exemplos/cap11/taglibs/**
  - *Vários diferentes exemplos de custom tags (de [6])*
  - *Código fonte em taglib/src/taglibdemo/*.java*
  - *Páginas exemplo em src/*Test.jsp (6 exemplos)*

1. Configure build.properties, depois, monte o WAR com:

```
> ant build
```

2. Copie o WAR para o diretório webapps do Tomcat

```
> ant deploy
```

3. Execute os tags, acessando as páginas via browser:

```
http://localhost:porta/mut/
```

- *Veja também* **exemplos/cap12/mvc/hellojsp_2/**
  - *Aplicação MVC que usa custom tags*

JSP Standard Tag Library

- Esforço de padronização do JCP: JSR-152
  - Baseado no Jakarta Taglibs (porém bem menor)
- Oferece dois recursos
  - Conjunto padrão de tags básicos (Core, XML, banco de dados e internacionalização)
  - Linguagem de expressões do JSP 1.3
- Oferece mais controle ao autor de páginas sem necessariamente aumentar a complexidade
  - Controle sobre dados sem precisar escrever scripts
  - Estimula a separação da apresentação e lógica
  - Estimula o investimento em soluções MVC

Como usar JSTL

- *1. Fazer o download da última versão do site da Sun*
- *2. Copiar os JARs das bibliotecas desejadas para o diretório WEB-INF/lib/ da sua aplicação Web e os arquivos TLD para o diretório WEB-INF/*
- *3. Declarar cada taglib e associá-la com seu TLD no deployment descriptor web.xml.*
- *4. Incluir em cada página que usa os tags:*

```
<%@ taglib uri="uri_da_taglib"
           prefix="prefixo" %>
```

- *5. Usar os tags da biblioteca com o prefixo definido no passo anterior*

```
<prefixo:nomeTag atributo="..."> ...
</prefixo:nomeTag>
```

Quatro bibliotecas de tags

- *Core library: tags para condicionais, iterações, urls, ...*

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

  - *Exemplo:* `<c:if test="..." ... >...</c:if>`
- *XML library: tags para processamento XML*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/xml"
           prefix="x" />
```

  - *Exemplo:* `<x:parse>...</x:parse>`
- *Internationalization library*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/fmt"
           prefix="fmt" />
```

  - *Exemplo:* `<fmt:message key="..." />`
- *SQL library*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/sql"
           prefix="sql" />
```

  - *Exemplo:* `<sql:update>...</sql:update>`

Linguagem de expressões

- *Permite embutir em atributos expressões dentro de delimitadores ${...}*
  - *Em vez de* `request.getAttribute("nome")`
    `${nome}`
  - *Em vez de* `bean.getPessoa().getNome()`
    `${bean.pessoa.nome}`
- *Suporta operadores aritméticos, relacionais e binários*
- *Converte tipos automaticamente*

```
<tag item="${request.valorNumerico}" />
```

- *Valores default*

```
<tag value="${abc.def}" default="todos" />
```

Principais ações

- *Suporte à impressão da linguagem expressões*
  - `<c:out value="${pessoa.nome}" />`
- *Expressões condicionais*
  - `<c:if test="${pessoa.idade >= 18}">`
    `<a href="adultos.html">Entrar</a>`
    `</c:if>`
  - `<c:choose>`
    `<c:when test="${dia.hora == 13}">`
    `<c:out value="${mensagemEspecial}" />`
    `</c:when>`
    `<c:otherwise>`
    `<c:out value="${mensagemPadrao}" />`
    `</c:otherwise>`
  - `</c:choose>`
- *Iteração*
  - `<c:forEach items="${pessoas}" var="p" varStatus="s">`
    `<c:out value="${s.count}"/>. <c:out value="${p}"/>`
    `</c:forEach>`

Internacionalização, XML e SQL

- Ler propriedade de ResourceBundle
  - `<fmt:message key="chave.do.bundle" />`
- Operações diretas em banco de dados
  - `<sql:query dataSource="${dsn}">`
    `SELECT...</sql:query>`
  - `<sql:transaction>`, `<sql:update>`, etc.
- Operações com XML
  - Uso de expressões XPath em tags JSTL para XML
  - Ações XML: `<x:out>`, `<x:set>`, `<x:if>`, `<x:choose>`, `<x:forEach>` (atributo select contém expr. XPath)
  - `<x:parse>` Processa XML usando DOM ou filtro SAX
  - `<x:transform>` Realiza transformação XSLT.

Exercícios

- 1. Inclua a taglib exemplo.tld em uma de suas aplicações e use o tag dataHoje
  - O tag está dentro de exemplo.jar que deve ser copiado para o diretório lib de sua aplicação
- 2. Instalar tags do JSTL
  - Instale a biblioteca JSTL na sua aplicação, copiando os JARs e TLDs para os locais exigidos
  - Veja a documentação e os tags disponíveis
- 3. Use os tags de lógica <if> e <forEach> para remover as expressões condicionais e iterações das páginas da aplicação de mensagens

helder@acm.org

argonavis.com.br

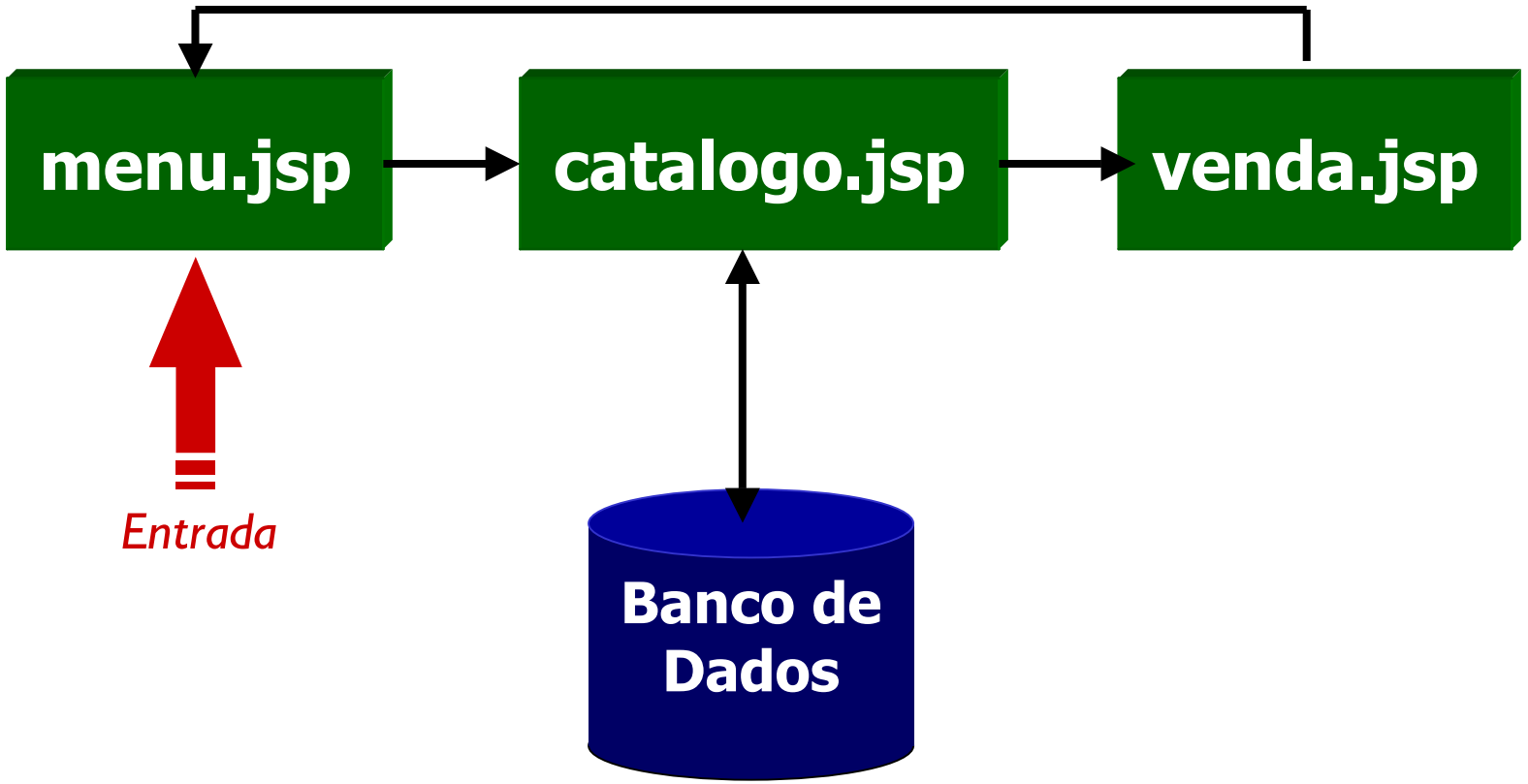
12 Model View Controller

Design de aplicações JSP

- *Design centrado em páginas*
  - *Aplicação JSP consiste de seqüência de páginas (com ou sem beans de dados) que contém código ou links para chamar outras páginas*
- *Design centrado em servlet (FrontController* ou MVC)*
  - *Aplicação JSP consiste de páginas, beans e servlets que controlam todo o fluxo de informações e navegação*
  - *Este modelo favorece uma melhor organização em camadas da aplicação, facilitando a manutenção e promovendo o reuso de componentes.*
  - *Um único servlet pode servir de fachada*
  - *Permite ampla utilização de J2EE design patterns*

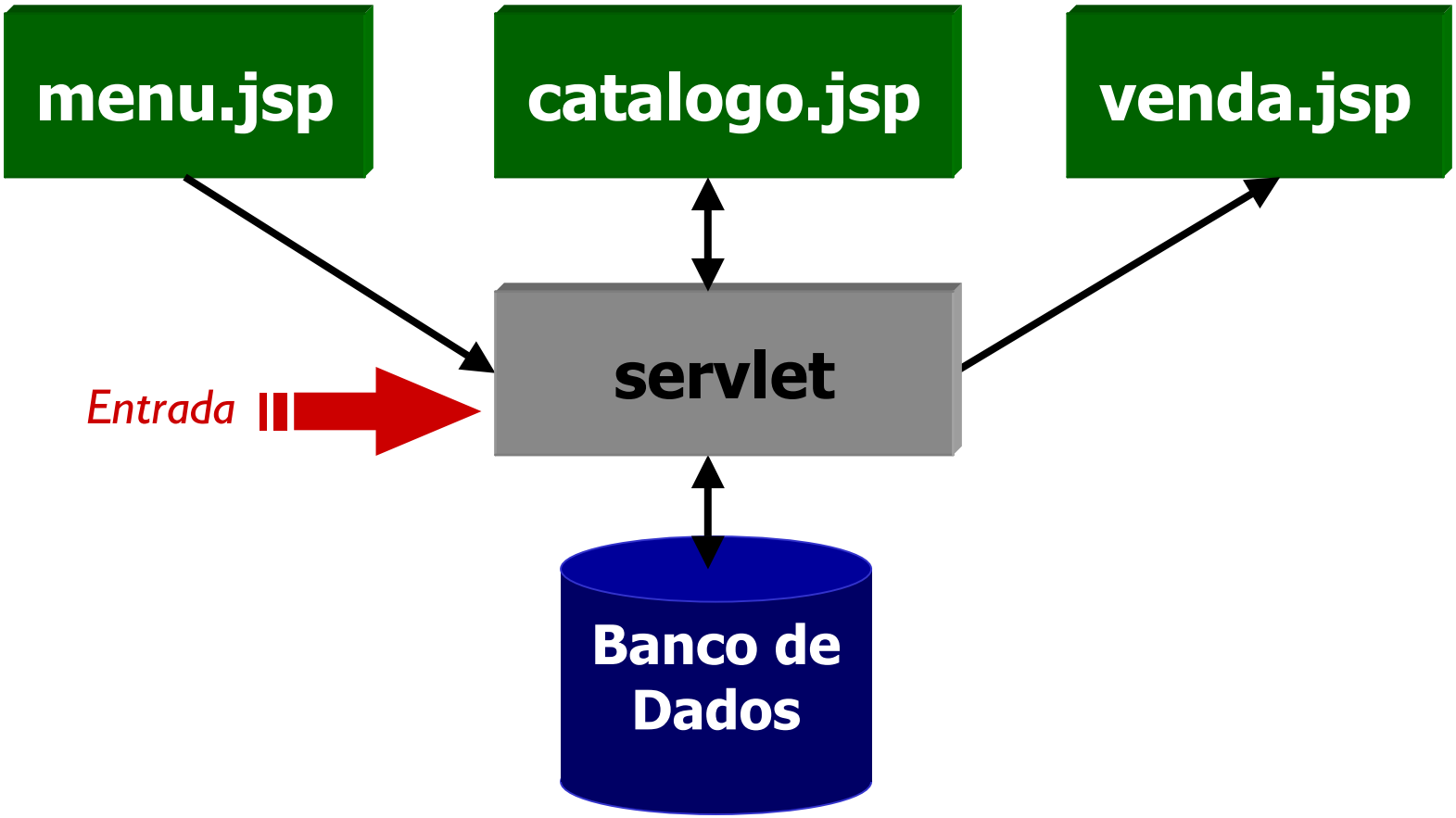
** FrontController é um J2EE design pattern. Vários outros design patterns serão identificados durante esta seção. Para mais informações, veja Sun Blueprints [7]*

Layout centrado em páginas (JSP Model 1)



Fonte: [6]

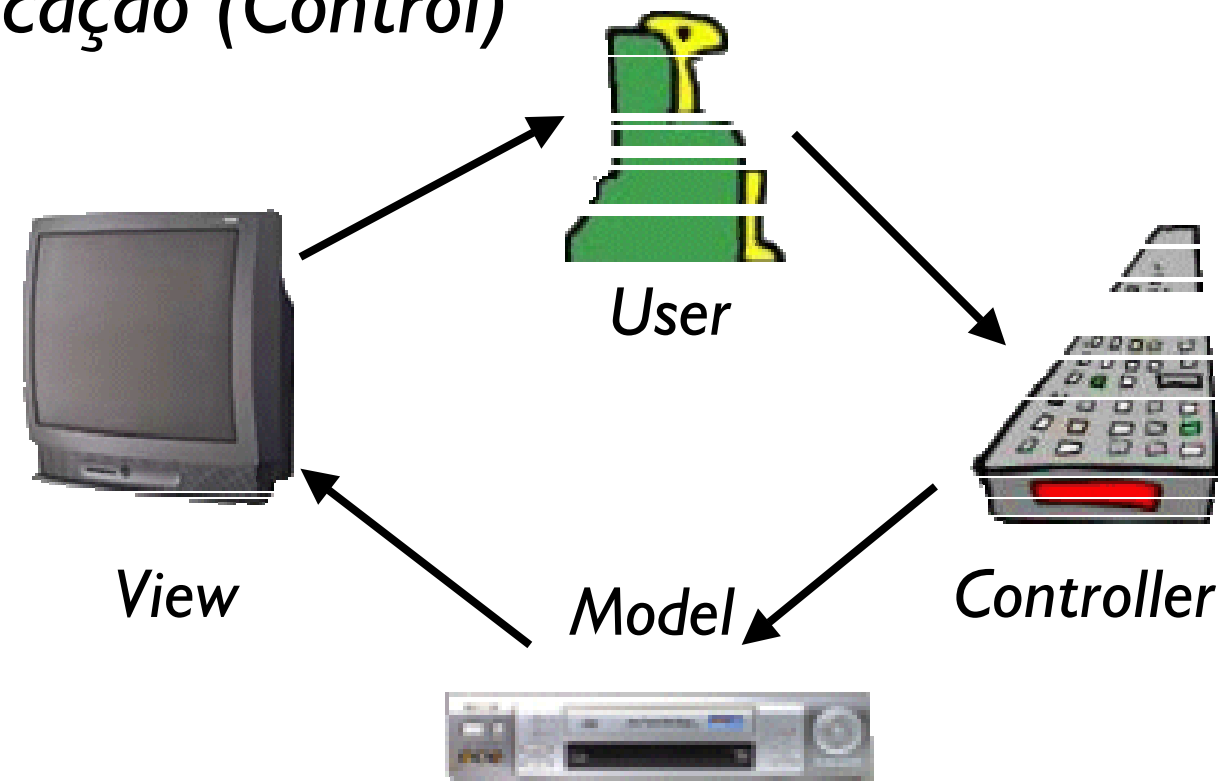
Layout centrado em servlet (JSP Model 2)



Fonte: [6]

O que é MVC

- *Padrão de arquitetura: Model View Controller*
- *Técnica para separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)*



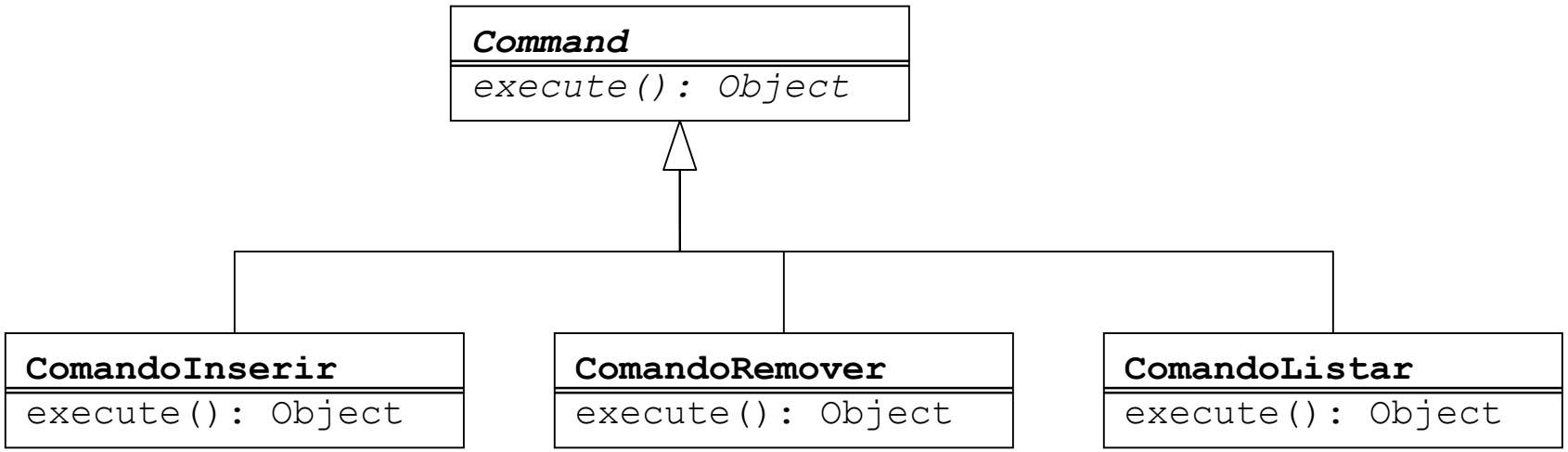
Fonte: *http://www.computer-programmer.org/articles/struts/*

Como implementar?

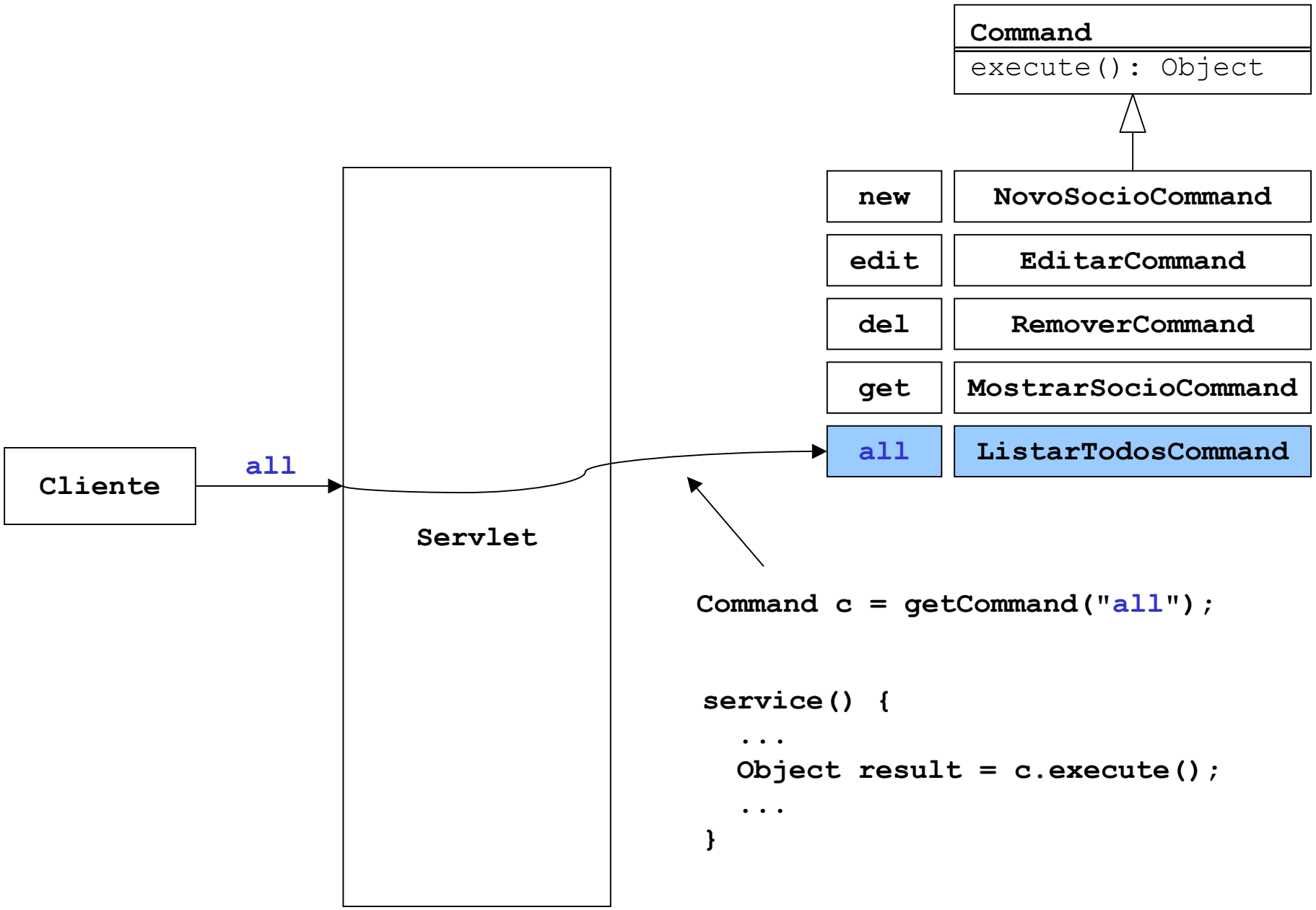
- *Há várias estratégias*
- *Todas procuram isolar*
  - *As operações de controle de requisições em servlets e classes ajudantes,*
  - *Operações de geração de páginas em JSP e JavaBeans, e*
  - *Lógica das aplicações em classes que não usam os pacotes javax.servlet*
- *Uma estratégia consiste em se ter um único controlador (FrontController pattern) que delega requisições a diferentes objetos que implementam comandos que o sistema executa (Command pattern)*

Command Pattern

- *É um padrão de projeto clássico catalogado no livro "Design Patterns" de Gamma et al (GoF = Gang of Four)*
  - *Para que serve: "Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e suportar operações reversíveis." [GoF]*
- *Consiste em usar polimorfismo para construir objetos que encapsulam um comando e oferecer um único método execute() com a implementação do comando a ser executado*



Command Pattern



Command em Java

