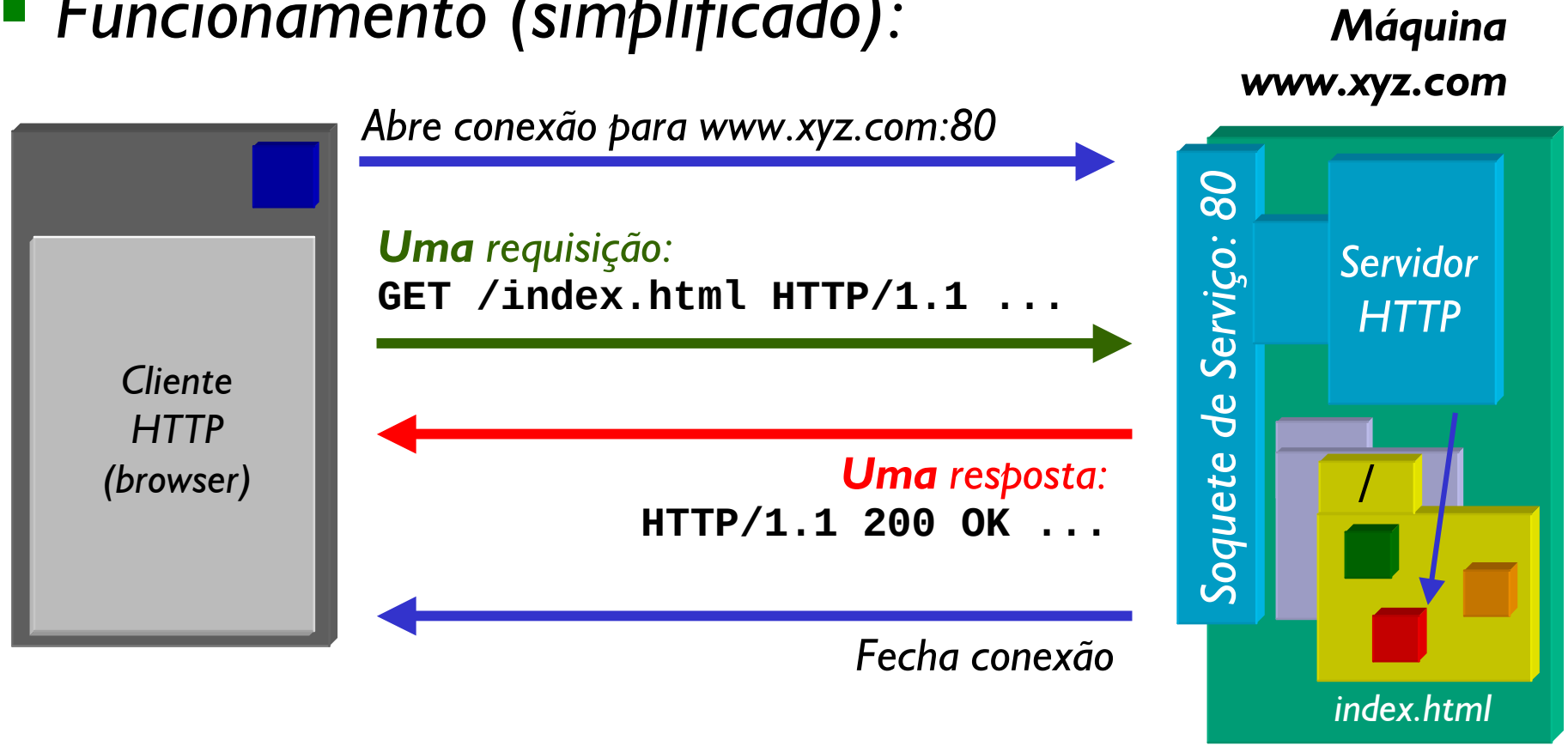
Java 2 Enterprise Edition

Aplicações Web

Helder da Rocha
www.argonavis.com.br

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

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*

