Java 2 Enterprise Edition

Componentes e Clientes Web

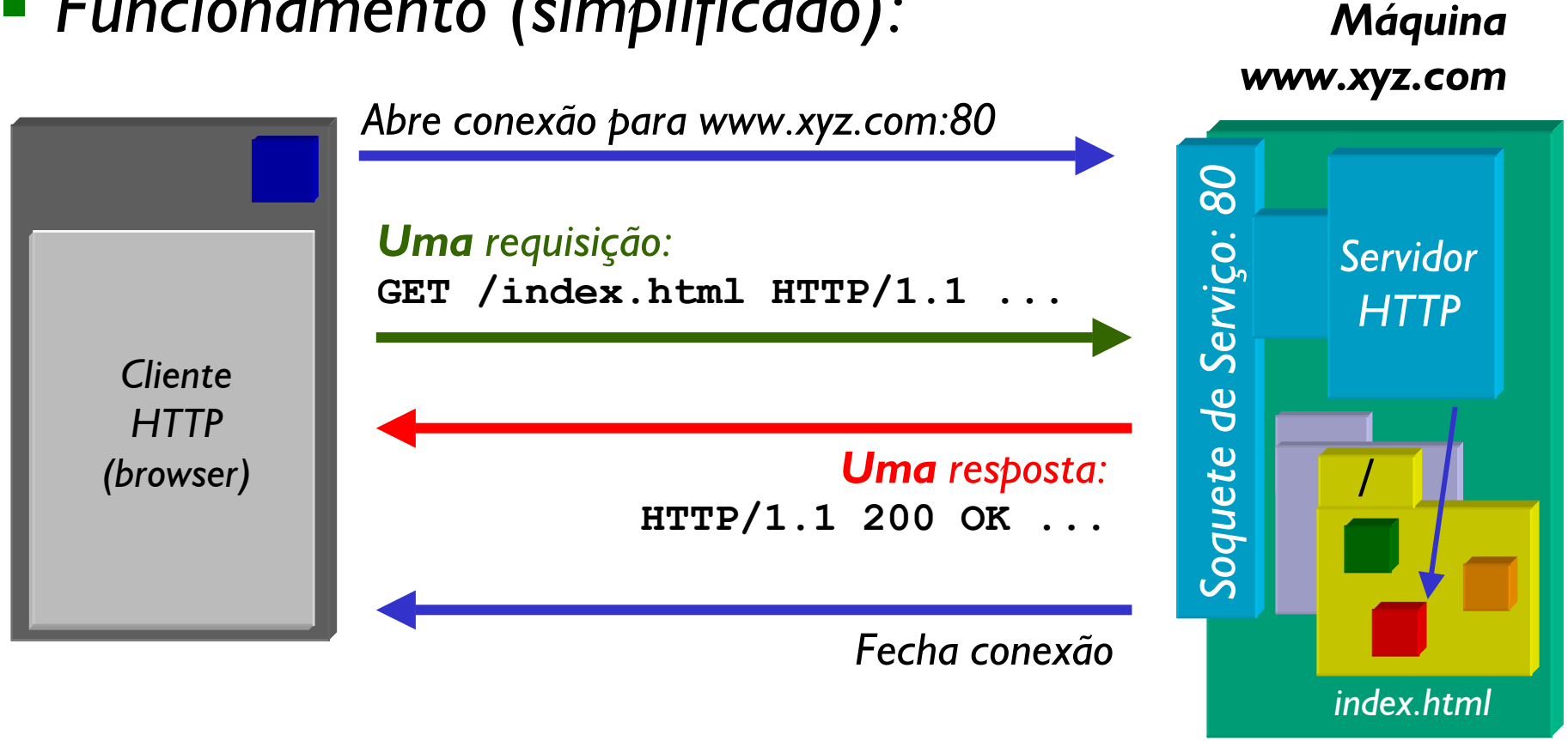
Helder da Rocha
www.argonavis.com.br

Sumário

- *Conceitos essenciais sobre a plataforma Web*
  - *Fundamentos de HTTP: métodos, respostas, requisições*
  - *Tecnologias lado-servidor: CGI, APIs e scripts*
  - *Cookies*
- *Aplicações Web em Java*
  - *Componentes Web*
  - *Servlet containers e conectores*
  - *Jakarta-Tomcat (implementação de referência)*
- *Web application archives (WAR)*
  - *Estrutura de uma aplicação Web*
  - *Arquivos WAR*
  - *Configuração e instalação de aplicações Web (web.xml)*
- *Integração J2EE*
  - *Inclusão de WARs em EARs e implantação no JBoss*