```
public interface Command {
   public Object execute(Object arg);
}
```

```
public class Server {
  private Database db = ...;
  private HashMap cmds = new HashMap();

  public Server() {
    initCommands();
  }

  private void initCommands() {
    cmds.put("new", new NewCommand(db));
    cmds.put("del",
                 new DeleteCommand(db));
    ...
  }

  public void service(String cmd,
                      Object data) {
    ...
    Command c = (Command)cmds.get(cmd);
    ...
    Object result = c.execute(data);
    ...
  }
}
```

```
public interface NewCommand implements Command {

  public NewCommand(Database db) {
    this.db = db;
  }

  public Object execute(Object arg) {
    Data d = (Data)arg;
    int id = d.getArg(0);
    String nome = d.getArg(1);
    db.insert(new Member(id, nome));
  }
}
```

```
public class DeleteCommand implements Command {

  public DeleteCommand(Database db) {
    this.db = db;
  }

  public Object execute(Object arg) {
    Data d = (Data)arg;
    int id = d.getArg(0);
    db.delete(id);
  }
}
```

FrontController com Command Pattern

- *Os comandos são instanciados e guardados em uma base de dados na memória (HashMap, por exemplo)*
  - *Pode-se criar uma classe específica para ser fábrica de comandos*
- *O cliente que usa o comando (o servlet), recebe na requisição o nome do comando, consulta-o no HashMap, obtém a instância do objeto e chama seu método execute()*
  - *O cliente desconhece a classe concreta do comando. Sabe apenas a sua interface (que usa para fazer o cast ao obtê-lo do HashMap*
- *No HashMap*

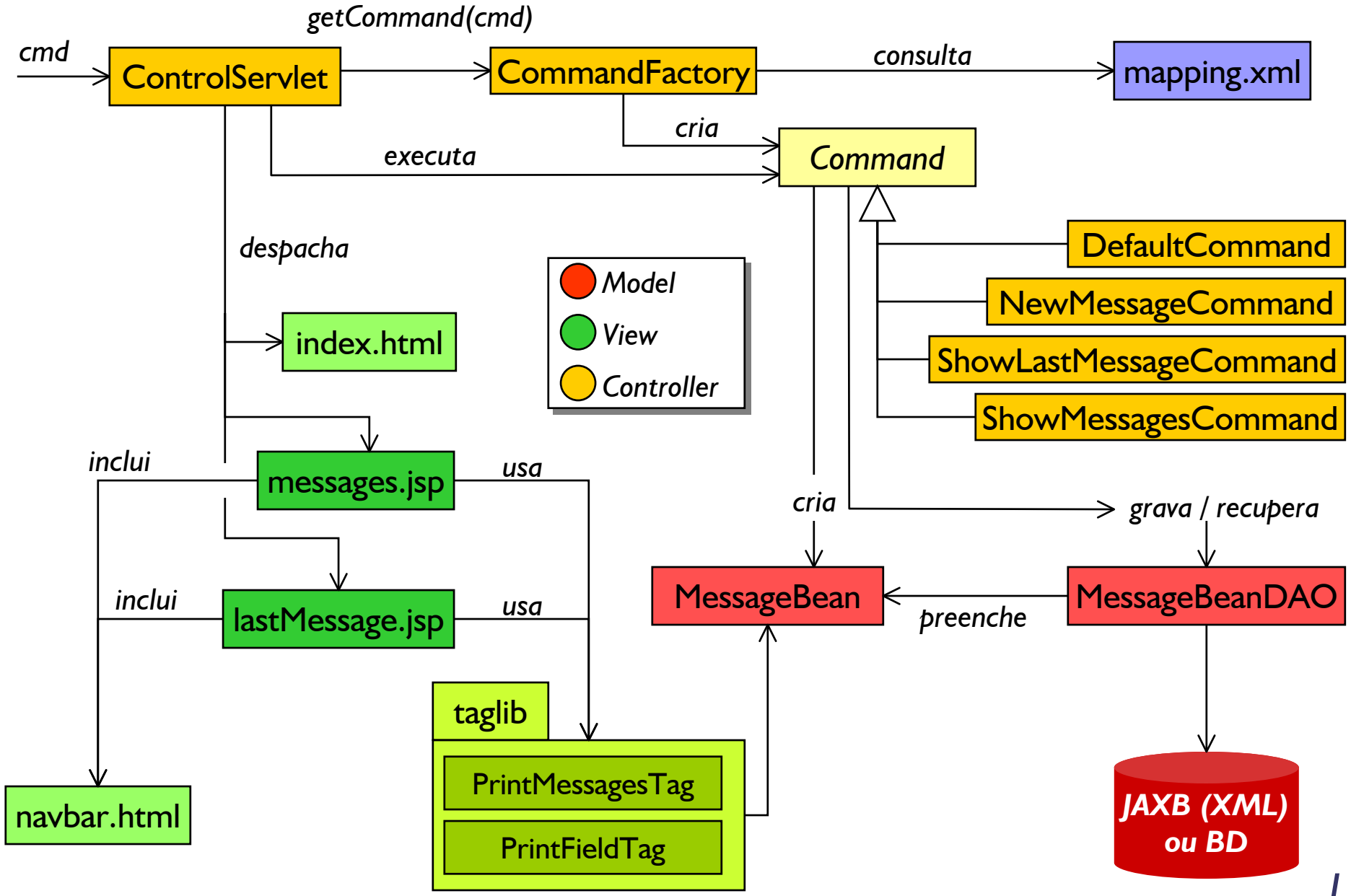
```
Comando c = new ComandoInserir();
comandosMap.put("inserir", c);
```

- *No servlet:*

```
String cmd = request.getParameter("cmd");
Comando c = (Comando)comandosMap.get(cmd);
c.execute();
```

Exemplo de implementação

cap12/mvc/hellojsp_2



Mapeamentos de comandos ou ações

- *No exemplo hellojsp_2, o mapeamento está armazendo em um arquivo XML (webinf/mapping.xml)*

```
<command-mapping> (...)
    <command>
        <name>default</name>
        <class>hello.jsp.DefaultCommand</class>
        <success-url>/index.html</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>newMessage</name>
        <class>hello.jsp.NewMessageCommand</class>
        <success-url>/lastMessage.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>showAllMessages</name>
        <class>hello.jsp.ShowMessagesCommand</class>
        <success-url>/messages.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
</command-mapping>
```

Comandos ou ações (Service to Worker)

- *Comandos implementam a interface Command e seu método*

```
Object execute(HttpServletRequest request,
               HttpServletResponse response,
               MessageBeanDAO dao);
```

- *Criados por CommandFactory na inicialização e executados por ControlServlet que os obtém via getCommand(nome)*
- *Retornam página de sucesso ou falha (veja mapping.xml)*
- *Exemplo: ShowMessagesCommand:*

```
public class ShowMessagesCommand implements Command {

   public Object execute(...) throws CommandException {
        try {
            MessageBean[] beanArray = dao.retrieveAll();
            request.setAttribute("messages", beanArray);
            return successUrl;
        } catch (PersistenceException e) {
            throw new CommandException(e);
        }
   } (...)
```

Data Access Objects (DAO)

- *Isolam a camada de persistência*
  - *Implementamos persistência JAXB, mas outra pode ser utilizada (SGBDR) sem precisar mexer nos comandos.*
- *Interface da DAO:*

```
public interface MessageBeanDAO {
   public Object getLocator();

   public void persist(MessageBean messageBean)
                              throws PersistenceException;

   public MessageBean retrieve(int key)
                              throws PersistenceException;

   public MessageBean[] retrieveAll()
                              throws PersistenceException;

   public MessageBean retrieveLast()
                              throws PersistenceException;
}
```

Controlador (FrontController)

- *Na nossa aplicação, o controlador é um servlet que recebe os nomes de comandos, executa os objetos que os implementam e repassam o controle para a página JSP ou HTML retornada.*

```
public void service( ..., ... ) ... {
    Command command = null;
    String commandName = request.getParameter("cmd");

    if (commandName == null) {
        command = commands.getCommand("default");
    } else {
        command = commands.getCommand(commandName);
    }

    Object result = command.execute(request, response, dao);
    if (result instanceof String) {
        RequestDispatcher dispatcher =
              request.getRequestDispatcher((String)result);
        dispatcher.forward(request, response);
    }
   ...
}
```

Método de CommandFactory

Execução do comando retorna uma URI

Repassa a requisição para página retornada

ValueBean ViewHelper (Model)

- *Este bean é gerado em tempo de compilação a partir de um DTD (usando ferramentas do JAXB)*

```
public class MessageBean
    extends MarshallableRootElement
    implements RootElement {

    private String _Time;
    private String _Host;
    private String _Message;

    public String getTime() {...}
    public void setTime(String _Time) {...}

    public String getHost() {...}
    public void setHost(String _Host) {...}

    public String getMessage() {...}
    public void setMessage(String _Message) {...}

    ...
}
```

interfaces JAXB permitem que este bean seja gravado em XML (implementa métodos marshal() e unmarshal() do JAXB)

Página JSP (View) com custom tags

- *Página messages.jsp (mostra várias mensagens)*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib uri="/hellotags" prefix="hello" %>
<html>
<head><title>Show All Messages</title></head>
<body>
<jsp:include page="navbar.html" />
<h1>Messages sent so far</h1>
<table border="1">
<tr><th>Time Sent</th><th>Host</th><th>Message</th></tr>

<hello:printMessages array="messages">
    <tr>
        <td><hello:printField property="time" /></td>
        <td><hello:printField property="host" /></td>
        <td><hello:printField property="message" /></td>
    </tr>
</hello:printMessages>

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

Para executar o exemplos

- 1. Mude para exemplos/cap12/mvc/hellojsp_2
- 2. Configure build.properties, depois rode

```
> ant DEPLOY
```

- 3. Inicie o servidor (Tomcat ou JBoss)
- 4. Rode os testes do Cactus

```
> ant RUN-TESTS
```

- 5. Rode a aplicação, acessando a URI

```
http://localhost:porta/hellojsp/
```

- 6. Digite mensagens e veja resultados. Arquivos são gerados em /tmp/mensagens (ou c:\tmp\mensagens)

Exercício

- *1. Coloque para funcionar o exemplo e analise suas classes*
- *2. Implemente a aplicação de mensagens que você criou nos capítulos anteriores em MVC*
  - *Crie um servlet controlador para interceptar todas as requisições*
  - *Crie páginas JSP que leiam os beans e que enviem comandos desejados através de um parâmetro cmd*
  - *Crie uma interface Comando e os comandos ComandoListar e ComandoAdicionar. Coloque-os em um HashMap inicializada no init() do servlet*
  - *Implemente o service que obtenha o parâmetro cmd, localize o comando desejado e o execute.*

helder@acm.org

argonavis.com.br

Padrões de Projeto J2EE para Aplicações Web

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Introdução

- Este módulo aborda os principais padrões de projeto J2EE, dentre o catálogo organizado pelo Sun Java Center (SJC) que são aplicáveis a aplicações Web
  - É um módulo de referência. A abordagem, neste curso, será superficial
- Os padrões representam boas práticas e estratégias de implementação para diversos problemas recorrentes no design de aplicações Web
- Conhecer os padrões ajuda a entender melhor sistemas semelhantes, especialmente frameworks
  - Consulte também os padrões GoF, aplicáveis não só à plataforma J2EE mas a OO

Padrões para Aplicações Web

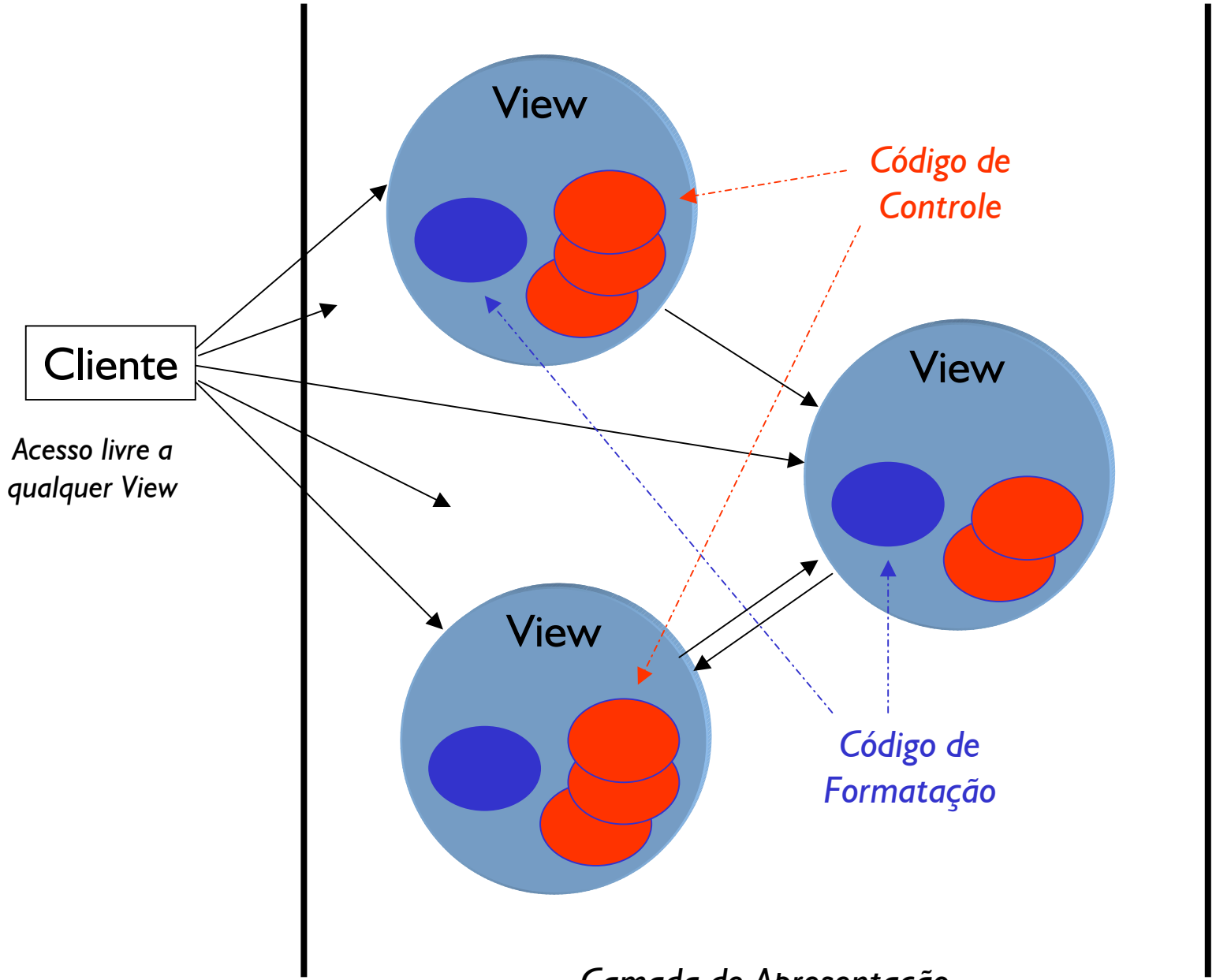
- (1) Front Controller
  - Controlador centralizado para processar de uma requisição
- (2) View Helper
  - Encapsula lógica não-relacionada à formatação
- (3) Composite View
  - Cria uma View composta de componentes menores
- (4) Intercepting Filter
  - Viabiliza pré- e pós-processamento de requisições
- (5) Data Access Object
  - Esconde detalhes do meio de persistência utilizado
- (6) Business Delegate
  - Interface com a camada de negócios
- (7) Transfer Object (ou Value Object)
  - Objeto que é utilizado na comunicação para evitar múltiplas requisições e respostas

1

Front Controller

Objetivo: centralizar o processamento de requisições em uma única fachada. Front Controller permite criar uma interface genérica para processamento de comandos.

Problema


View
Código de Controle
Cliente
View
Acesso livre a qualquer View
View
Código de Formatação
Camada de Apresentação

Solução: Front Controller

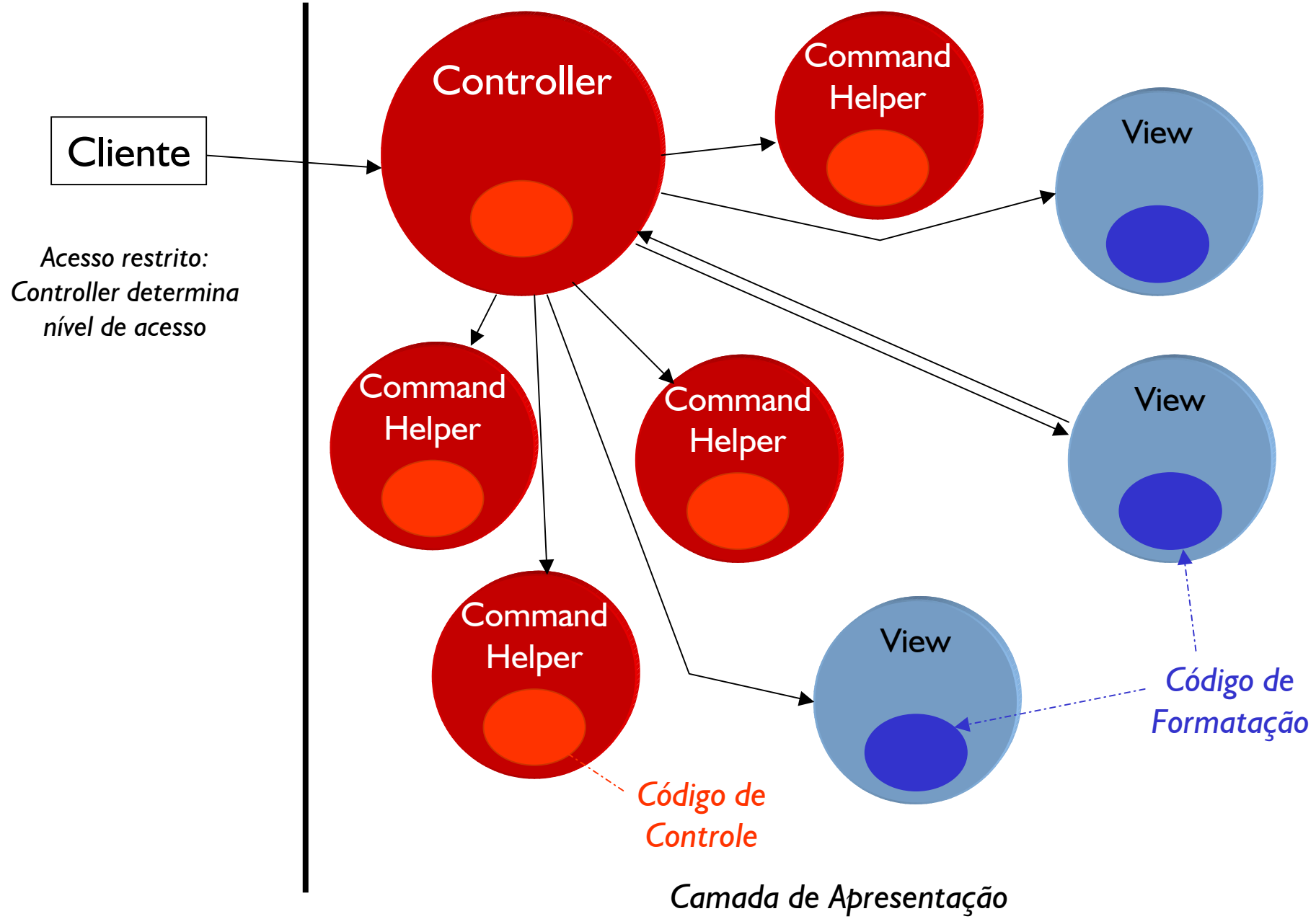


Diagrama de Seqüência