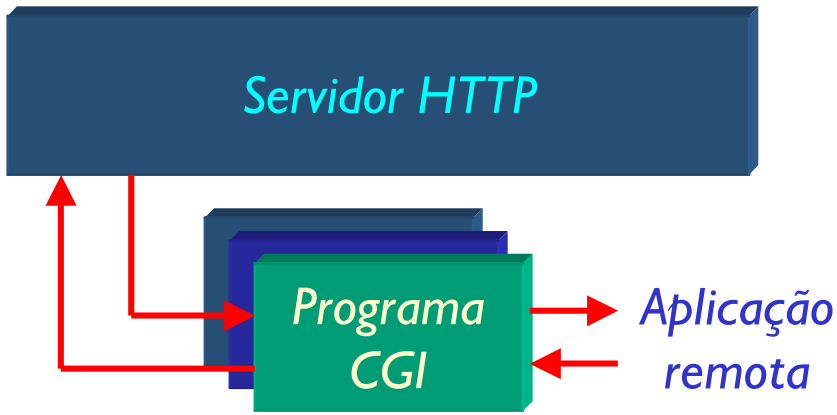
CGI - Common Gateway Interface

- ***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*

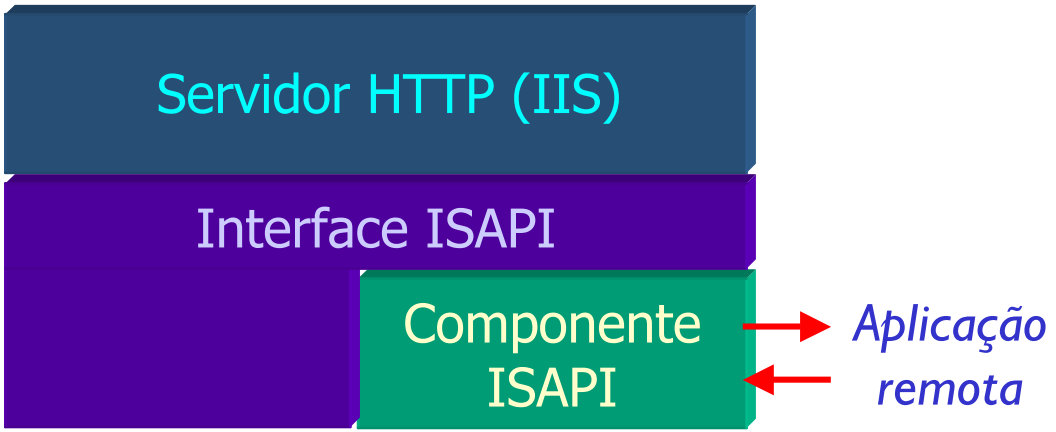
CGI é prático... Mas ineficiente!