```
1.1    RequestDispatcher rd = request.getRequestDispatcher("View.jsp");
1.1.1  rd.forward(request, response);
```

Fonte: [SJC]

Participantes e responsabilidades

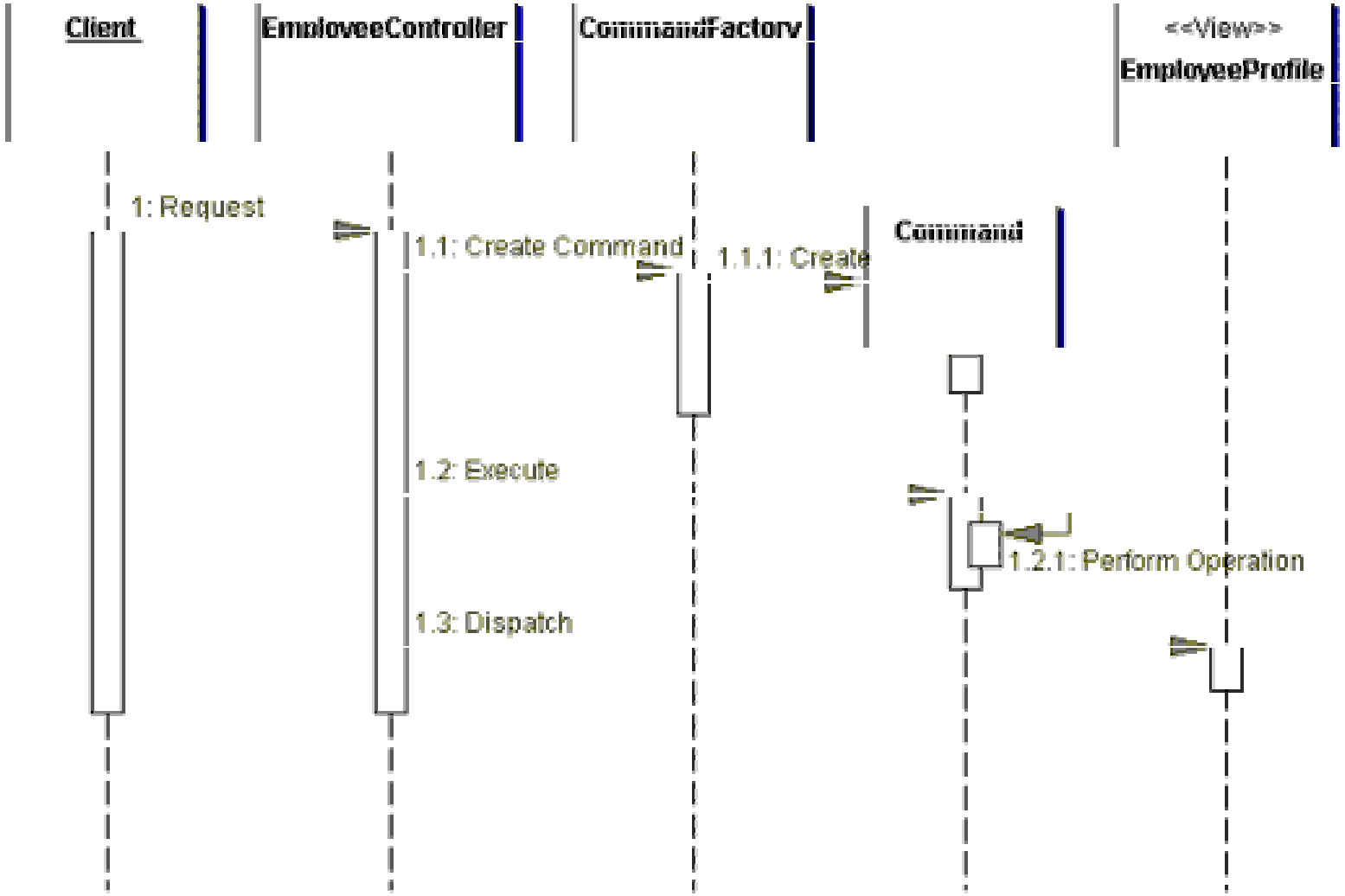
- *Controller*
  - *Ponto de entrada que centraliza todas as requisições*
  - *Pode delegar responsabilidade a Helpers*
- *Dispatcher*
  - *Tipicamente usa ou encapsula objeto javax.servlet.RequestDispatcher*
- *Helper*
  - *Pode ter inúmeras responsabilidades, incluindo a obtenção de dados requerido pelo View*
  - *Pode ser um Value Bean, Business Delegate, Command, ...*
- *View*
  - *Geralmente página JSP*

Melhores estratégias de implementação*

- Servlet Front Strategy
  - Implementa o controlador como um servlet.
  - Dispatcher and Controller Strategy implementa o Dispatcher dentro do próprio servlet
- Command and Controller Strategy
  - Interface baseada no padrão Command (GoF) para implementar Helpers para os quais o controlador delega responsabilidades.
- Logical Resource Mapping Strategy
  - Requisições são feitas para nomes que são mapeados a recursos (páginas JSP, servlets) ou comandos
  - Multiplexed Resource Mapping Strategy usa wildcards para selecionar recursos a serem processados

* Veja exemplos de código em exemplos/preslayer/

Command and Controller Strategy



Fonte: [Core]

Conseqüências

- Controle centralizado
  - Facilidade de rastrear e logar requisições
- Melhor gerenciamento de segurança
  - Requer menos recursos. Não é preciso distribuir pontos de verificação em todas as páginas
  - Validação é simplificada
- Melhor possibilidade de reuso
  - Distribui melhor as responsabilidades

Exercícios

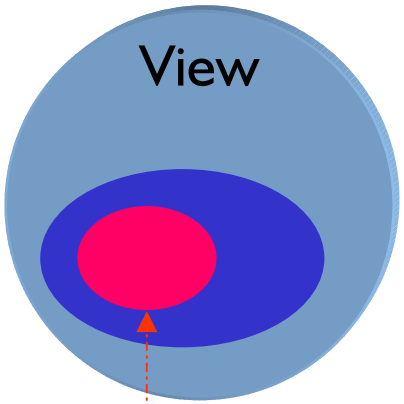
- 1. Altere a aplicação em cap13/fc/ para que utilize FrontController. Empregue as três estratégias de implementação apresentadas:
  - a) Implemente o controlador usando um Servlet
  - b) Escreva um RequestHelper que mantenha uma tabela de comandos/nomes de classe de objetos Command e receba um request na construção. Seu método getCommand() deve retornar o comando correspondente recebendo newMessage, lastMessage, allMessages
  - c) Configure o web.xml para mapear todas as requisições ao controlador

2

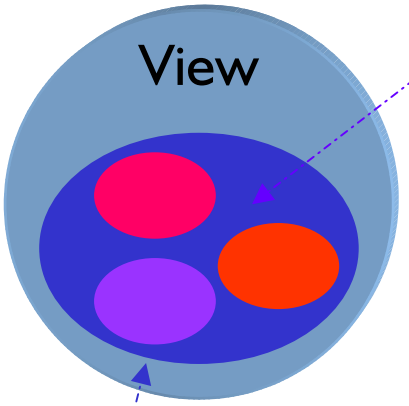
View Helper

Objetivo: separar código e responsabilidades de formatação da interface do usuário do processamento de dados necessários à construção da View. Tipicamente implementados como JavaBeans e Custom Tags.

Problema



Código de Conversão de Dados



Código de Lógica de Negócio e de Controle

Código de Formatação

Solução: View Helpers

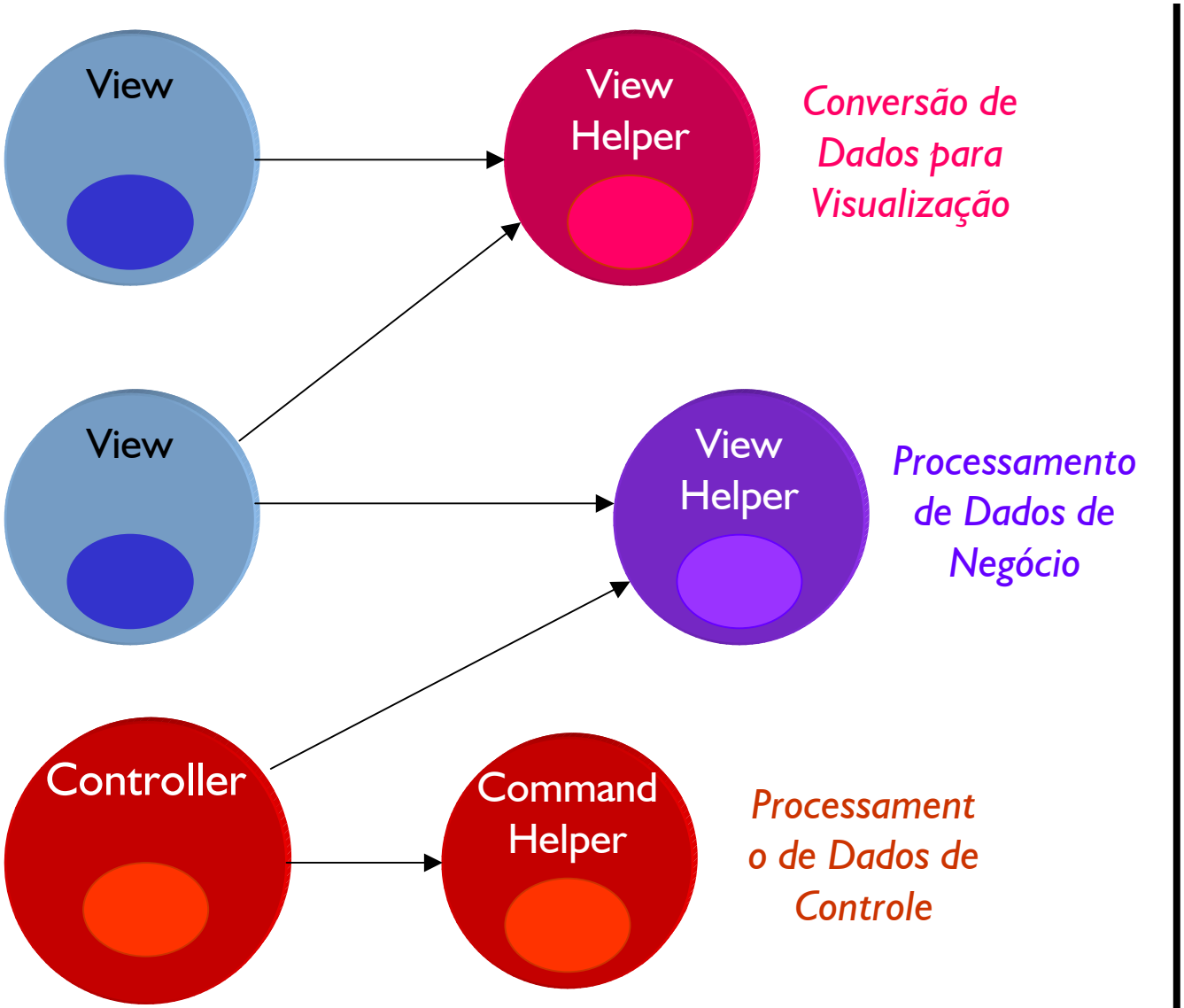


Diagrama de Seqüência

Fonte: [SJC]

Melhores estratégias de implementação

- JSP View Strategy
  - JSP é componente de View
- JavaBean Helper Strategy
  - Helper implementado como JavaBean
- Custom Tag Helper Strategy
  - Mais complexo que JavaBean Helper
  - Separação de papéis maior (isola a complexidade)
  - Maior índice de reuso (pode-se usar custom tags existentes)
- Business Delegate as Helper Strategy
  - Papéis de View Helper e Business Delegate podem ser combinados para acesso à camada de negócio
  - Pode misturar papéis J2EE

Conseqüências

- Melhora particionamento da aplicação
  - Facilita o reuso
  - Facilita a manutenção
  - Facilita a realização de testes funcionais, de unidade e de integração
- Melhora separação de papéis J2EE
  - Reduz a complexidade para todos os participantes: Web Designer não precisa ver Java e Programador Java não precisa ver JavaScript e HTML

Exercícios

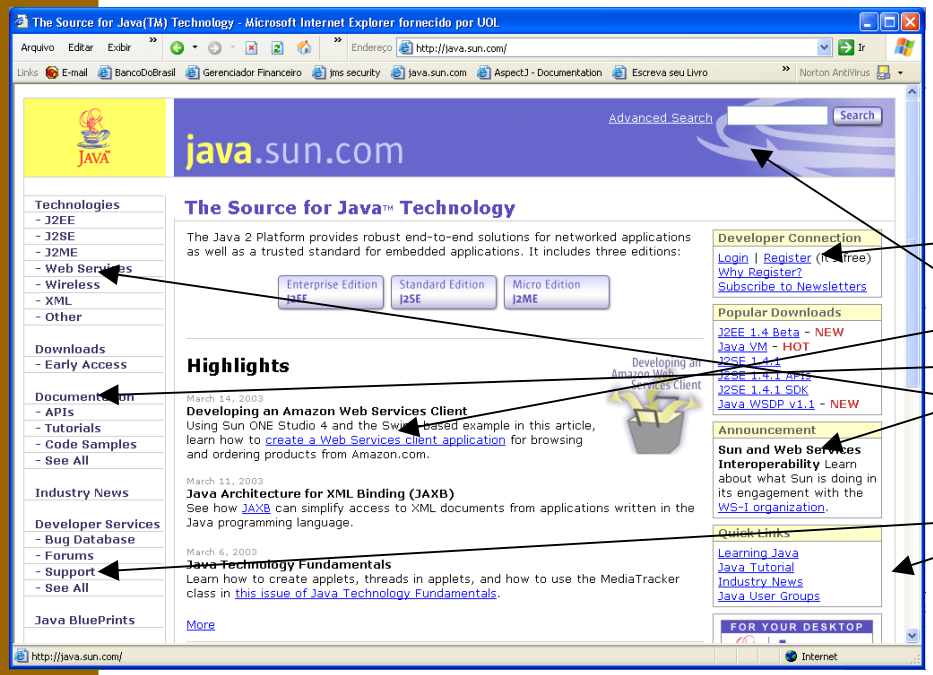
- 2. Altere a aplicação em cap13/vh/ para que utilize ViewHelper. Use JavaBean Helper Strategy:
  - a) Identifique código de conversão de formatos, código de negócio e código de controle (se houver)
  - b) Construa um Helper para cada responsabilidade encontrada
- 3. Use Custom Tag Helper Strategy para encapsular a lógica de repetição
  - Use tags do Struts <logic:iterate> ou JSTL <c:forEach>

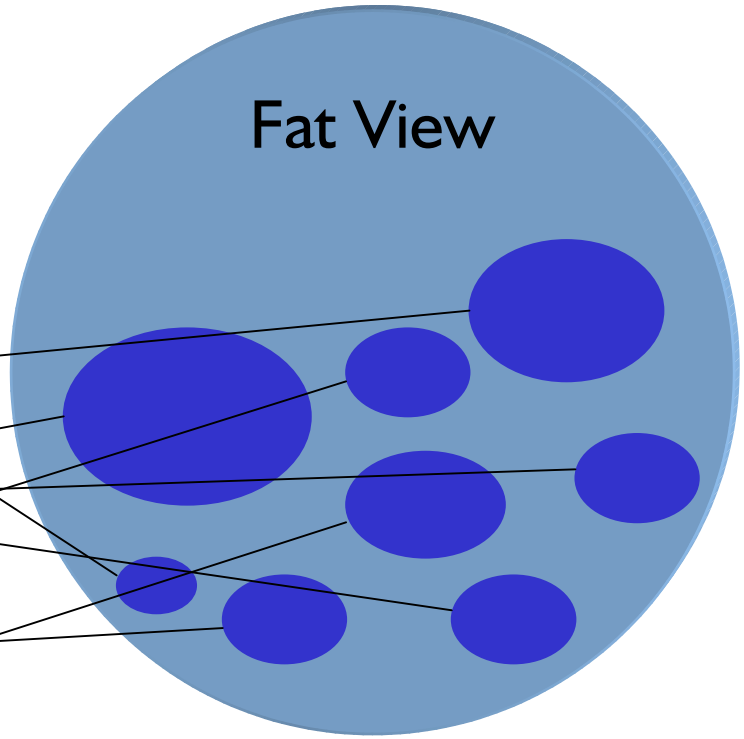
3

Composite View

Objetivo: criar um componente de View a partir de Views menores para dividir as responsabilidades, simplificar a construção da interface e permitir o reuso de componentes da View.

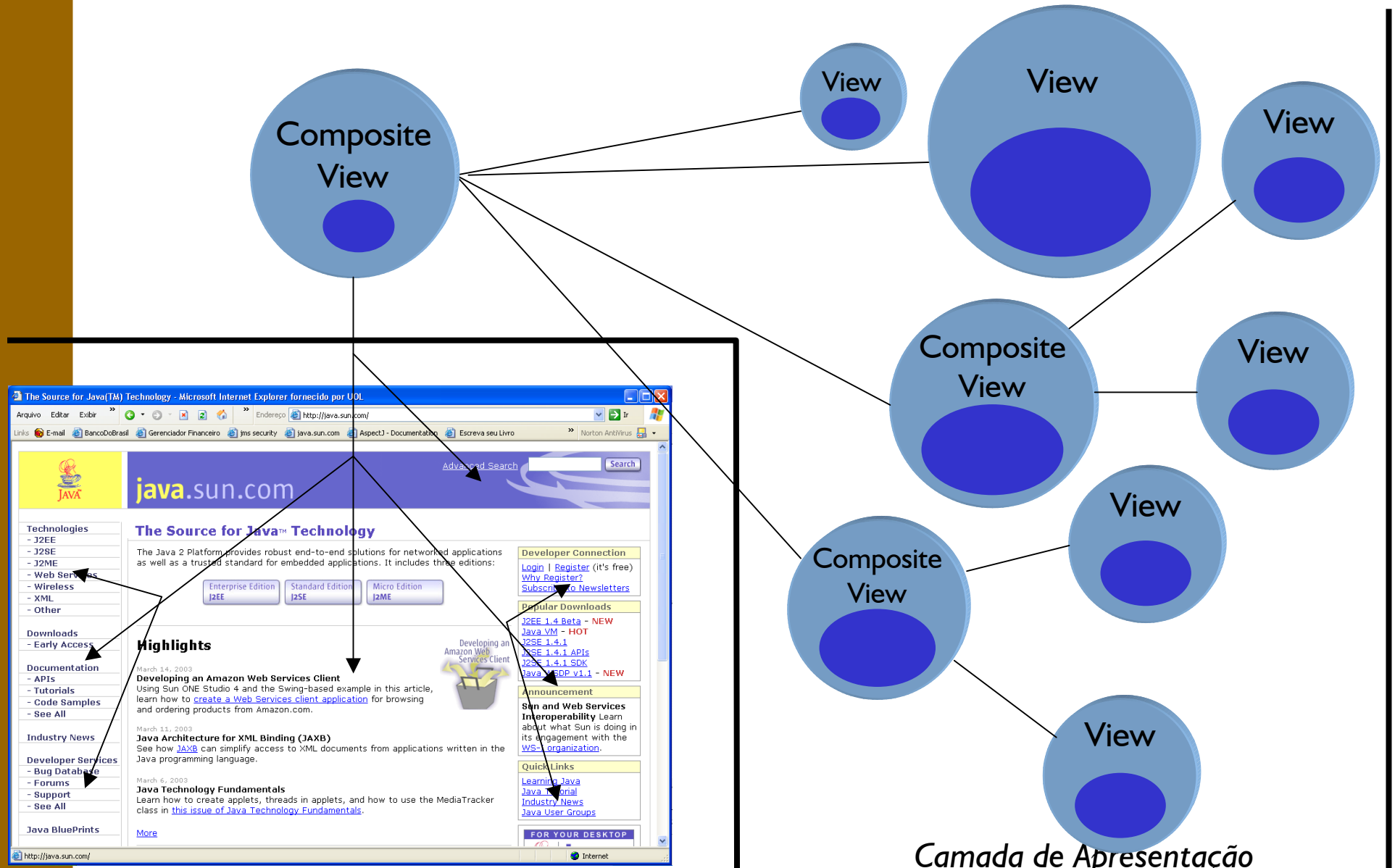
Problema





Camada de Apresentação

Solução: Composite View



Diagramas de Seqüência

Fonte: [SJC]

Participantes e responsabilidades

- *Composite View*
  - *Agregado composto de sub-views*
- *View Manager*
  - *Gerencia a inclusão de porções de fragmentos de template no Composite View*
  - *Geralmente parte do processador JSP mas pode ser implementado também como JavaBean*
- *Included View*
  - *Sub view que pode ser uma view final ou uma composição de views menores*

Melhores estratégias de implementação

- *JavaBean View Management Strategy*
  - *Utiliza JavaBeans para incluir outros views na página*
  - *Mais simples que solução com Custom Tags*
- *Early Binding Resource Strategy (Translation-time)*
  - *Usa tags padrão: <%@ include %> e <%@ file %>*
  - *Carga é feita em tempo de compilação: alterações só são vistas quando página for recompilada*
- *Late Binding Resource Strategy (Run-time)*
  - *Usa tag padrão do JSP: <jsp:include>*
  - *Carga é feita quando página é carregada: alterações são visíveis a cada atualização*
- *Custom Tag View Management Strategy (7.23)*
  - *Utiliza Custom Tags: solução mais elegante e reutilizável*

Conseqüências

- *Promove design modular*
  - *Permite maior reuso e reduz duplicação*
- *Melhora flexibilidade*
  - *Suporta inclusão de dados com base em decisões de tempo de execução*
- *Melhora facilidade de manutenção e gerenciamento*
  - *Separação da página em pedaços menores permite que sejam modificados e mantidos separadamente*
- *Reduz facilidade de gerenciamento*