- 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



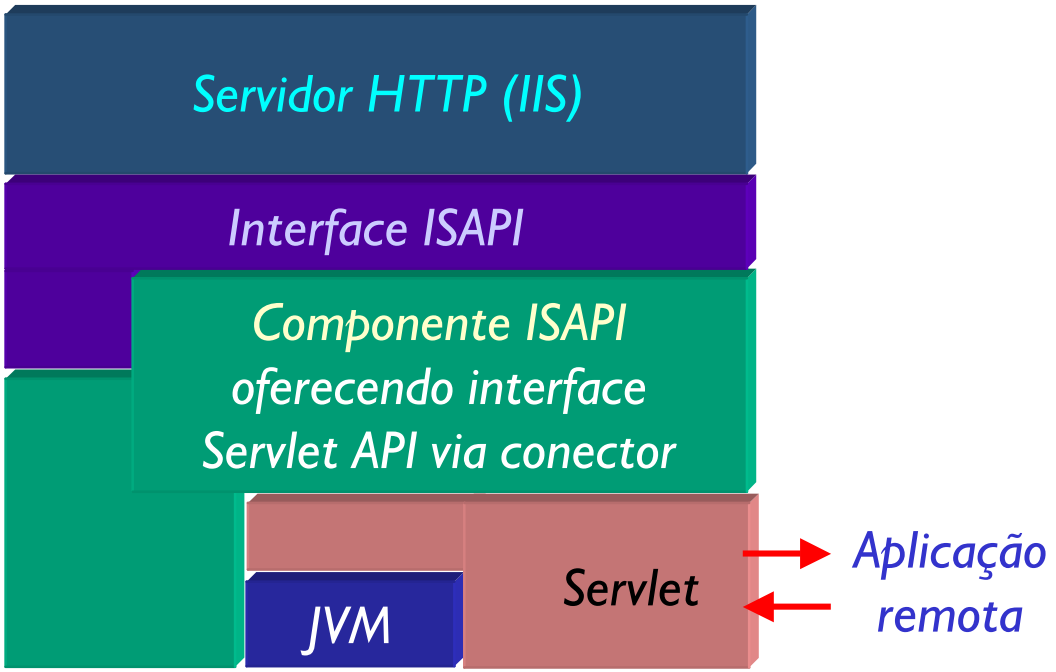
APIs do servidor

- *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*



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)*

Controle de sessão

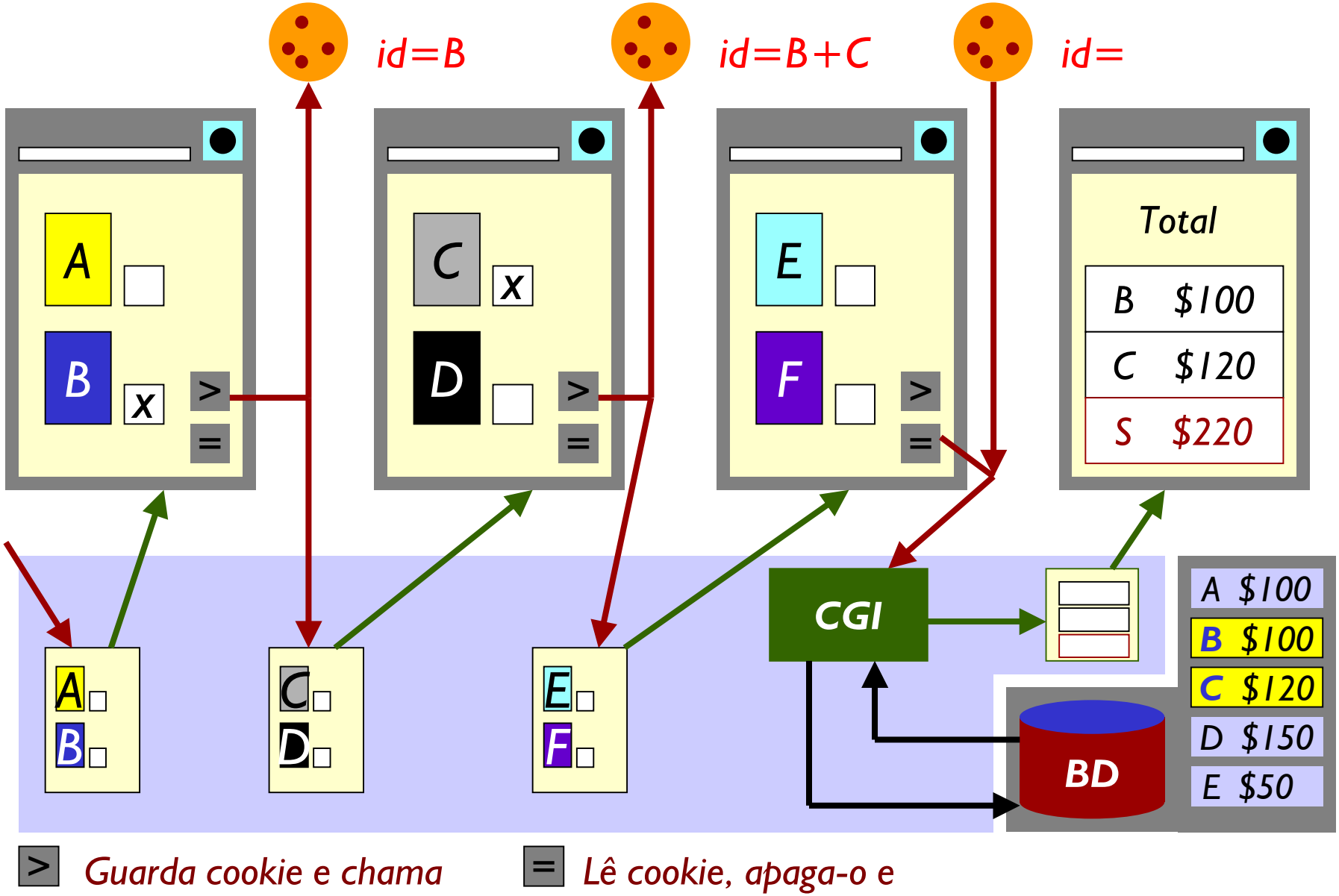
- HTTP não preserva o estado de uma sessão. É preciso usar mecanismos artificiais com CGI (ou qualquer outra tecnologia Web)
  - Seqüência de páginas/aplicações: desvantagens: seqüência não pode ser quebrada; mesmo que página só contenha HTML simples, precisará ser gerada por aplicação
  - Inclusão de 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*

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

Primeiro 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();
    }
}
```

Primeiro JSP

```
<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>
```

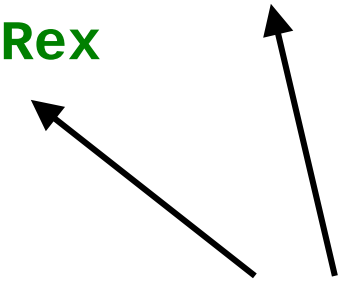
Página recebida no browser

- *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;
      }
}
```

Primeiro Bean JSP

- *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>
```

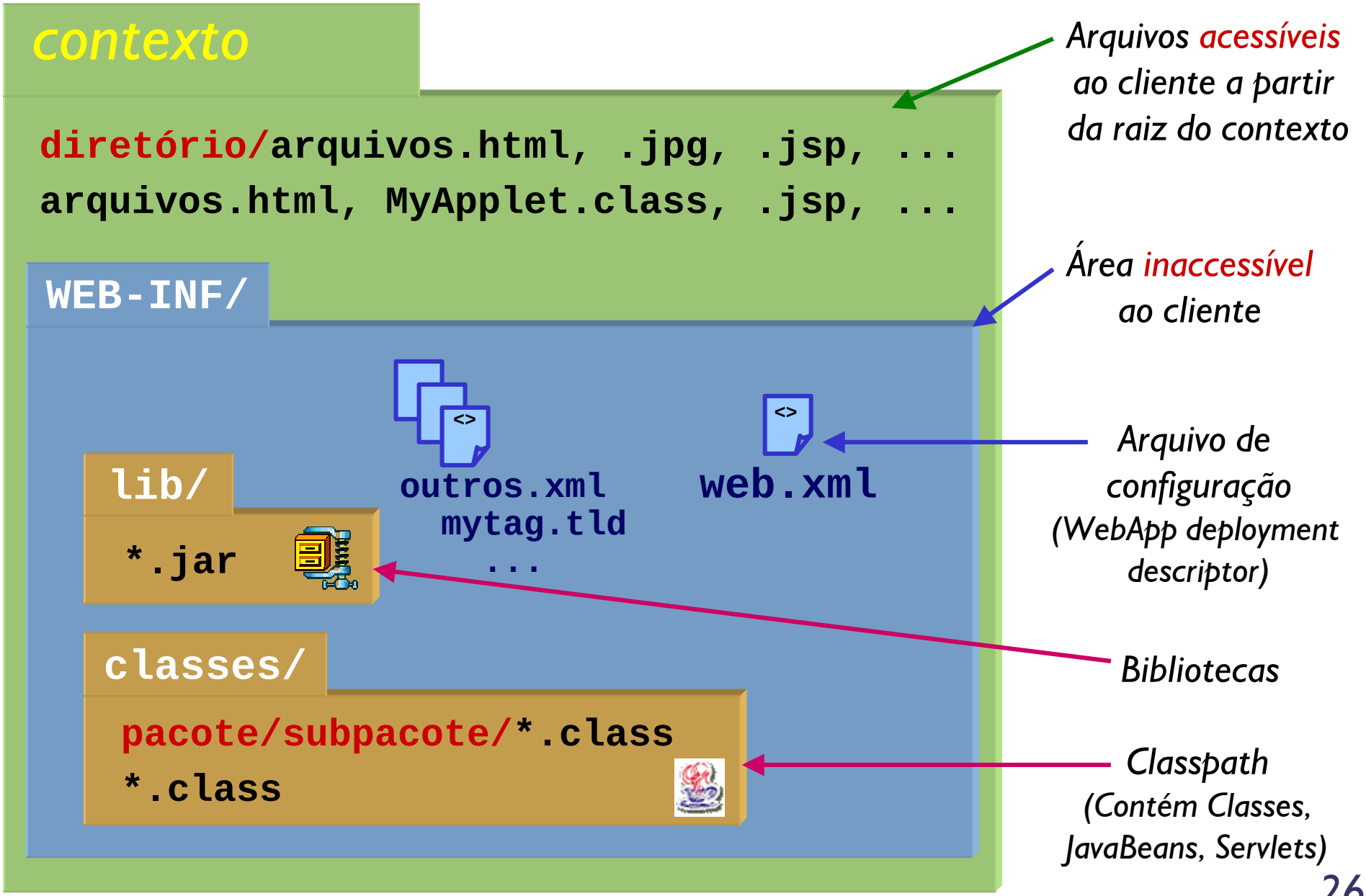
Componentes Web

- São aplicações J2EE
- Rodam em um Web Container que oferece serviços como repasse de requisições, segurança, concorrência (threads), gerência do ciclo de vida
- São compostos de servlets e páginas JSP empacotados em um arquivo WAR (tipo de JAR)
- O WAR é essencial para implantar o cliente Web J2EE, mas opcional em servidor standalone
- Os componentes de um WAR ocupam um contexto que pode ser acessado através de um cliente HTTP (browser)
  - Fisicamente, o contexto representa uma estrutura de diretórios. Logicamente, representa uma aplicação Web.
  - O contexto tem uma estrutura padrão definida em especificação

Aplicações Web (contextos)

- No JBoss, aplicações Web são implantadas copiando contextos compactados em arquivos WAR para a pasta deploy/
  - A estrutura da árvore de diretórios deve ser mantida dentro do arquivo WAR
- Todo diretório de contexto tem uma estrutura definida, que consiste de
  - Área de documentos do contexto (/), acessível externamente
  - Área inaccessível (/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



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 de acesso

- *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*

Configuração: exemplo (1/3)

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

  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>example.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

Instância de servlet de página JSP

Ordem para pré-compilar JSP

Configuração: exemplo (2/3)

```
...
  <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>
...
```

Servlet examples.myServlet foi mapeado à URL /myservlet

Sessão do usuário expira expira em 30 minutos

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

Redirecionar para esta página em caso de erro 404

Configuração: exemplo (3/3)

Recursos externos acessíveis via JNDI (java:comp/env/jdbc/MeuBanco)