  - *Possibilidade de erros na apresentação devido à composição incorreta das partes*
- *Impacto na performance*
  - *Inclusões dinâmicas fazem página demorar mais para ser processada*

Exercícios

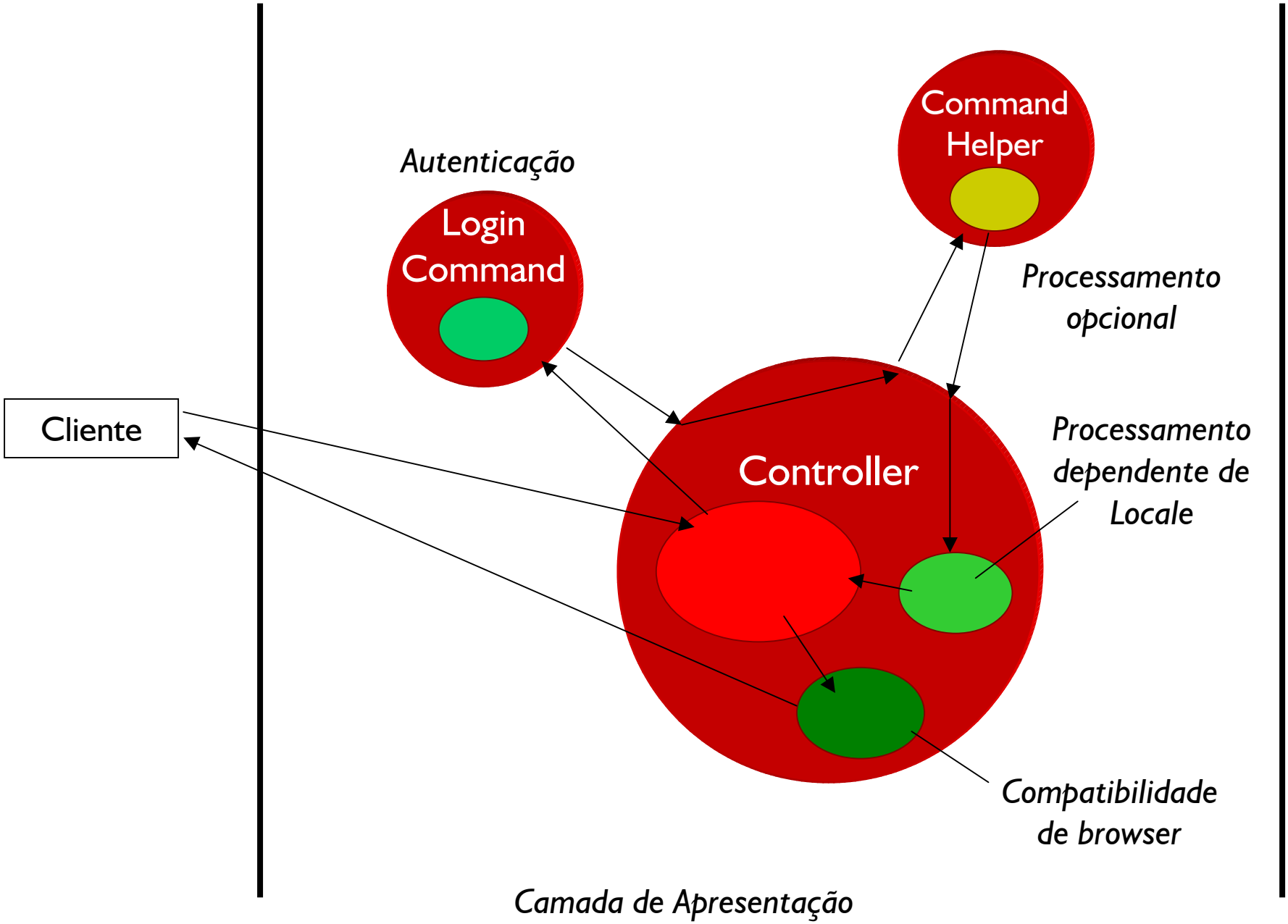
- *4. Refatore a aplicação em cap13/cv/ para que utilize CompositeView (os blocos estão identificados com comentários no HTML em messages.jsp). Escolha as melhores estratégias entre Translation-time e Run-time Stategies*
  - *a) Qual a melhor estratégia para o navbar (raramente muda)?*
  - *b) E para o bloco principal?*
- *5. Implemente o menu usando Custom Tag View Management Strategy*
- *6. Implemente o bloco de mensagens usando JavaBean View Management Strategy (já está implementado)*

4

Intercepting Filter

Objetivo: permitir o pré- e pós processamento de uma requisição. Intercepting Filter permite encaixar filtros decoradores sobre a requisição ou resposta e remover código de transformação da requisição do controlador

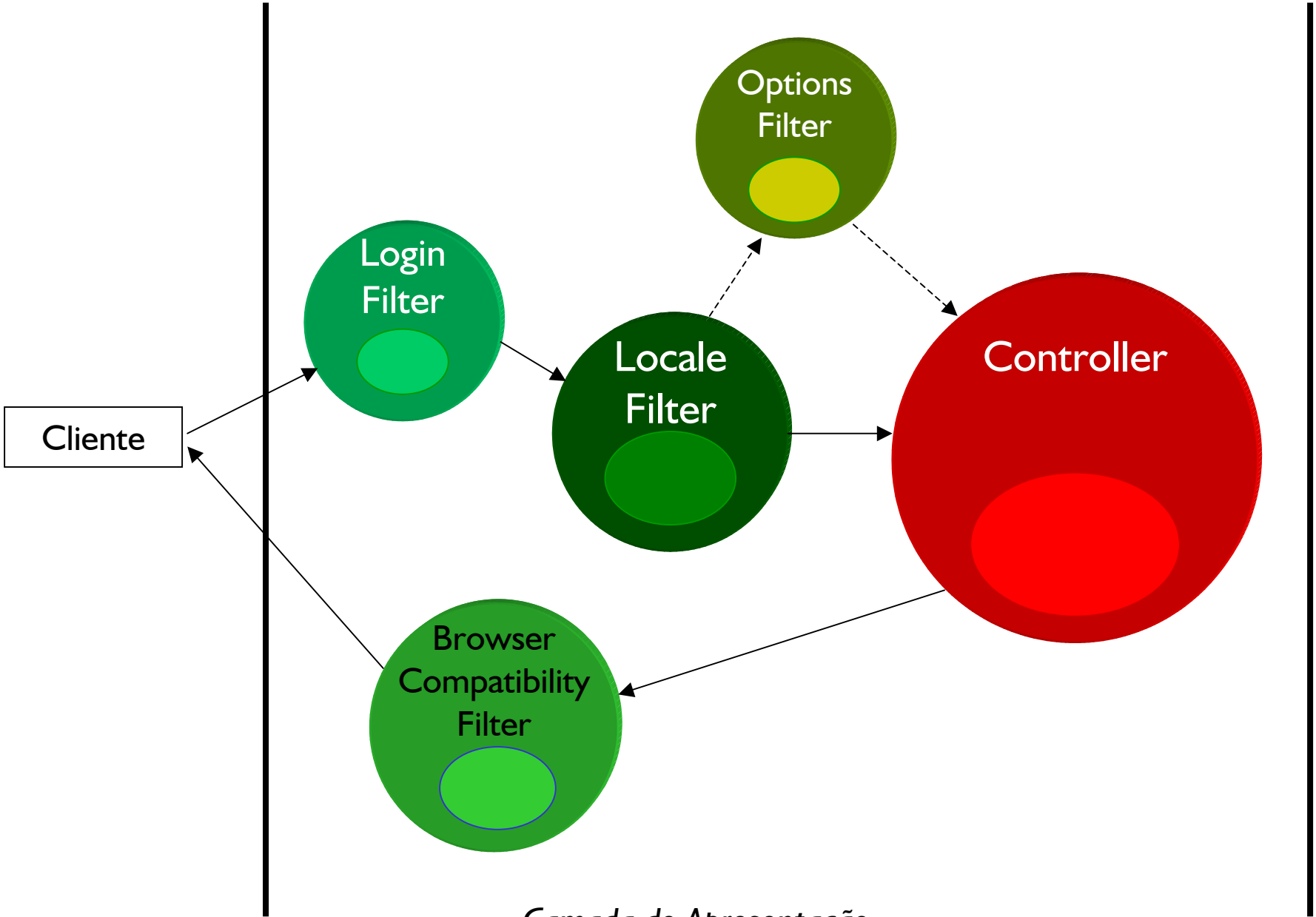
Problema


Command
Helper
Autenticação
Login
Command
Processamento
opcional
Cliente
Controller
Processamento
dependente de
Locale
Compatibilidade
de browser
Camada de Apresentação

Descrição do problema

- A camada de apresentação recebe vários diferentes tipos de requisições, que requerem processamento diferenciado
- No recebimento de uma requisição, várias decisões precisam ser tomadas para selecionar a forma de realização do processamento
  - Isto pode ser feito diretamente no controlador via estruturas if/else. Desvantagem: embute fluxo da filtragem no código compilado, dificultando a sua remoção ou adição
  - Incluir tratamento de serviços no próprio controlador impede que esse código possa ser reutilizado em outros contextos

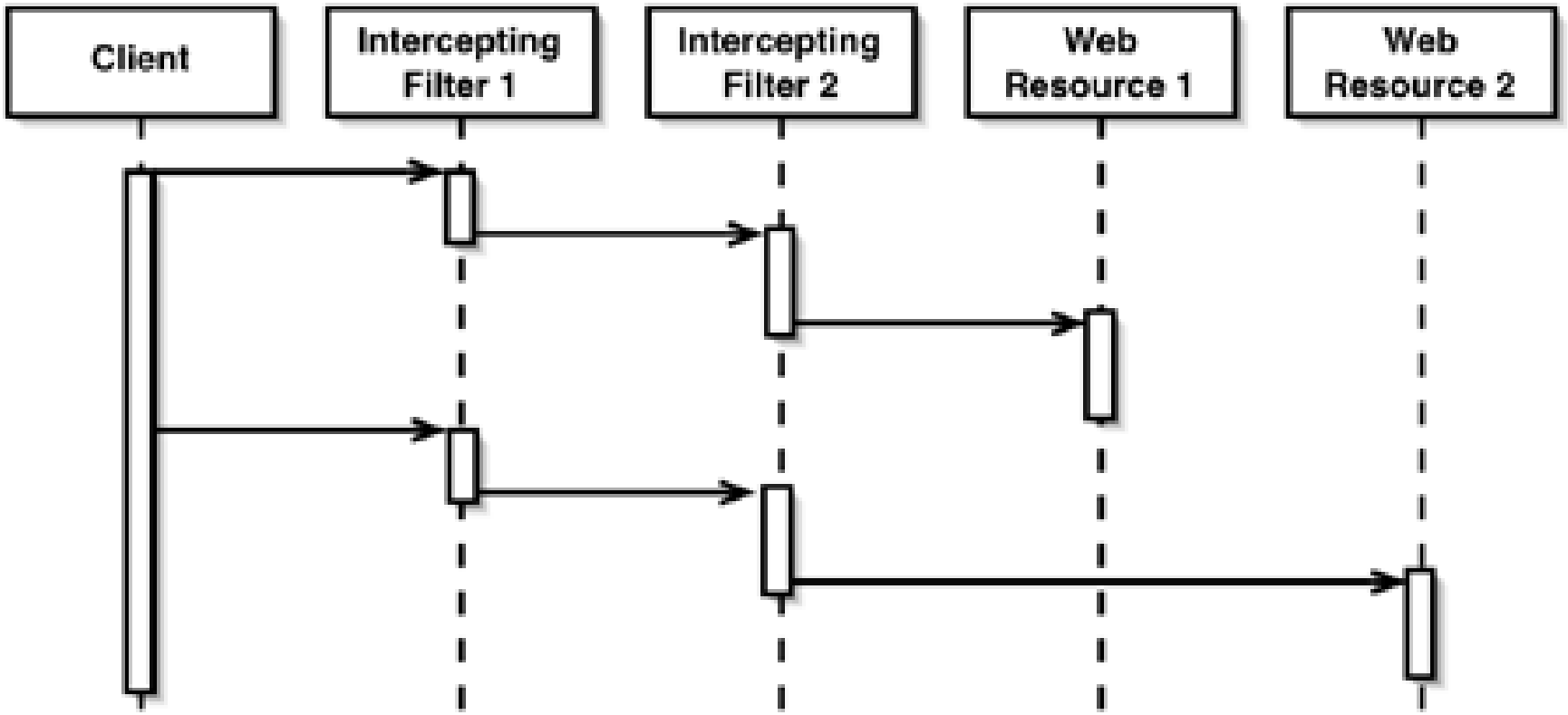
Solução: Intercepting Filter



Descrição da solução

- *Criar filtros plugáveis para processar serviços comuns de forma padrão, sem requerer mudanças no código de processamento*
  - *Filtros interceptam requisições entrantes e respostas, viabilizando pré- e pós-processamento*
  - *Filtros podem ser incluídos dinamicamente e sua composição pode ser alterada*
  - *Filtros são uma estrutura implementada na API Servlet 2.3 (veja cap. 6)*

Exemplo de solução



Fonte: [Blueprints]

Diagrama de Seqüência

Fonte: [SJC]

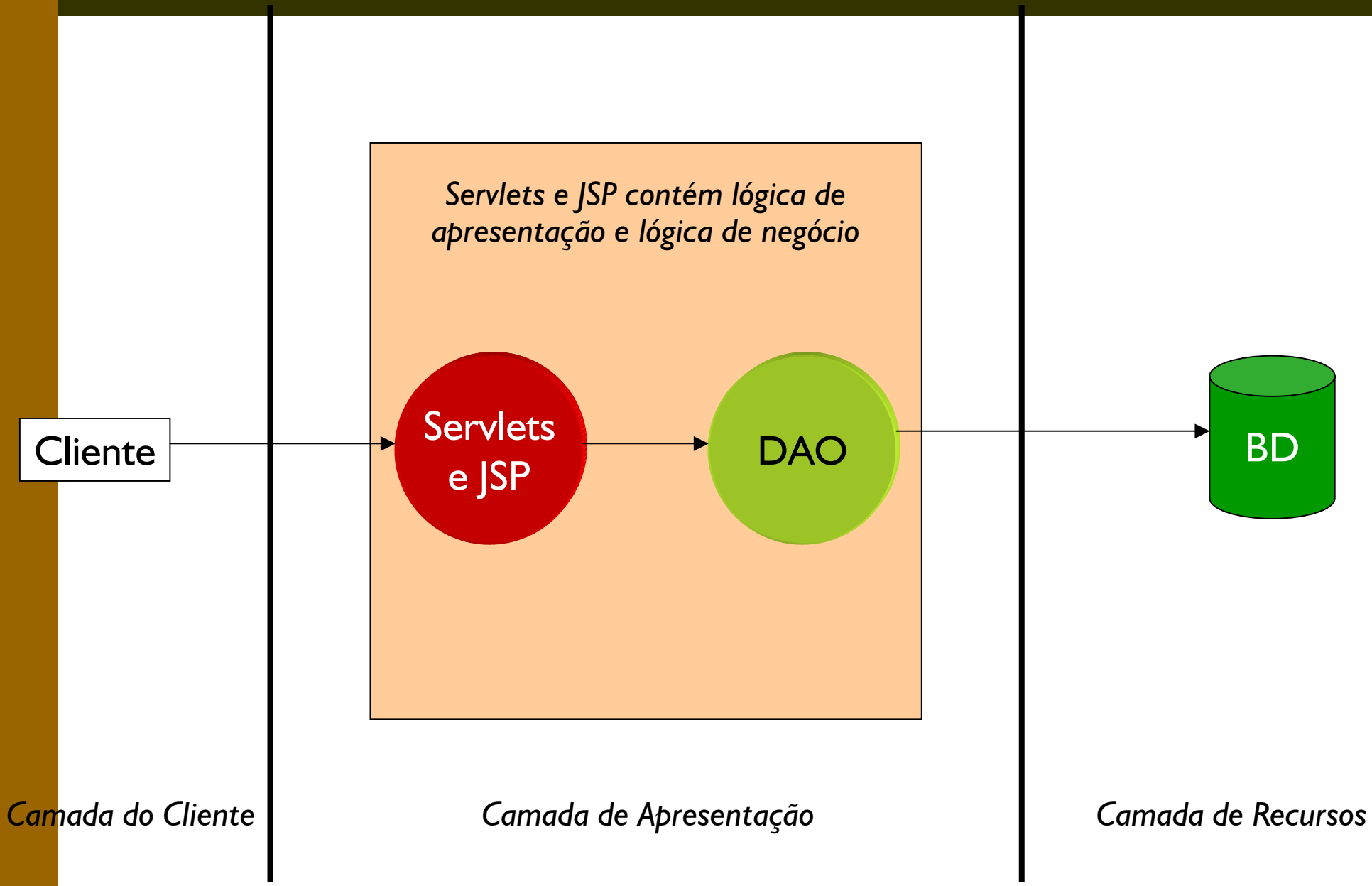
Conseqüências

- Centraliza controle com processadores fracamente acoplados
  - Como um controlador, fornecem um ponto centralizado para processamento de requisições
  - Podem ser removidos, adicionados, combinados em cascata
- Melhora reuso
  - Filtros são destacados do controlador e podem ser usados em outros contextos
- Configuração declarativa e flexível
  - Serviços podem ser reorganizados sem recompilação
- Compartilhamento ineficiente de informações
  - Se for necessário compartilhar informações entre filtros, esta solução não é recomendada

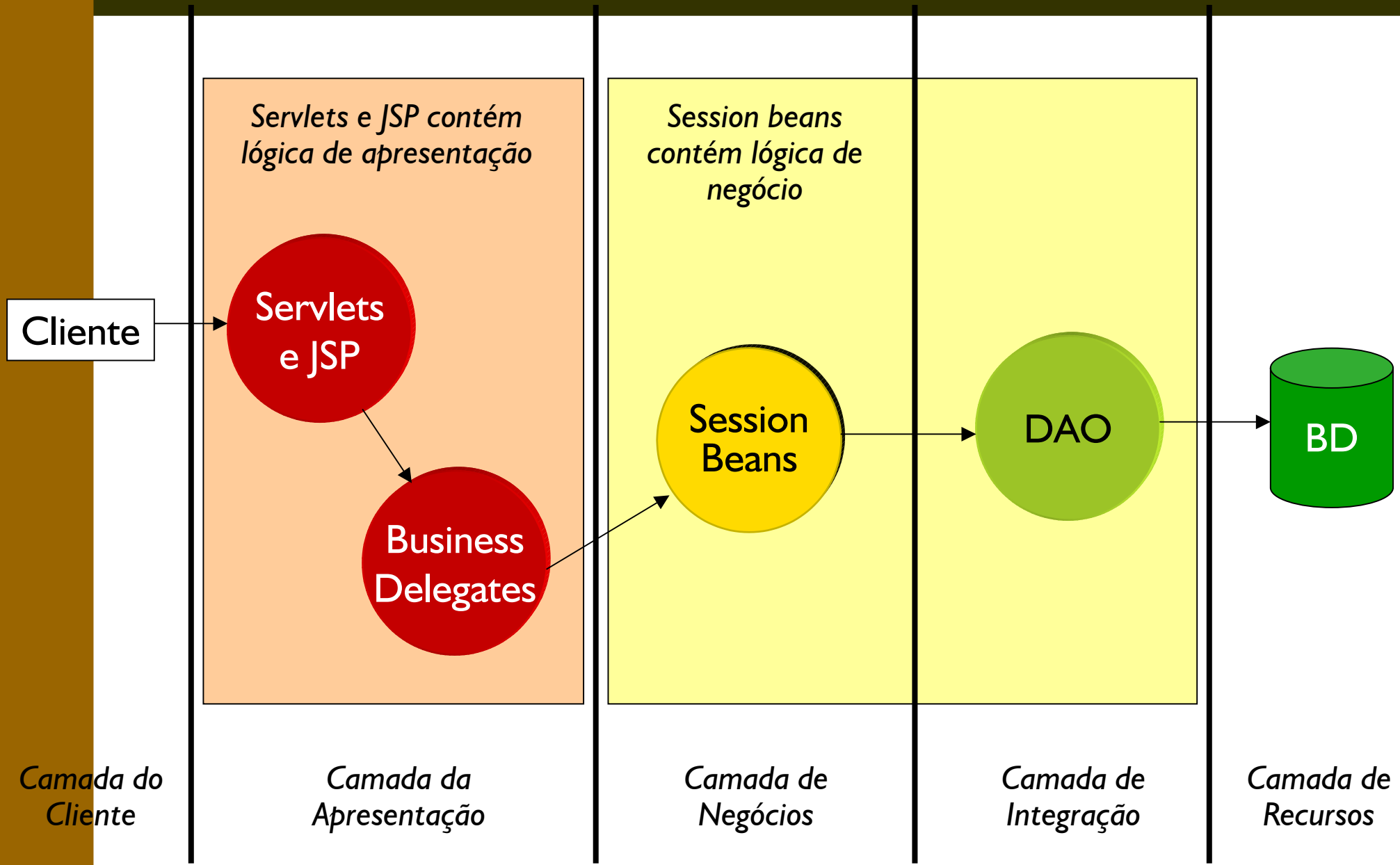
Exercícios

- 7. Refatore a aplicação em cap13/if/ para que utilize Intercepting Filter:
  - a) A página login.jsp é chamada se o LoginBean for null. Implemente esta funcionalidade usando um filtro
  - b) Implemente um filtro que coloque os parâmetros de entrada em caixa-alta
  - c) Experimente com composição de filtros no deployment descriptor

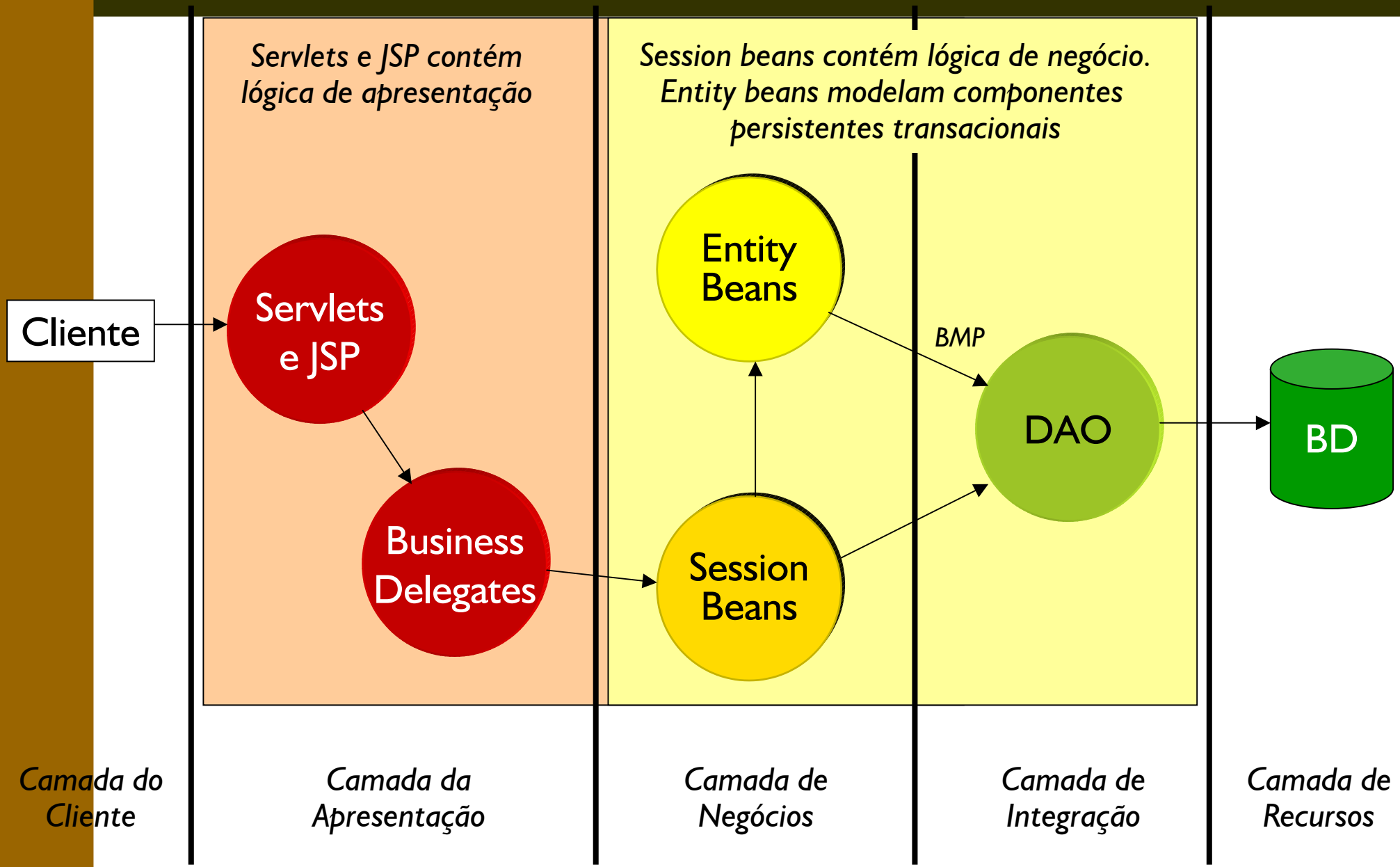
Refatoramento por Camadas (I)



Refatoramento por Camadas (2)



Refatoramento por Camadas (3)

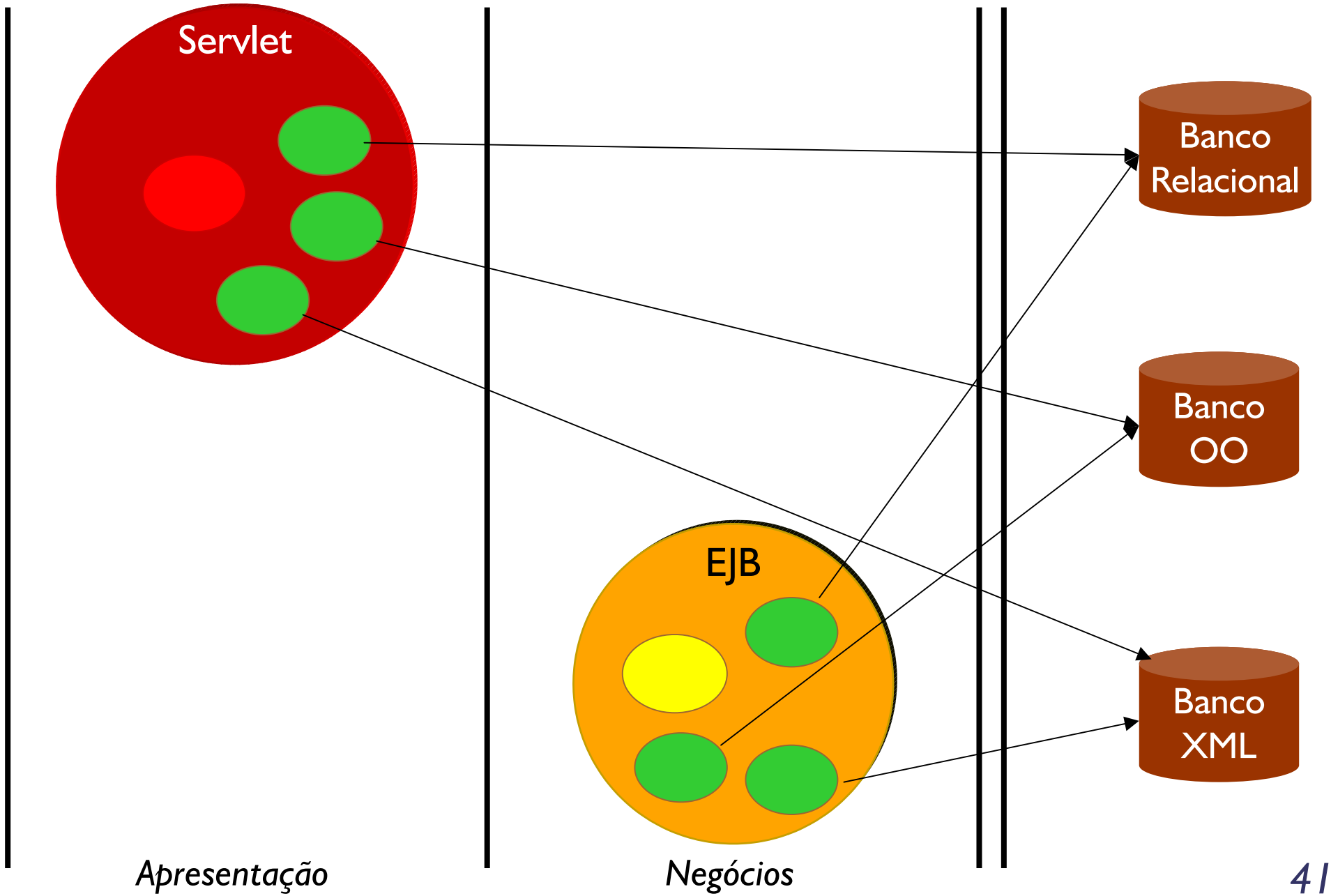


5

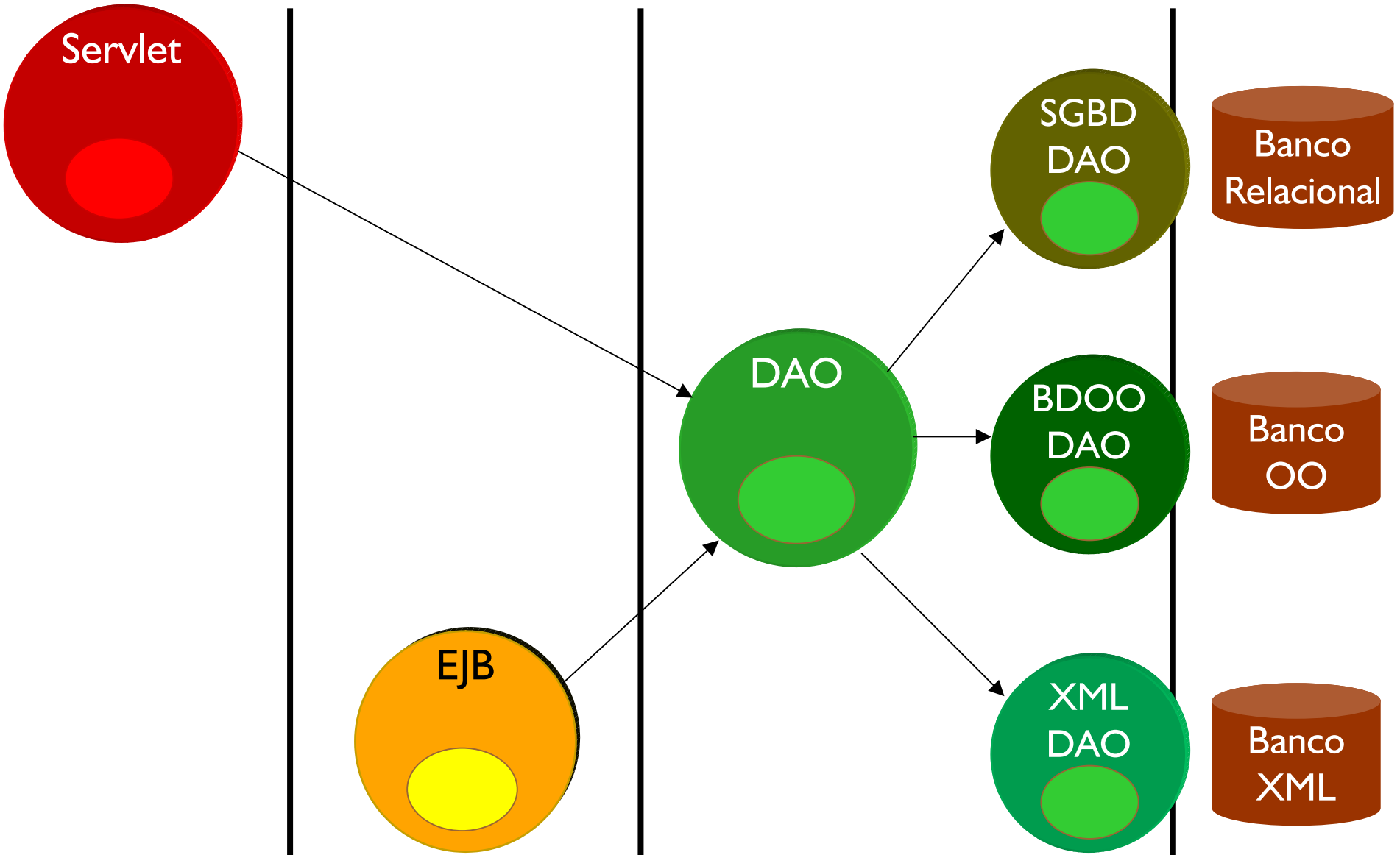
Data Access Object (DAO)

Objetivo: Abstrair e encapsular todo o acesso a uma fonte de dados. O DAO gerencia a conexão com a fonte de dados para obter e armazenar os dados.

Problema


Servlet
Banco Relacional
Banco OO
EJB
Banco XML
Apresentação
Negócios

Solução: Data Access Object



Conseqüências

- *Transparência quanto à fonte de dados*
- *Facilita migração para outras implementações*
- *Reduz complexidade do código nos objetos de negócio (ex: Entity Beans BMP e servlets)*
- *Centraliza todo acesso aos dados em camada separada*
- *Requer design de hierarquia de classes (Factory)*

Exercícios

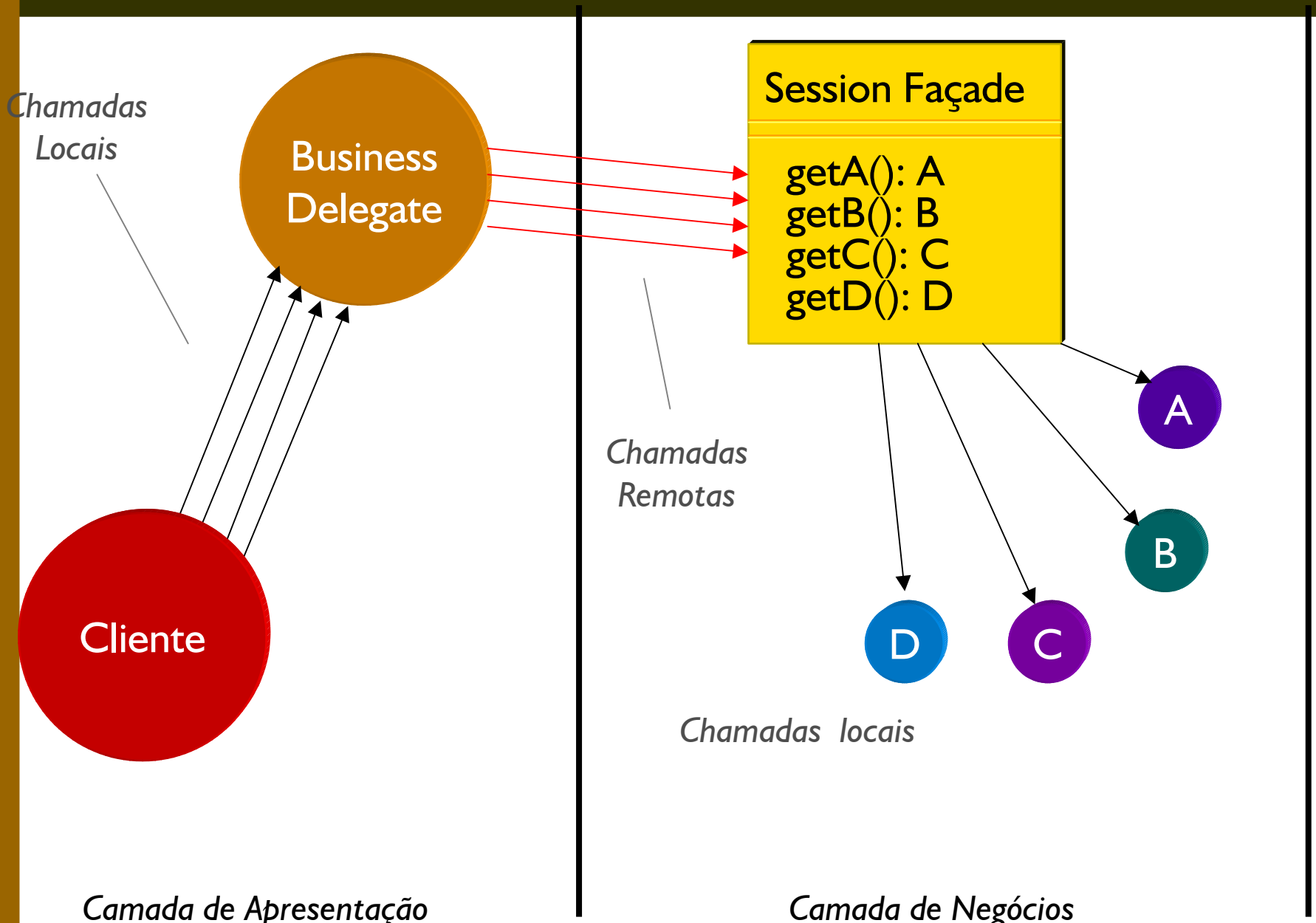
- *8. Analise o código do DAO existente (XML) e implemente um DAO e código para armazenar as mensagens no banco de dados Cloudscape:*
  - *a) Implemente a interface MessageBeanDAO*
  - *b) Implemente um mecanismo de seleção do meio de persistência escolhido através do web.xml e um Factory Method através do qual a aplicação possa selecionar o DAO desejado*

8

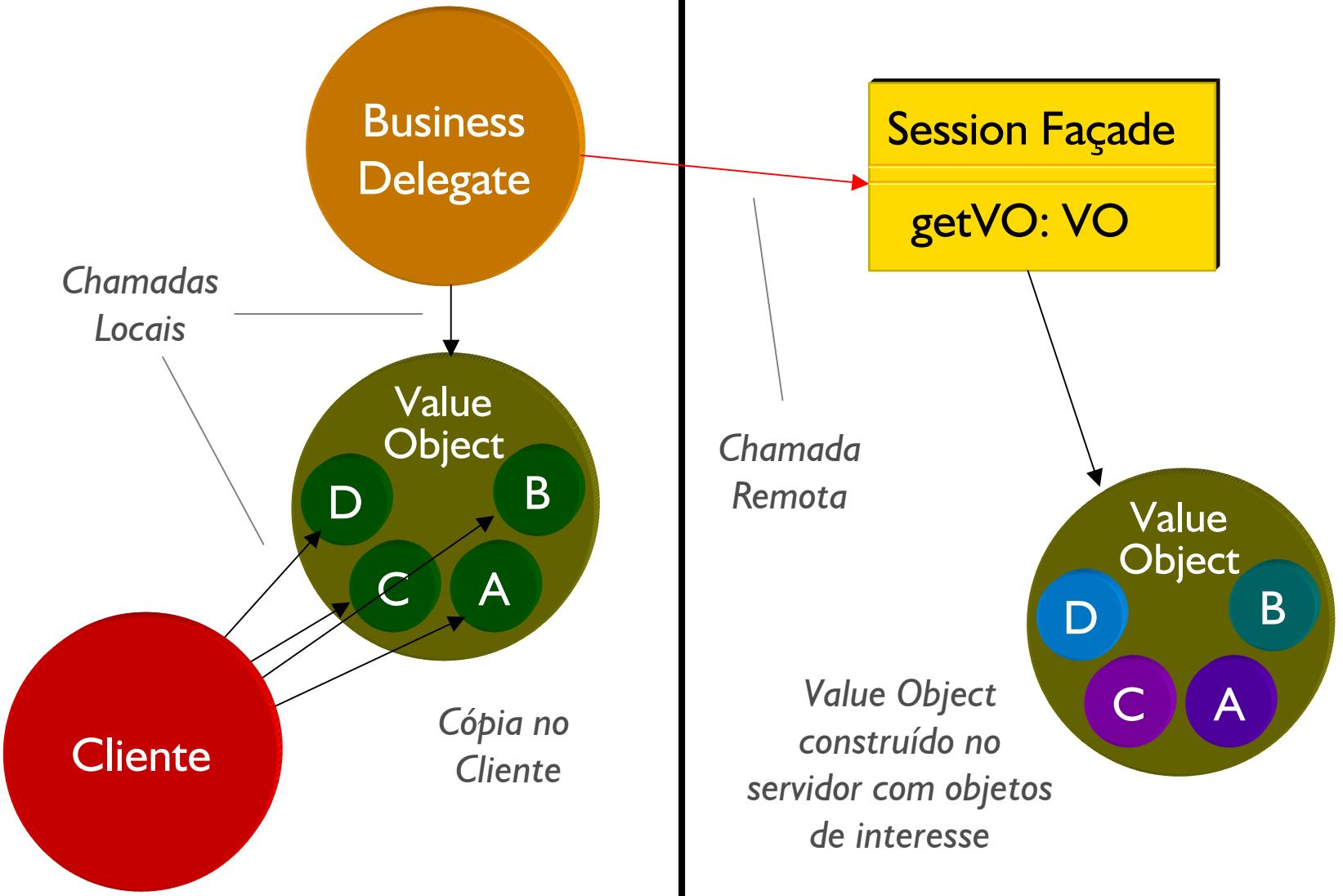
Value Object ou Transfer Object

Objetivo: Reduzir a quantidade de requisições necessárias para recuperar um objeto. Value Object permite encapsular em um objeto um subconjunto de dados utilizável pelo cliente e utilizar apenas uma requisição para transferi-lo.

Problema


Chamadas
Locais
Business
Delegate
Session Façade
getA(): A
getB(): B
getC(): C
getD(): D
A
B
C
D
Chamadas
Remotas
Chamadas locais
Cliente
Camada de Apresentação
Camada de Negócios

Solução: Value Object



Conseqüências

- Simplifica DAO, EJBs e interface remota
- Transfere mais dados em menos chamadas
- Reduz tráfego de rede
- Reduz duplicação de código
- Pode introduzir objetos obsoletos
- Pode aumentar a complexidade do sistema
  - Sincronização
  - Controle de versões para objetos serializados

Exercícios

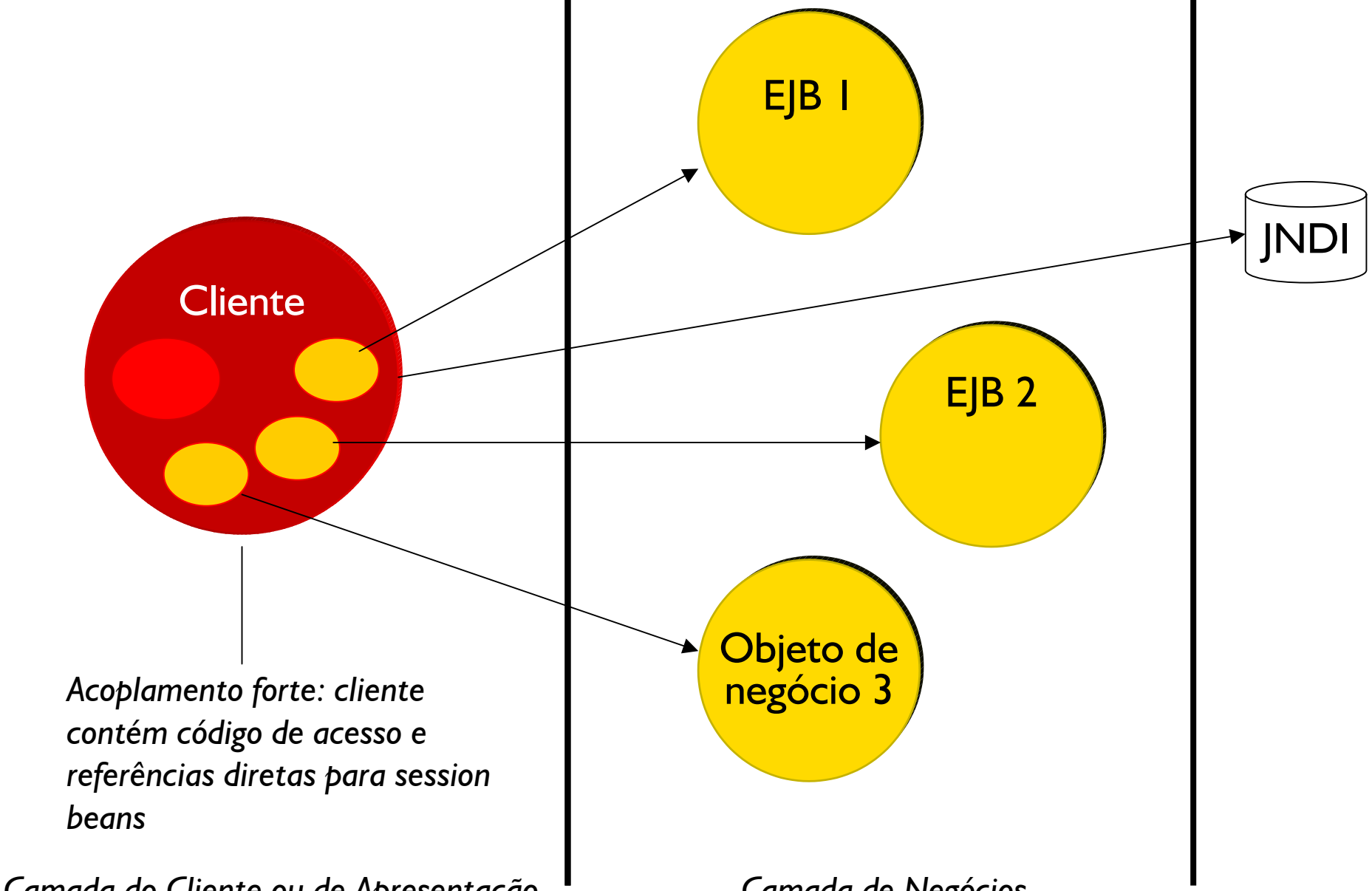
- 9. Refatore a aplicação em cap13/vo/ para que utilize Value Object
  - a) Crie um Value Object que representa uma cópia do objeto MensagemBean
  - b) Implemente no Façade um método que retorne o Value Object para o cliente, e outro que o receba de volta a atualize os dados corretamente.
  - b) Refatore o cliente para que ele use esse objeto e extraia os dados corretamente.