```
...
  <resource-ref>
    <res-ref-name>jdbc/MeuBanco</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>CONTAINER</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
```

Ligação com nome JNDI em outro contexto pode ser feito em arquivo externo (no EAR ou configuração proprietária, ex: jboss-web.xml)

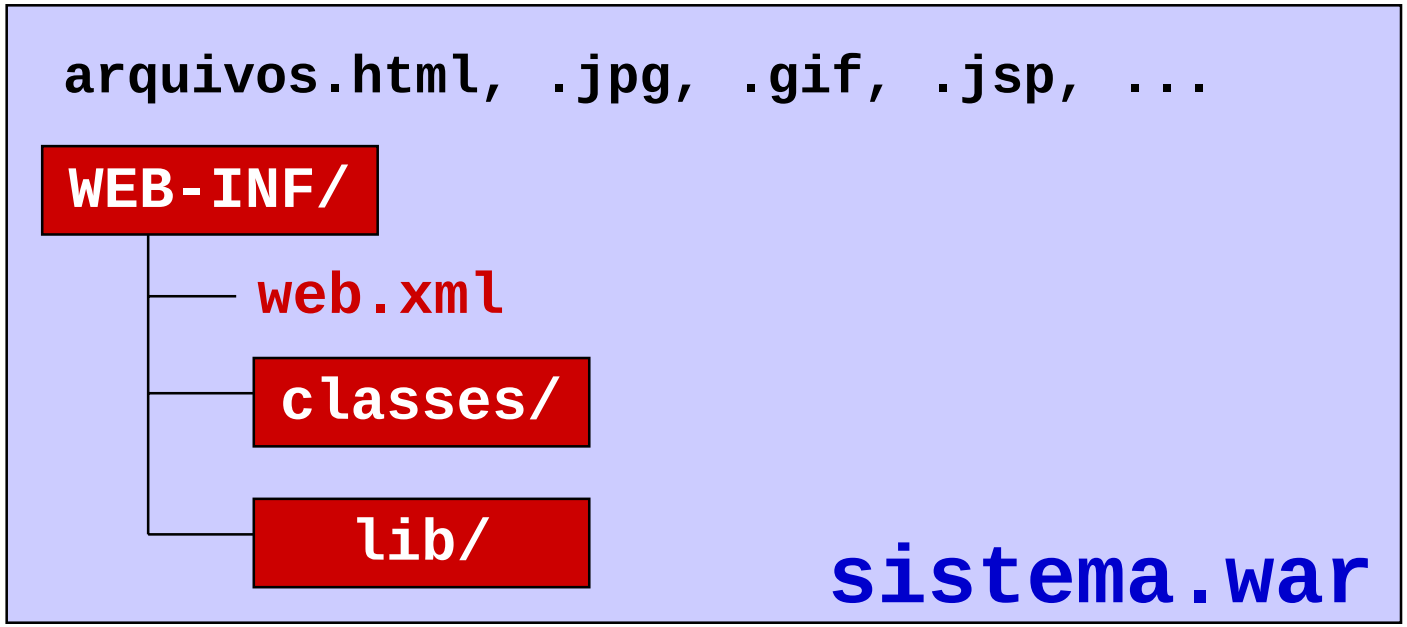
```
  <env-entry>
    <env-entry-name>valor</env-entry-name>
    <env-entry-value>34.45</env-entry-value>
    <env-entry-type>java.lang.Double</env-entry-type>
  </env-entry>