7

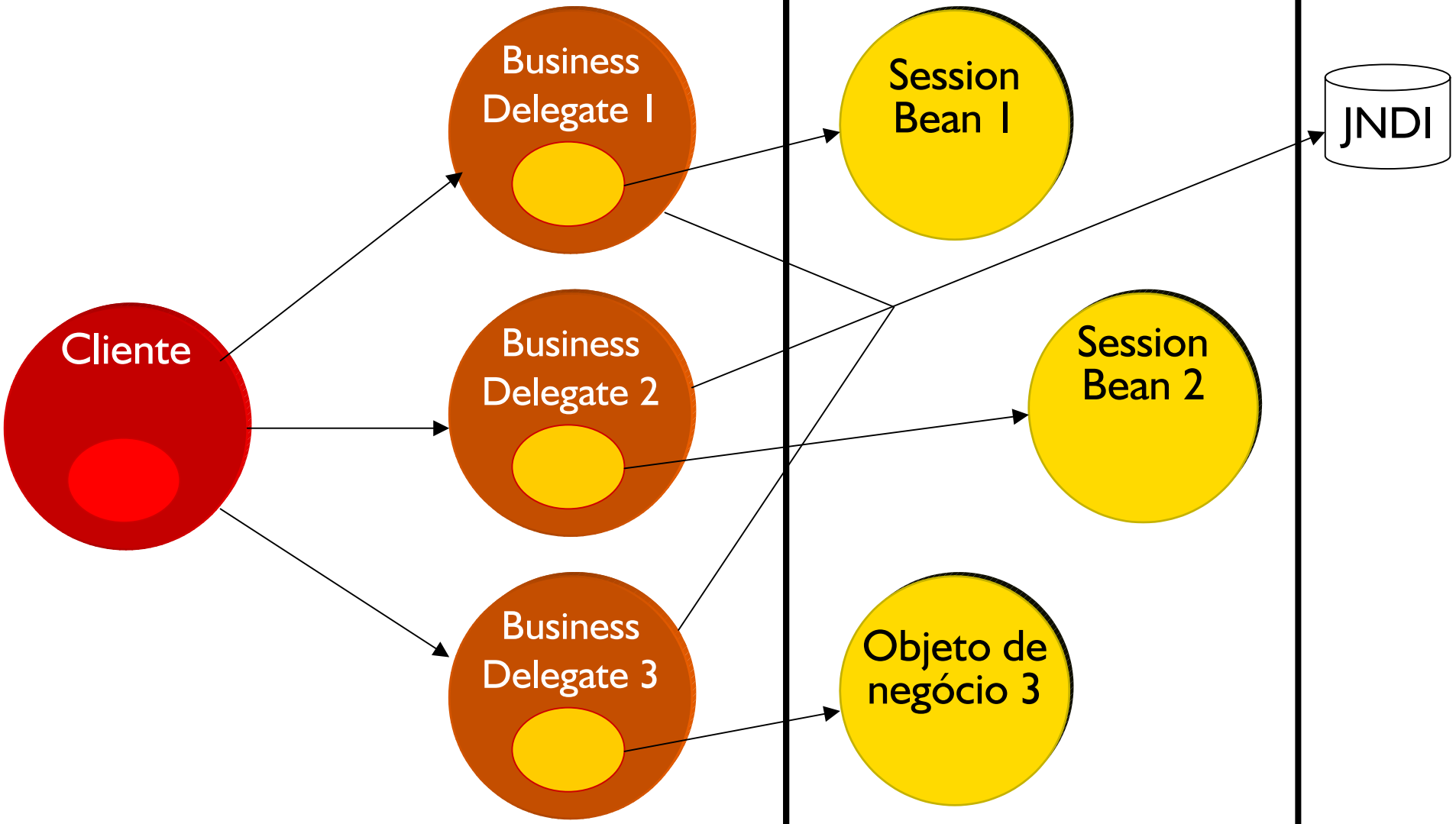
Business Delegate

Objetivo: isolar cliente de detalhes acerca da camada de negócios. Business delegates funcionam como proxies ou fachadas para cada session bean.

Problema


EJB 1
JNDI
Cliente
EJB 2
Objeto de negócio 3
Acoplamento forte: cliente contém código de acesso e referências diretas para session beans
Camada do Cliente ou de Apresentação
Camada de Negócios

Solução: Business Delegate



Melhores estratégias de implementação

- Delegate Proxy Strategy
  - Interface com mesmos métodos que o objeto de negócio que está intermediando
  - Pode realizar cache e outros controles
- Delegate Adapter Strategy
  - Permite integração de um sistema com outro (sistemas podem usar XML como linguagem de integração)

Conseqüências

- Reduz acoplamento
- Traduz exceções de serviço de negócio
- Implementa recuperação de falhas
- Expõe interface mais simples
- Pode melhorar a performance com caches
- Introduz camada adicional
- Transparência de localidade
  - Oculta o fato dos objetos estarem remotos

Exercícios

- *10. Refatore a aplicação em cap13/bd/ para que utilize Business Delegate:*
  - *a) Implemente um Business Delegate para fazer interface com entre o Controller Servlet (ou comandos) e o DAO.*
  - *b) Trate as exceções específicas de cada camada e encapsule-as em exceções comuns a todo o sistema*

helder@acm.org

argonavis.com.br

14 Testes em Aplicações Web com Cactus

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Sobre este módulo

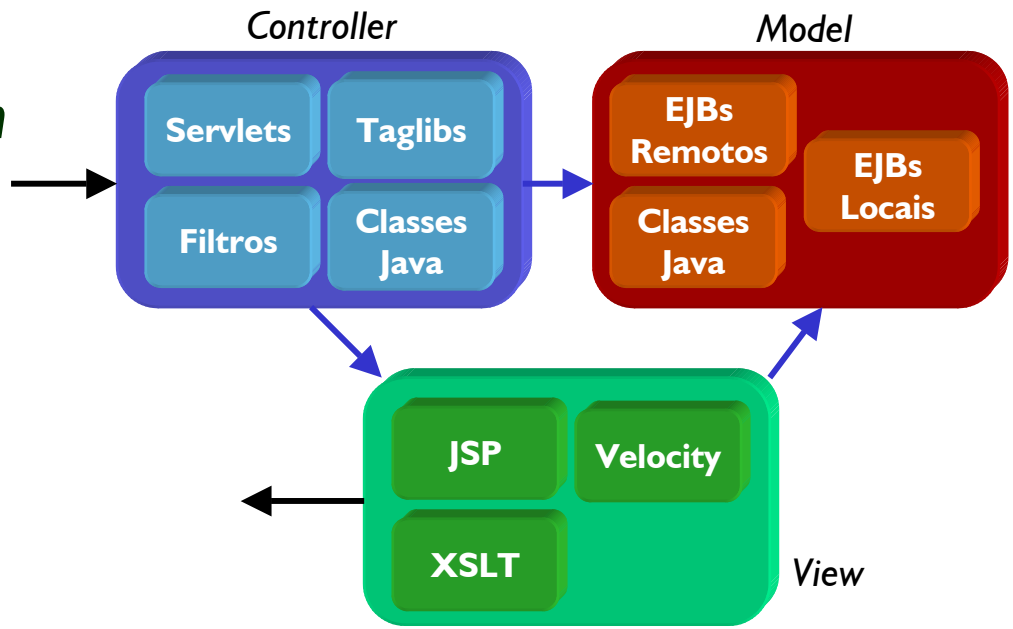
- Este módulo descreve um framework - o Jakarta Cactus - que pode ser utilizado para testar aplicações Web em Java
  - É uma extensão do JUnit
  - Testa os componentes dentro do container
- O assunto é extenso e implementar os primeiros testes pode ser trabalhoso, portanto, o assunto será apresentado superficialmente através de demonstrações
  - Execute as demonstrações usando o build.xml
  - Leia o README.txt para instruções mais detalhadas

O que é Cactus?

- É um framework que oferece facilidades para testar componentes J2EE
  - Componentes Web (Camada de Controle)
  - Camada EJB (Model) e cliente (View): indiretamente
- Produto Open Source do projeto Jakarta
  - Metas de curto prazo: testar componentes acima + EJB
  - Metas de longo prazo: oferecer facilidades para testar todos os componentes J2EE; ser o framework de referência para testes in-container.
- Cactus estende o JUnit framework
  - Execução dos testes é realizada de forma idêntica
  - TestCases são construídos sobre uma subclasse de junit.framework.TestCase

Para que serve?

- Para testar aplicações que utilizam componentes J2EE
- Arquitetura MVC
  - Servlets, filtros e custom tags (Controladores)
  - JSPs (camada de apresentação: View, através de controladores)
  - EJB (Modelo de dados/ lógica de negócios)



- Cactus testa a integração desses componentes com seus containers
  - não usa stubs - usa o próprio container como servidor e usa JUnit como cliente
  - comunicação é intermediada por um proxy

Ilustração: Manual do Cactus

Como funciona?

- Cactus utiliza os test cases simultaneamente no cliente e no servidor: duas cópias
  - Uma cópia é instanciada pelo servlet container
  - Outra cópia é instanciada pelo JUnit
- Comunicação com o servlet container é feita através de um proxy (XXXRedirector)
  - JUnit envia requisições via HTTP para proxy
  - Proxy devolve resultado via HTTP e JUnit os mostra
- Há, atualmente (Cactus 1.3) três tipos de proxies:
  - ServletRedirector: para testar servlets
  - JSPRedirector: para testar JSP custom tags
  - FilterRedirector: para testar filtros de servlets

Arquitetura

- *Parte da mesma classe (ServletTestCase) é executada no cliente, parte no servidor*

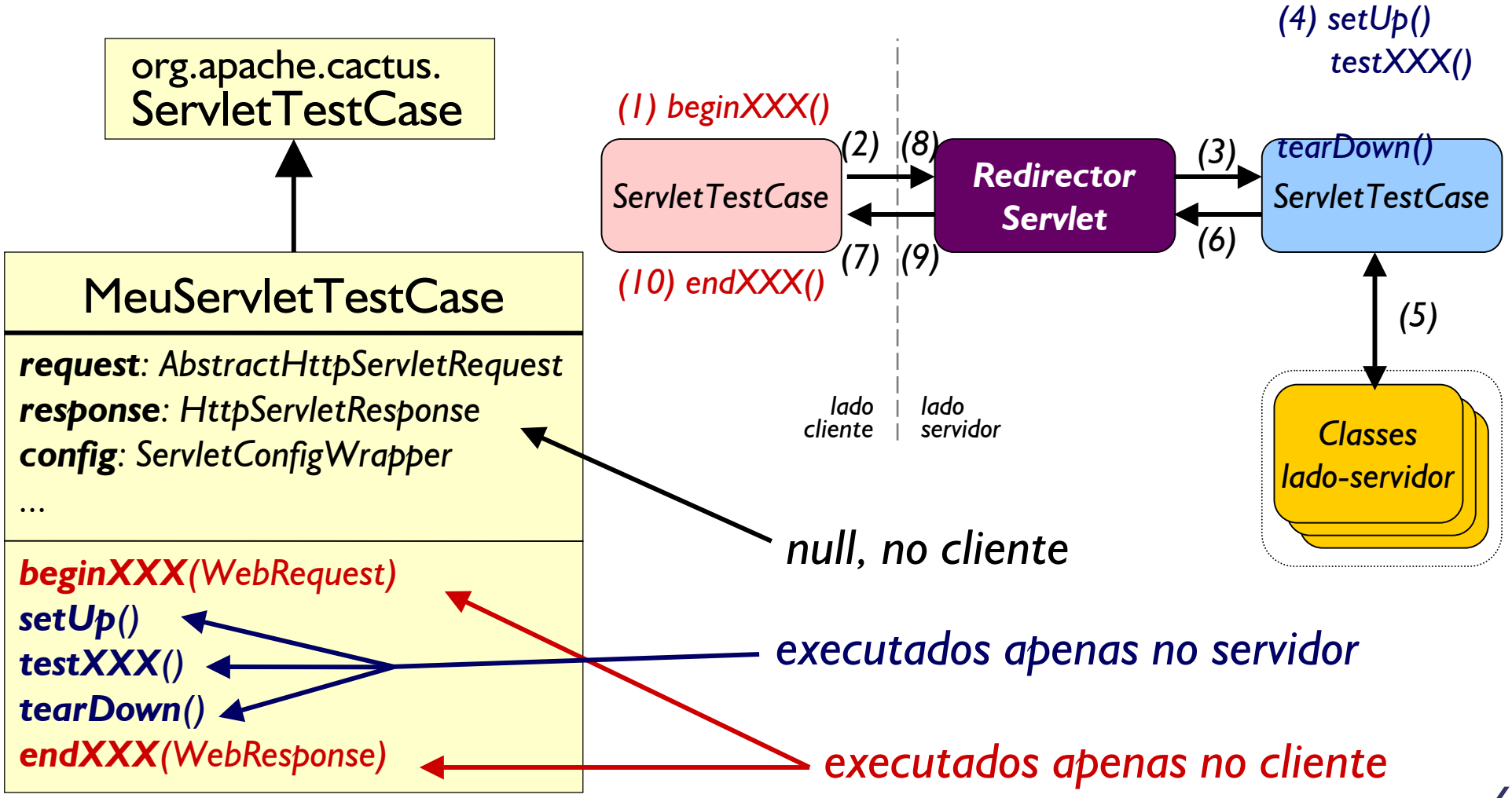
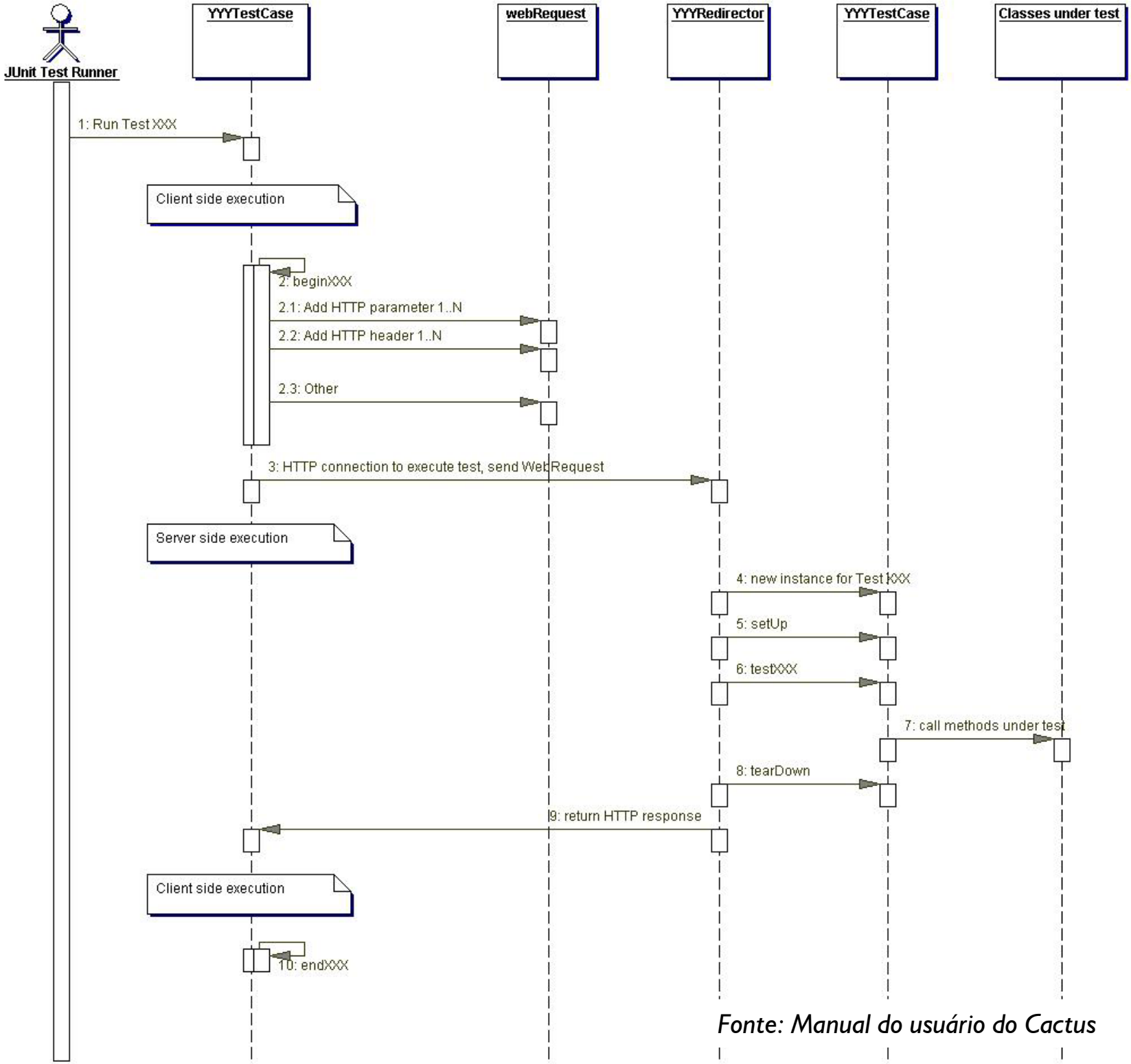


Diagrama UML



Fonte: Manual do usuário do Cactus

ServletTestCase (ou similar)

- Para cada método XXX() a ser testado, pode haver:
  - Um `beginXXX()`, para inicializar a requisição do cliente
    - encapsulada em um objeto WebRequest a ser enviado ao servidor
  - Um `testXXX()`, para testar o funcionamento do método no servidor (deve haver ao menos um)
  - Um `endXXX()`, para verificar a resposta do servidor
    - devolvida em um objeto WebResponse retornada pelo servidor
- Além desses três métodos, cada TestCase pode conter
  - `setUp()`, opcional, para inicializar objetos no servidor
  - `tearDown()`, opcional, para liberar recursos no servidor
- Os métodos do lado do servidor têm acesso aos mesmos objetos implícitos disponíveis em um servlet ou página JSP: request, response, etc.

Cactus: exemplo

- Veja cactusdemo.zip (distribuído com esta palestra)
  - Usa duas classes: um servlet (MapperServlet) e uma classe (SessionMapper) que guarda cada parâmetro como atributo da sessão e em um HashMap - veja fontes em src/xptoolkit/cactus
- Para rodar, configure o seu ambiente:
  - build.properties - localização dos JARs usados pelo servidor Web (CLASSPATH do servidor)
  - runtests.bat (para Windows) e runtests.sh (para Unix) - localização dos JARs usados pelo JUnit (CLASSPATH do cliente)
  - lib/client.properties (se desejar rodar cliente e servidor em máquinas separadas, troque as ocorrências de localhost pelo nome do servidor)
- Para montar, execute:
  - 1. `ant test-deploy` instala cactus-tests.war no tomcat
  - 2. o servidor (Tomcat 4.0 startup)
  - 3. `runtests.bat` roda os testes no JUnit

veja demonstração

cactusdemo

CactusDemo: servlet

- *O objetivo deste servlet é*
  - *1) gravar qualquer parâmetro que receber na sessão (objeto session)*
  - *2) devolver uma página contendo os pares nome/valor em uma tabela*
  - *3) imprimir resposta em caixa-alta se <init-param> ALL_CAPS definido no web.xml contiver o valor true*