</web-app>
```

Variáveis compartilhadas pelo ambiente

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/`**



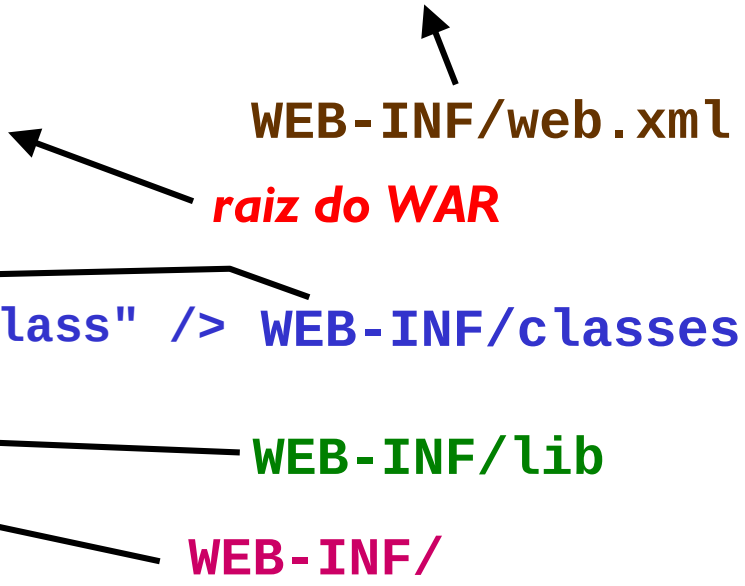
Como criar um WAR

- 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 J2EE Reference Implementation Server, use o deployment wizard
- 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

JBoss

- *Caso seja necessário configurar, no JBoss*
  - *Nomes JNDI para referências declaradas no web.xml*
  - *Nome da raiz do contexto (o EAR sempre tem precedência)*
  - *Configurações de segurança*

  pode-se criar um arquivo jboss-web.xml e empacotá-lo junto com o WAR
- *Para montar, use o DTD jboss-web_3_0.dtd. Exemplo:*

```
<jboss-web>
    <security-domain>java:jaas/mydomain</security-domain>

    <context-root>myroot</context-root>

    <resource-ref>
        <res-ref-name>jdbc/MyBank</res-ref-name>
        <jndi-name>java:/DefaultDS</jndi-name>
    </resource-ref>
</jboss-web>
```

Conclusões

- *Este capítulo apresentou uma visão superficial de aplicações Web*
  - *Detalhes sobre servlets, sessões, contextos e JSP serão vistos nos capítulos seguintes*
  - *Outros detalhes relacionados a aplicações Web não serão abordados: consulte documentação adicional no CD ou material do curso J550 (Servlets e JSP)*
- *Para utilizar componentes Web em aplicações J2EE é essencial saber implantá-las*
  - *Crie WARs e implante-os no JBoss*
  - *Familiarize-se com a sintaxe das URLs e contextos*
  - *Explore os recursos de configuração do web.xml*

Exercícios

- *1. Use os arquivos em cap09/exercicio para montar um contexto cap09 e empacotá-lo em um WAR*
  - *Os arquivos que devem ser copiados para a raiz do WAR (document root) estão na pasta web/*
  - *Os servlets e JavaBeans, que devem ser copiados para WEB-INF/classes após a compilação estão em src/. Após a execução do Ant, serão compilados e colocados em classes/*
  - *O Web Deployment Descriptor (web.xml) está em etc/*

  a) Escreva um script ou alvo do ant para montar o contexto corretamente em um diretório temporário

  b) Crie um WAR contendo os três componentes

  c) Copie o WAR para o diretório deploy/ do JBoss

  d) Acesse as aplicações via browser

  http://localhost:8080/cap09/hello.jsp
  http://localhost:8080/cap09/servlet/SimpleServlet
  http://localhost:8080/cap09/hellobean.jsp

Referências

[1] *Stephanie Bodoff. Web Clients and Components. J2EE Tutorial, Sun Microsystems, 2002*

[2] *Fields/Kolb. Web Development with JavaServer Pages, Manning, 2000*

[3] *Eduardo Pelegri Lopart, Java Server Pages 1.2 Specification, Sun, August 2001. http://java.sun.com. Referência oficial sobre JSP*

[4] *Danny Coward, Java Servlet Specification 2.3. Sun, August 2001. Referência oficial sobre servlets*

[5] *Bill Shannon, Java 2 Enterprise Edition Specification v. 1.3, Sun, July 2001 Referência oficial sobre J2EE. Descreve WARs e EARs.*

[6] *JBoss Group. JBoss User's Manual 2.44. www.jboss.org. Contém referência sobre arquivos de configuração jboss-web.xml*

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)