```
public void doGet(...) throws IOException {
     SessionMapper.mapRequestToSession(request);
     writer.println("<html><body><table border='1'>");
    // (... loop for each parameter ...)
        if (useAllCaps()) {
          key = key.toUpperCase();
          val = val.toUpperCase();
        }
        str = "<tr><td><b>"+key+"</b></td><td>"+val+"</td></tr>";
        writer.println(str);
    // (...)
     writer.println("</table></body></html>");
}
```

(1) *Grava request em session*

(3) *Retorna* true *se <init-param> "ALL_CAPS" contiver "true"*

(2)

Trecho de MapperServlet.java

- *Escreveremos os testes para avaliar esses objetivos*

Fonte do exemplo: [1]

CactusDemo: testes

MapperServletTest.java

```
public class MapperServletTest extends ServletTestCase { (...)
  private MapperServlet servlet;
  public void beginDoGet(WebRequest cSideReq) {
     cSideReq.addParameter("user", "Jabberwock");
  }
  public void setUp() throws ServletException {
    this.config.setInitParameter("ALL_CAPS", "true");
    servlet = new MapperServlet();
    servlet.init(this.config);
  }
  public void testDoGet() throws IOException {
    servlet.doGet(this.request, this.response);
    String value = (String) session.getAttribute("user");
    assertEquals("Jabberwock", value);
  }
  public void tearDown() { /* ... */ }
  public void endDoGet(WebResponse cSideResponse) {
    String str = cSideResponse.getText();
    assertTrue(str.indexOf("USER</b></td><td>JABBERWOCK") > -1);
  }
}
```

Simula DD <init-param>

Simula servlet container

Verifica se parâmetro foi mapeado à sessão

Verifica se parâmetro aparece na tabela HTML

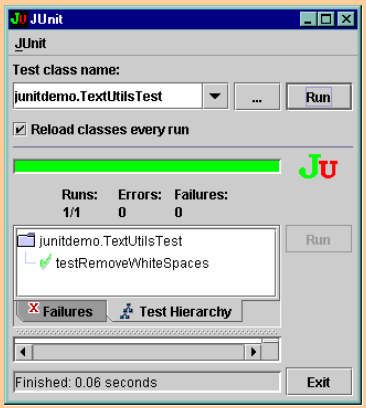
Exemplo: funcionamento

Cliente (JUnit)

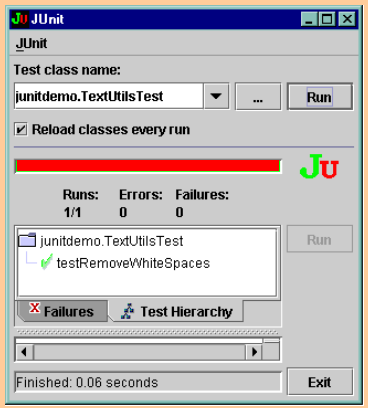
`beginDoGet(WebRequest req)`

- Grava parâmetro:
nome=`user`
value=`Jabberwock`

SUCCESS!!



FAIL!



falha local

falha remota

&

`endDoGet(WebResponse res)`

- Verifica se resposta contém
`USER</b></td><td>JABBERWOCK`

ReqInfo

2 conexões HTTP:

- *Uma p/ rodar os testes e obter saida do servlet*
- *Outra para esperar resultados de testes (info sobre exceções)*

TestInfo

Output

Servidor (Tomcat)

`setUp()`
- *Define init-params no objeto* `config`
- *Roda* `init(config)`

`testDoGet()`
- *Roda* `doGet()`
- *Verifica se parâmetro (no response) foi mapeado à sessão*

`tearDown()`

HttpUnit

- Onde encontrar
  - `http://httpunit.sourceforge.net`
- Framework para testes funcionais de interface (teste tipo "caixa-preta")
  - Verifica a resposta de uma aplicação Web ou página HTML
  - É teste funcional caixa-preta (não é "unit")
  - Oferece métodos para "navegar" na resposta
    - links, tabelas, imagens
    - objetos DOM (Node, Element, Attribute)
- Pode ser combinado com Cactus no endXXX()
  - Argumento `com.meterware.httpunit.WebResponse`
- Acompanha ServletUnit
  - stub que simula o servlet container

Resumo da API do HttpUnit

- *WebConversation*
  - *Representa uma sessão de cliente Web (usa cookies)*

```
WebConversation wc = new WebConversation();
WebResponse resp = wc.getResponse("http://xyz.com/t.html");
```

- *WebRequest*
  - *Representa uma requisição*
- *WebResponse*
  - *Representa uma resposta. A partir deste objeto pode-se obter objetos WebLink, WebTable e WebForm*
- *WebLink*
  - *Possui métodos para extrair dados de links de hipertexto*
- *WebTable*
  - *Possui métodos para navegar na estrutura de tabelas*
- *WebForm*
  - *Possui métodos para analisar a estrutura de formulários*

HttpUnit com Cactus

Veja MapperServletTest2.java

- *Troque o* **WebResponse** *em cada endXXX() por* **com.meterware.httpunit.WebResponse**

```
public void endDoGet(com.meterware.httpunit.WebResponse resp)
                                    throws org.xml.sax.SAXException {
    WebTable[] tables = resp.getTables();
    assertNotNull(tables);
    assertEquals(tables.length, 1);  // só há uma tabela
    WebTable table = tables[0];
    int rows = table.getRowCount();
    boolean keyDefined = false;
    for (int i = 0; i < rows; i++) {
        String key   = table.getCellAsText(i, 0); // col 1
        String value = table.getCellAsText(i, 1); // col 2
        if (key.equals("USER")) {
            keyDefined = true;
            assertEquals("JABBERWOCK", value);
        }
    }
    if (!keyDefined) {
        fail("No key named USER was found!");
    }
}
```

Outros testes com Cactus

- Testes em taglibs (JspRedirector)
  - Veja exemplos em cactusdemo/taglib/src
- Testes em filtros (FilterRedirector)
  - Usa proxy FilterRedirector
  - Teste básico é verificar se método doFilter() foi chamado
  - Veja exemplos em cactusdemo/src/xptoolkit/AuthFilter
- Testes indiretos em páginas JSP (camada **V**iew)
  - Ideal é JSP não ter código Java
  - Principais testes são sobre a interface: HttpUnit!
- Testes indiretos em EJB (camada **M**odel)
  - Indireto, através dos redirectors + JUnitEE
  - Redirectors permitem testar EJBs com interface local ou remota chamados por código no servidor

veja

hellojsp_2

Testes em aplicações Web: conclusões

- Aplicações Web são difíceis de testar porque dependem da comunicação com servlet containers
  - Stubs, proxies e APIs, que estendem ou cooperam com o JUnit, tornam o trabalho mais fácil
  - Neste bloco, conhecemos três soluções que facilitam testes de unidade, de integração e de caixa-preta em aplicações Web
- Stubs como ServletUnit permitem testar as unidades de código mesmo que um servidor não esteja presente
- Proxies como os "redirectors" do Cactus permitem testar a integração da aplicação com o container
- Uma API, como a fornecida pelo HttpUnit ajuda a testar o funcionamento da aplicação do ponto de vista do usuário

Exercícios

- *1. Escreva testes JUnit (não Cactus) para a MessageBeanDAO.*
- *2. Escreva testes Cactus para testar um dos servlets que você desenvolveu neste curso (comece com uma simples)*

helder@acm.org

argonavis.com.br

J550

Apache Struts

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Sobre este módulo

- Este é um módulo opcional.
- Apresenta uma visão geral do framework Struts, usado para desenvolver aplicações Web com o padrão MVC
  - Não é exaustivo. Consulte a documentação suplementar no CD para mais informações
- A apresentação será feita através de um exemplo simples, mas que utiliza diversos recursos comuns em aplicações típicas
  - Utilize-o como base para aplicações maiores

O que é Struts?

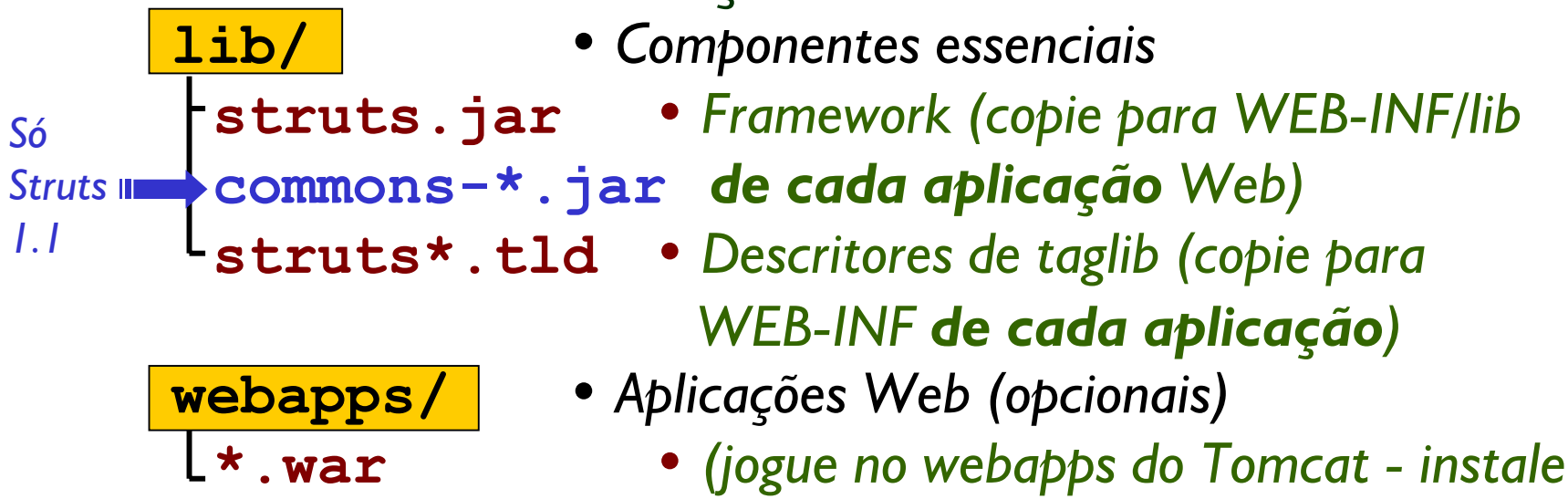
- Framework para facilitar a implementação da arquitetura MVC em aplicações JSP
- Oferece
  - Um servlet controlador configurável (Front Controller) através de documentos XML externos, que despacham requisições a classes Action (Command) criadas pelo desenvolvedor
  - Uma vasta coleção de bibliotecas de tags JSP (taglibs)
  - Classes ajudantes que oferecem suporte a tratamento de XML, preenchimento de JavaBeans e gerenciamento externo do conteúdo de interfaces do usuário
- Onde obter: jakarta.apache.org/struts

Componentes MVC no Struts

- Model (M)
  - Geralmente um objeto Java (JavaBean)
- View (V)
  - Geralmente uma página HTML ou JSP
- Controller (C)
  - org.apache.struts.action.ActionServlet ou subclasse
- Classes ajudantes
  - FormBeans: encapsula dados de forms HTML (M)
  - ActionErrors: encapsulam dados de erros (M)
  - Custom tags: encapsulam lógica para apresentação (V)
  - Actions: implementam lógica dos comandos (C)
  - ActionForward: encapsulam lógica de redirecionamento (C)

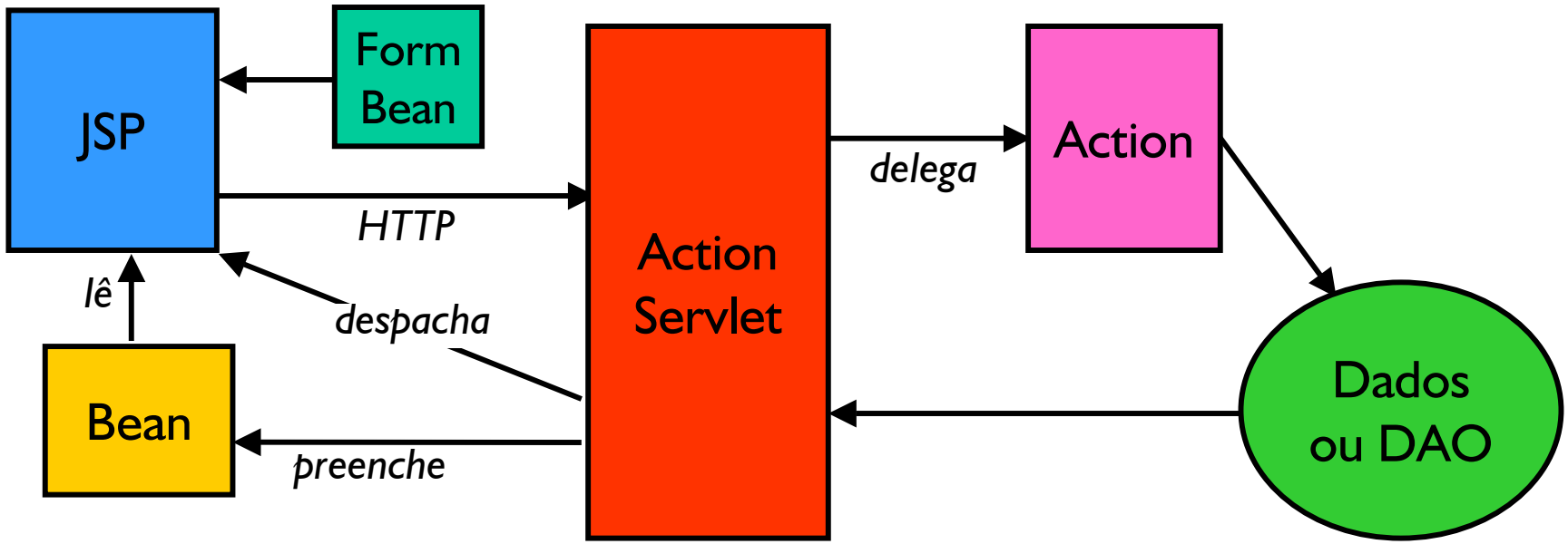
Componentes da distribuição

- Requisitos
  - J2SDK 1.4 ou J2SDK1.3 + JAXP
  - Servlet container, servlet.jar e Jakarta Commons (Struts 1.1)
- Distribuição binária (pré-compilada)
  - Abra o ZIP da distribuição. Conteúdo essencial:



Como funciona?

- Componentes-chave
  - ActionServlet: despachante de ações
  - Action: classe estendida por cada ação (comando) a ser implementada (usa Command design pattern)
  - struts-config.xml: arquivo onde se define mapeamentos entre ações, páginas, beans e dados



Como instalar

- 1. Copiar os arquivos necessários para sua aplicação
  - Copie lib/struts.jar e lib/commons-*.jar para weu WEB-INF/lib (não coloque no common/lib do Tomcat ou no jre/lib/ext do JDK ou o struts não achará suas classes!)
  - Copie os TLDs das bibliotecas de tags que deseja utilizar para o WEB-INF de sua aplicação (copie todos)
- 2. Para usar o servlet controlador (MVC)
  - Defina-o como um <servlet> no seu web.xml
  - Crie um arquivo WEB-INF/struts.config.xml com mapeamentos de ações e outras as configurações
- 3. Para usar cada conjunto de taglibs
  - Defina, no seu web.xml, cada taglib a ser instalada
  - Carregue a taglib em cada página JSP que usá-la

Configuração do controlador no web.xml

- Acrescente no seu web.xml

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>
      org.apache.struts.action.ActionServlet
   </servlet-class>
   <init-param>
      <param-name>config</param-name>
      <param-value>
         /WEB-INF/struts-config.xml
      </param-value>
   </init-param>
   ... outros init-param ...
</servlet>
```

- Acrescente também o <servlet-mapping> desejado
- Crie e configure as opções de struts-config.xml
- Veja nos docs: /userGuide/building_controller.html
  - Use os arquivos de struts-example.war para começar

Configuração das Taglibs

- *Acrescente em **web.xml***
- *Veja detalhes na aplicação struts-example.war ou nos docs:*
  /userGuide/building_controller.html#dd_config_taglib

```
<taglib>
   <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-bean.tld
   </taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-form.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-form.tld
   </taglib-location>
... outros taglibs ...
</taglib>
```

URI é fácil de lembrar e tem o mesmo nome que a localização ideal da TLD

- *Acrescente em cada página JSP*

```
<@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
   ...
```

Como criar uma aplicação com o Struts?

Preparação

- 1. Defina seus comandos (Controller), escolha as URLs para chamá-los e nomes de classes Action que irão executá-los
- 2. Defina as páginas JSP que você irá precisar (Views), incluindo páginas de sucesso e de erro
- 3. Defina as páginas de entrada de dados (formulários)

Com os dados que você obteve na fase de preparação, crie

- 1. Um arquivo struts-config.xml. Use o DTD fornecido pelo Struts ou um esqueleto mínimo. Neste arquivo estão mapeadas todas as ações da sua aplicação: como serão chamadas (URLs de comando), classes Action que irão executá-las, formulários que serão usados e páginas que serão retornadas
- 2. Uma subclasse de Action para cada comando
- 3. Uma subclasse de ActionForm para cada formulário
- 4. Um resource bundle (arquivo .properties), possivelmente vazio, inicialmente, para guardar mensagens de erro.

Anatomia do struts-config.xml

- *Este é o arquivo mais importante da aplicação. Ele encapsula toda a lógica de processamento*

Bean que reflete os campos de seu formulário (criar)

```
<struts-config>
  <form-beans>
    <form-bean name="novaMsg" type="forum.EntradaDados" />
  </form-beans>
  <action-mappings>
    <action path="/nova"
            type="forum.NovaAction"
            validate="true"
            input="/index.jsp" name="novaMsg" scope="request">
      <forward name="sucesso" path="/todas.do" />
      <forward name="default" path="/index.jsp" />
    </action>
    <action path="/todas"
            type="forum.ListarAction" scope="request">
      <forward name="sucesso" path="/todas.jsp" />
      <forward name="erro" path="/index.jsp" />
    </action>
  </action-mappings>
  <message-resources parameter="forum.ApplicationResources" />
</struts-config>
```

Esta ação é iniciada por um formulário associado ao bean novaMsg implementado em index.jsp, e requer validação

Uma ação

Chama ação /todas

Resource-bundle (tem extensão .properties e está no CLASSPATH)

Ações

- *Não é preciso mexer no ActionServlet*
  - *Para grande parte das aplicações, basta escrever as classes Action e defini-las no struts-config.xml*
- *Para implementar um Action, crie uma nova classe e implemente seu método execute()*

Usado para redirecionar a uma View: "sucesso" ou "default" neste exemplo

```
public class NovaAction extends Action {

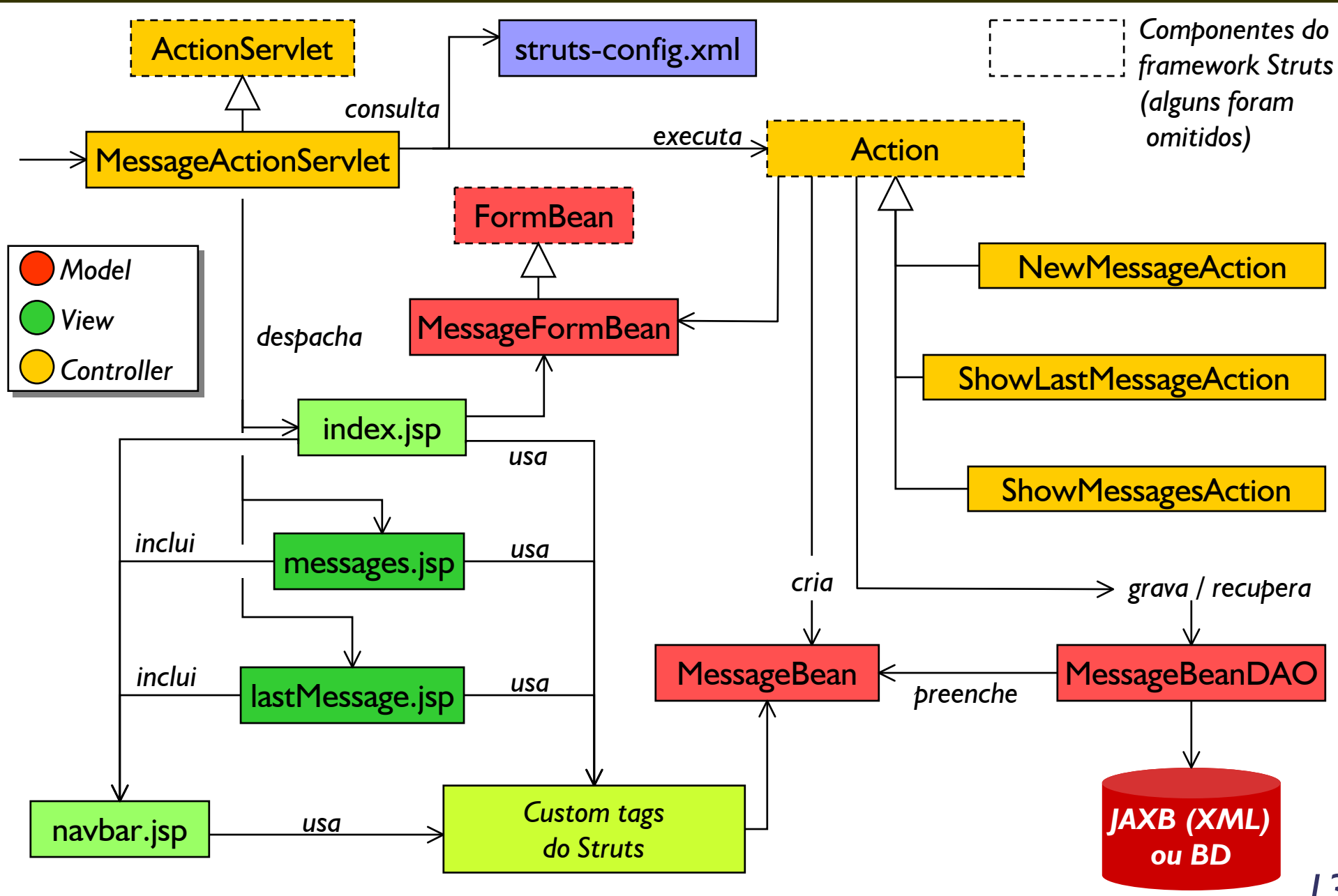
   public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
                        throws IOException, ServletException {
```

Se esta ação precisa de dados digitados no formulário, recupere-os através dos métodos get/set deste bean

```
      EntradaDados entrada = (EntradaDados)form;
      String mensagem = entrada.getMensagem();
        ...
      if (funcionou) return (mapping.findForward("sucesso"));
      else return (mapping.findForward("default"));
   }
```

Veja struts-config.xml

Implementação de hellojsp com Struts



Mapeamentos (ActionMappings)

- *Veja webinf/struts-config.xml*

```
<struts-config>
  <form-beans>
    <form-bean name="newMessageForm" type="hello.jsp.NewMessageForm" />
  </form-beans>
  <global-forwards>
    <forward name="default" path="/index.jsp" />
  </global-forwards>

  <action-mappings>
    <action path="/newMessage" type="hello.jsp.NewMessageAction"
            validate="true"
            input="/index.jsp" name="newMessageForm" scope="request">
      <forward name="success" path="/showLastMessage.do" />
    </action>
    <action path="/showLastMessage"
            type="hello.jsp.ShowLastMessageAction" scope="request">
      <forward name="success" path="/lastMessage.jsp" />
    </action>
    <action path="/showAllMessages"
            type="hello.jsp.ShowMessagesAction" scope="request">
      <forward name="success" path="/messages.jsp" />
    </action>
  </action-mappings>

  <message-resources parameter="hello.jsp.ApplicationResources" />
</struts-config>
```

FormBeans

- *Form beans permitem simplificar a leitura e validação de dados de formulários*
  - *Devem ser usados em conjunto com custom tags da biblioteca <html:* />*

```
<html:form action="/newMessage" name="newMessageForm"
           type="hello.jsp.NewMessageForm">
    <p>Message: <html:text property="message" />
       <html:submit>Submit</html:submit>
    </p>
</html:form>
```

Configuração em struts-config.xml

```
public class NewMessageForm extends ActionForm {
    private String message = null;
    public String getMessage() { return message; }
    public void setMessage(String message) {
        this.message = message;
    }
    public void reset(...) {
        message = null;
    }
    public ActionErrors validate(...) {...}
}
```

ActionErrors

- *ActionErrors encapsulam erros de operação, validação, exceções, etc.*
  - *Facilitam a formatação e reuso de mensagens de erro.*
- *Exemplo: Método validate() do form bean:*

```
public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  if ( (message == null) || (message.trim().length() == 0) ) {
    errors.add("message",
               new ActionError("empty.message.error"));
  }
  return errors;
}
```

Nome de campo <input> ao qual o erro se aplica.

Este valor corresponde a uma chave no ResourceBundle

- *Como imprimir:*

```
<html:errors />
```

i18n

- *Informações localizadas podem ser facilmente extraídas de Resource Bundles através de*

```
<bean:message key="chave" />
```

  - *Locale default é usado automaticamente (pode ser reconfigurado)*
- *Exemplo de ResourceBundle*

```
empty.message.error=<tr><td>Mensagem não pode ser
vazia ou conter apenas espaços em branco.</td></tr>
new.message.input.text=Digite a sua mensagem
message.submit.button=Enviar Mensagem
```

hello/jsp/ApplicationResources_pt.properties

- *Configuração em struts-config.xml*

```
<message-resources
      parameter="hello.jsp.ApplicationResources" />
```

- *Exemplo de uso:*

```
<p><bean:message key="new.message.input.text" />
```

Action (Controller / Service To Worker)

- *Controlador processa comandos chamando o método execute de um objeto Action*

```
public class ShowMessagesAction extends Action {

  private String successTarget = "success";
  private String failureTarget = "default";

  public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
                          throws IOException, ServletException {
    try {
       MessageBeanDAO dao =
          (MessageBeanDAO)request.getAttribute("dao");
       MessageBean[] beanArray = dao.retrieveAll();
       request.setAttribute("messages", beanArray);
       return (mapping.findForward(successTarget));
     } catch (PersistenceException e) {
       throw new ServletException(e);
     }
  } ...
```

Como rodar o exemplo

- 1. Mude para cap15/exemplos/hellojsp_3
- 2. Configure build.properties, depois rode

```
> ant DEPLOY
```

- 3. Inicie o servidor (Tomcat ou JBoss)
- 4. Rode os testes do Cactus se desejar

```
> ant RUN-TESTS
```

- 5. Rode a aplicação, acessando a URI

```
http://localhost:porta/hellojsp-struts/
```

- 6. Digite mensagens e veja resultados. Arquivos são gerados em /tmp/mensagens (ou c:\tmp\mensagens)

Exercícios

- 1. Coloque o exemplo para funcionar
  - Analise o código das páginas JSP e classes Action
  - Veja o arquivo struts-config.xml
- 2. Adapte o segundo exercício do capítulo 2 para funcionar com o Struts

helder@acm.org

argonavis.com.br

16 Como criar Custom Tags

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Objetivos

- *Este é um módulo opcional. Explora os fundamentos para criação de custom tags*
  - *Funcionamento: como fazer um custom tag*
  - *Exemplos de tags simples*
- *Não exploramos aspectos mais avançados*
  - *A criação de uma biblioteca de custom tags complexa não é uma tarefa comum para a maior parte dos desenvolvedores. É mais comum reutilizar tags existentes e criar alguns mais simples*
  - *Veja exemplos de tags mais elaborados no diretório do CD correspondente a este capítulo*

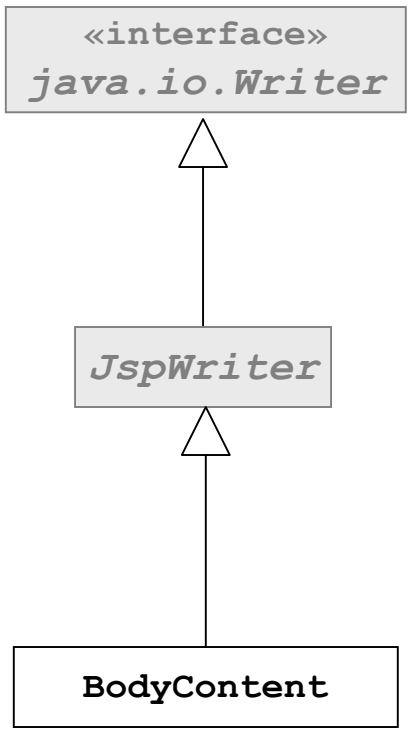
Como criar um custom Tag

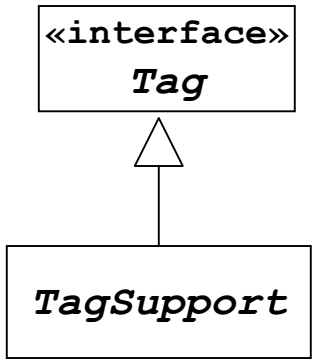
- Para criar um custom tag é preciso programar usando as APIs
  - javax.servlet.jsp e
  - javax.servlet.jsp.tagext
- Resumo dos passos típicos
  - Escreva uma classe que implemente a interface Tag ou BodyTag (ou as adaptadores TagSupport e BodyTagSupport)
  - Implemente os métodos do ciclo de vida desejados
  - Escreva um descritor TLD ou acrescente os dados de seu tag a um TLD existente
  - Empacote tudo em um JAR
  - Inclua o JAR em um WAR e use os tags em suas páginas

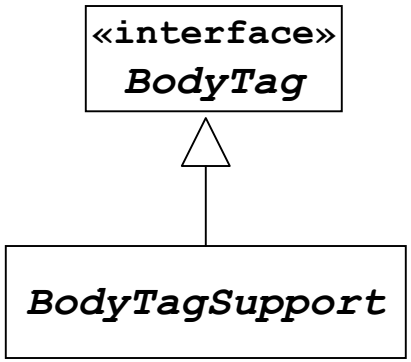
API

Pacote *javax.servlet.jsp.tagext*: principais componentes

- Interfaces
  - Tag
  - BodyTag
  - IterationTag
- Classes
  - TagSupport
  - BodyTagSupport
  - BodyContent







Exemplo (tag usado em capítulo anterior)

- No capítulo 11 mostramos como usar um custom tag muito simples
- `<exemplo:dataHoje />`

  que tinha como resultado

Tuesday, May 5, 2002 13:13:13 GMT-03

- Para construir este tag é preciso
  - 1. Escrever a classe que o implementa
  - 2. Declará-lo em seu arquivo TLD
- Para distribuí-lo, encapsule tudo em um JAR que o usuário do tag possa colocar em seu WEB-INF/lib

Configuração do Tag

```
<?xml version="1.0" ?>
<!DOCTYPE taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>exemplo</short-name>
    <uri>http://abc.com/ex</uri>

    <tag>
        <name>dataHoje</name>
        <tag-class>exemplos.DateTag</tag-class>
        <description>Data de hoje</description>
    </tag>

</taglib>
```

Sugestão de prefixo (autor de página pode escolher outro na hora)

URI identifica o prefixo. (autor de página tem que usar exatamente esta URI)

Implementação

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class DateTag extends TagSupport {
    /**
     * Chamado quando o tag terminar.
     */
    public int doEndTag() throws JspException {
        try {
            Writer out = pageContext.getOut();
            java.util.Date = new java.util.Date();
            out.println(hoje.toString());
        } catch (java.io.IOException e) {
            throw new JspException (e);
        }
        return Tag.EVAL_PAGE;
    }
}
```

Para tags que não precisam processar o corpo use a interface Tag ou sua implementação TagSupport

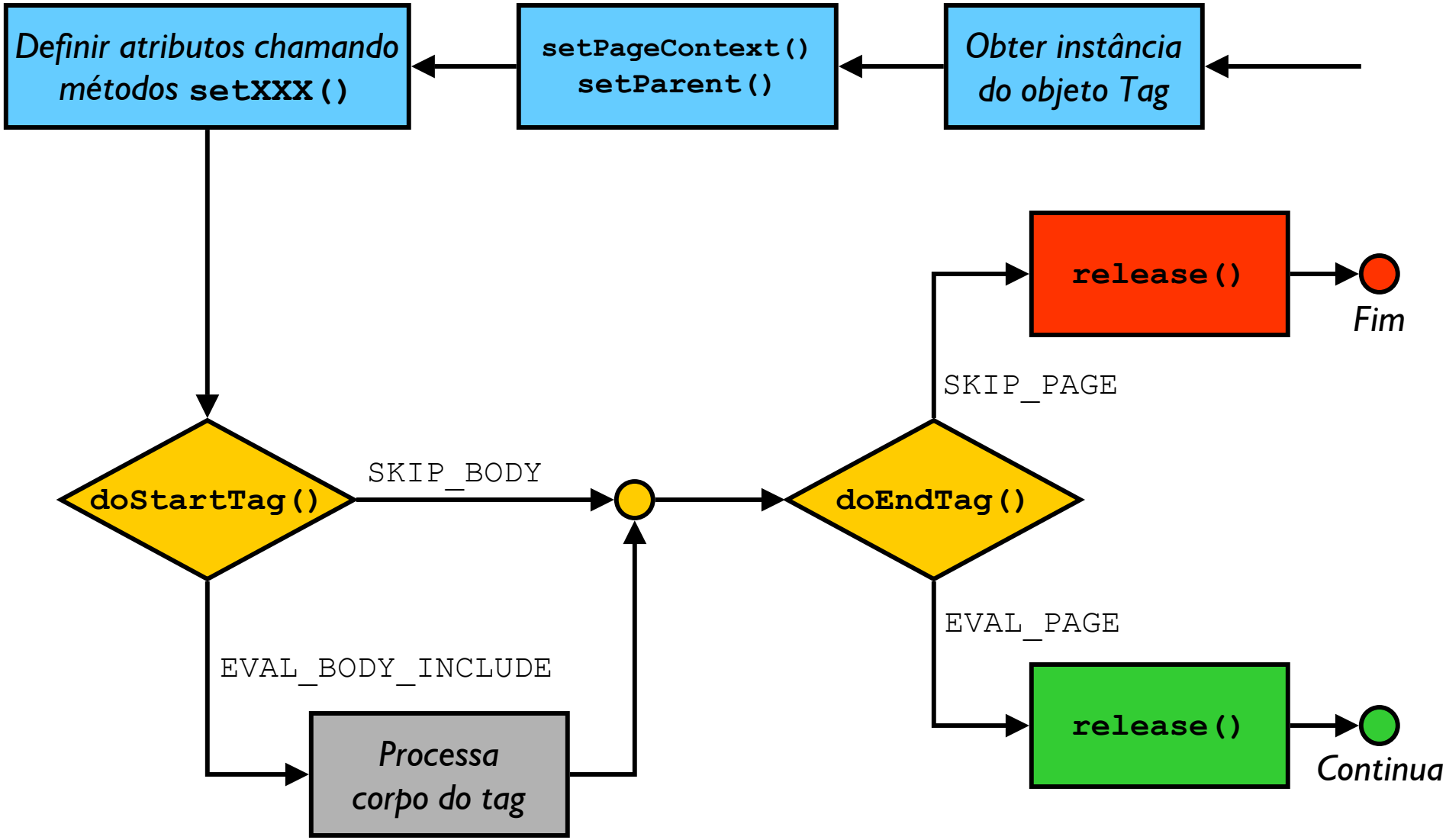
Use doStartTag() para processamento antes do tag, se necessário

Para este método, pode ser EVAL_PAGE ou SKIP_PAGE

Tipos de tags

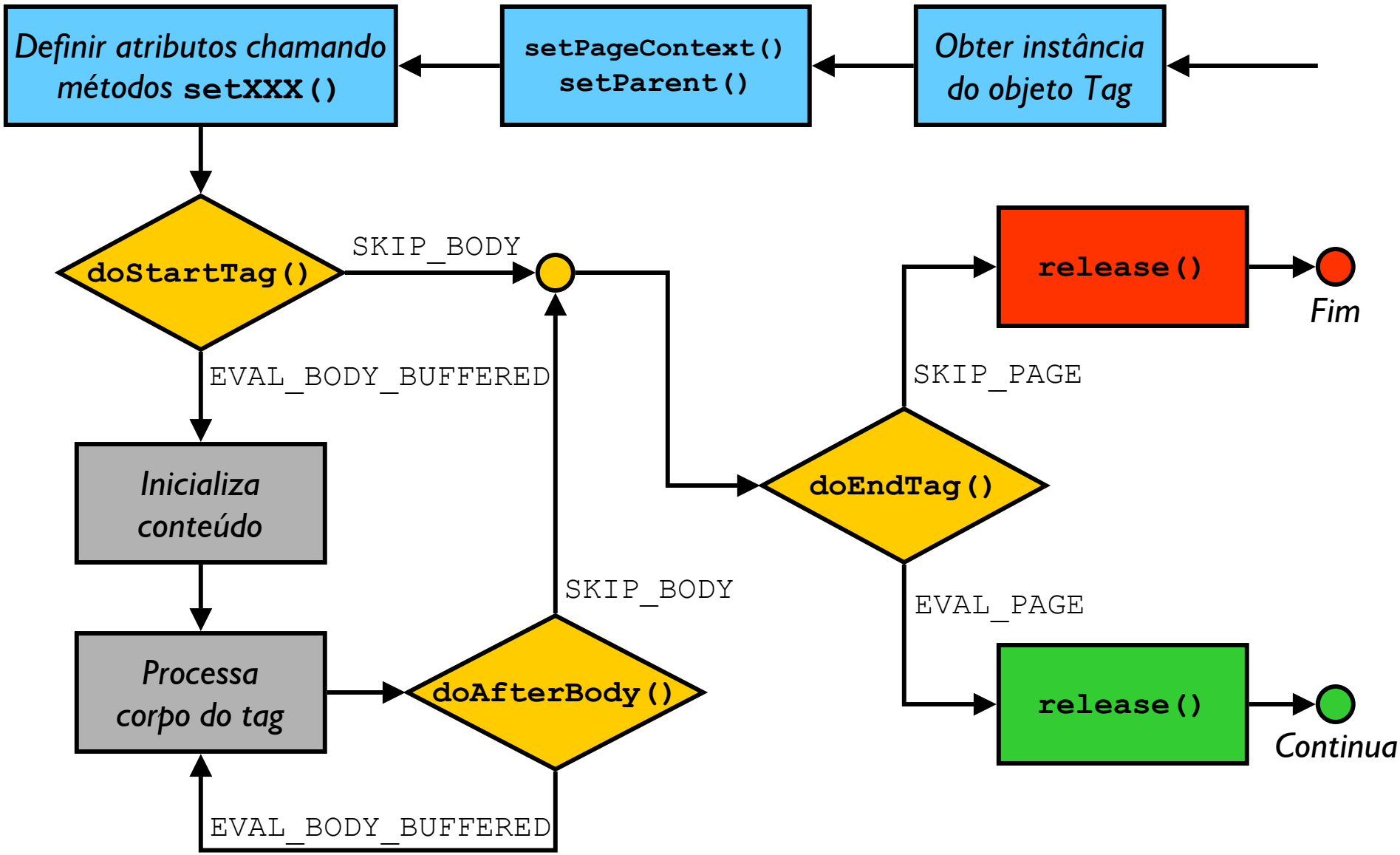
- *Há vários tipos de custom tags. Cada estratégia utiliza diferentes classes base e métodos*
- *1. Diferenciados por herança:*
  - *Tags simples: implementam a interface Tag (TagSupport é uma implementação neutra).*
  - *Tags com corpo que requer processamento: implementam BodyTag (BodyTagSupport é implementação neutra)*
- *Diferenciados por outras características*
  - *Tags que possuem atributos*
  - *Tags que definem variáveis de scripting fora do seu escopo (requerem classe extra com "Tag Extra Info")*
  - *Tags que interagem com outros tags*

Ciclo de vida de objetos Tag



Fonte: [6]

Ciclo de vida de objetos BodyTag



Fonte: [6]

Atributos

- *Para definir atributos em um tag é preciso*
  1. *Definir método setXXX() com o nome do atributo*
  2. *Declarar atributo no descritor (TLD)*

```
<xyz:upperCase text="abcd" />
```

```
public class UpperCaseTag {
    public String text;
    public void setText(String text) {
        this.text = text;
    } (...)
```

```
<tag> <name>upperCase</name> (...)
    <attribute>
        <name>text</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
    </attribute> (...) </tag>
```

- *Os atributos devem setar campos de dados no tag*
  - *Valores são manipulados dentro dos métodos doXXX():*

```
public int doStartTag() throws JspException {
    Writer out = pageContext.getOut();
    out.println(text.toUpperCase());   (...)
```

Obtenção do conteúdo do Body

- *O objeto out, do JSP, referencia a instância BodyContent de um tag enquanto processa o corpo*
  - *BodyContent é subclasse de JspWriter*
  - *Tag decide se objeto BodyContent deve ser jogado fora ou impresso (na forma atual ou modificada)*
- *Exemplo*

```
public int doAfterBody() throws JspException {
    BodyContent body = getBodyContent();
    String conteudo = body.getString();
    body.clearBody();
    (...)
    getPreviousOut().print(novoTexto);
}
```

Guarda conteúdo do tag

Apaga conteúdo

Imprime texto na página (e não no body do Tag)

Exemplos de Custom Tags

- *Veja* **`cap16/exemplos/taglibs/`**
  - *Vários diferentes exemplos de custom tags (do livro [6])*
  - *Código fonte em taglib/src/taglibdemo/*.java*
  - *Páginas exemplo em src/*Test.jsp (6 exemplos)*

1. Configure build.properties, depois, monte o WAR com:

```
> ant build
```

2. Copie o WAR para o diretório webapps do Tomcat

```
> ant deploy
```

3. Execute os tags, acessando as páginas via browser:

```
http://localhost:porta/mut/
```

- *Veja também* **`cap12/exemplos/hellojsp_2/`**
  - *Aplicação MVC que usa custom tags*

Exercício

- *1. Escreva um custom tag simples <j550:tabela> que receba um String como parâmetro (texto) e imprima o String dentro de uma tabela HTML:*
  - *O tag*

```
<j550:tabela texto="Texto Recebido"/>
```

deve produzir

```
<table border="1"><tr><td>
Texto Recebido</td></tr></table>
```

- *2. Crie mais dois atributos que definam cor-de-fundo e cor-do-texto para a tabela e o texto.*
- *3. Escreva uma segunda versão do tag acima que aceite o texto entre os tags*

```
<j550:tabela>Texto Recebido</j550:tabela>
```

helder@acm.org

argonavis.com.br

J2EE e Datasources

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Recursos em servidores J2EE

- Servlets rodando em servidores compatíveis J2EE podem acessar recursos através de JNDI (dominio java:comp/env)
  - Variáveis (environment entries)
  - Referências para componentes EJB
  - Referências para fábricas de recursos (conexões de banco de dados, URLs, serviço de e-mail, JMS, conectores EIS via JCA)
  - Serviços
- Para usar esses recursos
  - Servlet deve estar empacotado em um WAR
  - Nome das variáveis e referências devem ser declarados no web.xml
  - Servlet deve usar como contexto inicial o domínio java:comp/env
- Elementos (filhos de <web-app>) usados no web.xml
  - <env-entry>
  - <ejb-ref>
  - <resource-ref>

Environment Entries

- *Alternativa global (para o WAR) aos <init-param>*
  - *São acessíveis dentro de qualquer servlet ou JSP da aplicação WAR*
  - *Não são visíveis por outras aplicações do servidor (não é um nome JNDI global - está abaixo de java:comp/env - é local à aplicação)*
  - *Acessíveis via ferramentas de deployment (podem ser redefinidas)*
- *Exemplo de uso dentro do <web-app>*

```
<env-entry>
  <env-entry-name>cores/fundo</env-entry-name>
  <env-entry-value>rgb(255, 255, 200)</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

  - *Tipos de dados legais são String e wrappers (Double, Integer, etc.)*
- *Uso dentro do servlet*

```
Context initCtx = new InitialContext();
String fgColor = (String)
                 initCtx.lookup("java:comp/env/cores/fundo");
```

Componentes EJB

- *Servlets e JSPs podem se comunicar com EJBs da aplicação declarando uma referência associada ao bean chamado*
  - *A referência deve informar o tipo do bean (Session, Entity ou MessageDriven e suas interfaces remota e home.*

```
<ejb-ref>
   <description>Cruise ship cabin</description>
   <ejb-ref-name>ejb/CabinHome</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <home>com.titan.cabin.CabinHome</home>
   <remote>com.titan.cabin.Cabin</remote>
</ejb-ref>
```

- *Componentes EJB são retornados como objetos CORBA que precisam ser reduzidos através da função narrow.*

```
InitialContext initCtx = new InitialContext();
Object ref = initCtx.lookup("java:comp/env/ejb/CabinHome");
CabinHome home = (CabinHome)
          PortableRemoteObject.narrow(ref, CabinHome.class);
```

Conexões de banco de dados

- *Fábricas de objetos são acessíveis via <resource-ref>. A mais comum é a fábrica de conexões de banco de dados*

```
<resource-ref>
  <description>Cloudscape database</description>
  <res-ref-name>jdbc/BankDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>SERVLET</res-auth>
</resource-ref>
```

  - *<res-auth> informa quem é responsável pela autenticação*
- *Através da DataSource, obtém-se uma conexão.*

```
InitialContext initCtx = new InitialContext();
DataSource ds = (DataSource)
                initCtx.lookup("java:comp/env/jdbc/BankDB");
Connection con1 = ds.getConnection();// res-auth: CONTAINER
Connection con2 =
   source.getConnection("user","pass"); // res-auth: SERVLET
```

Exercício

- 1. Altere o exercício onde usamos banco de dados (DAO) e faça com que ele use um DataSource em vez da conexão JDBC usual
  - Se possível, use o mesmo banco (se não, crie uma base de dados e a tabela no novo banco)
  - Não precisa mexer no SQL.
  - Mude apenas a chamada para obter a conexão: obtenha a conexão com Connection con = ds.getConnection() em cada método e libere-a no final com con.close();
- 2. Grave algumas variáveis como no ambiente do servidor usando <env-entry> e leia-as de um servlet usando JNDI

helder@acm.org

argonavis.com.br