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:*

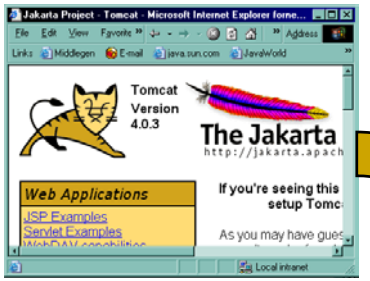
```
Content-type: text/html; charset-iso-8859-1
Refresh: 15; url=/pags/novaPag.html
Content-length: 246
Set-Cookie: nome=valor; expires=Mon, 12-03-2001 13:03:00 GMT
```

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*

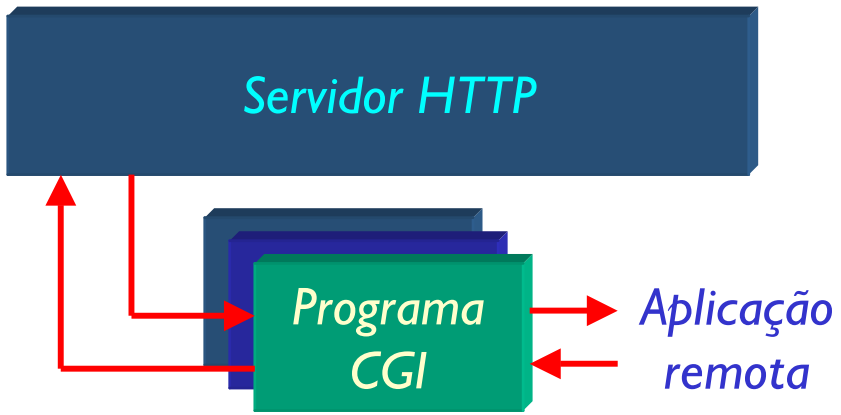
CGI - Common Gateway Interface

- ***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*

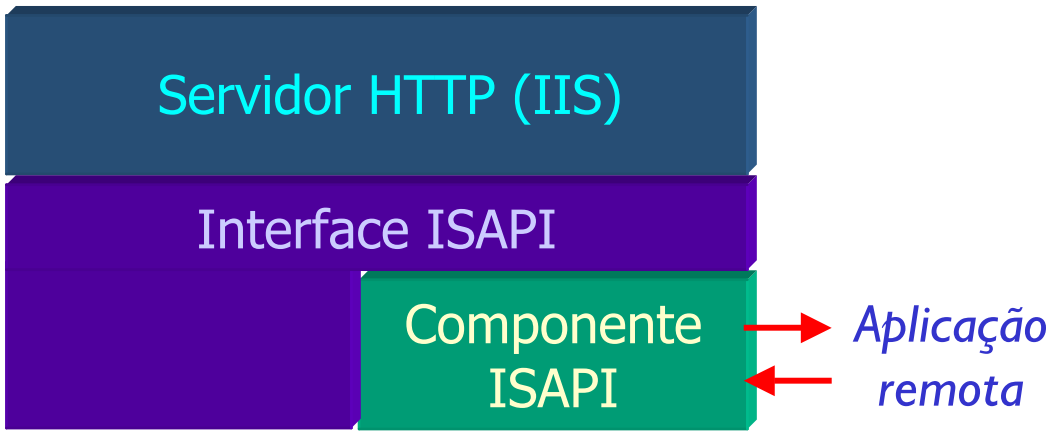
CGI é prático... Mas ineficiente!

- 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



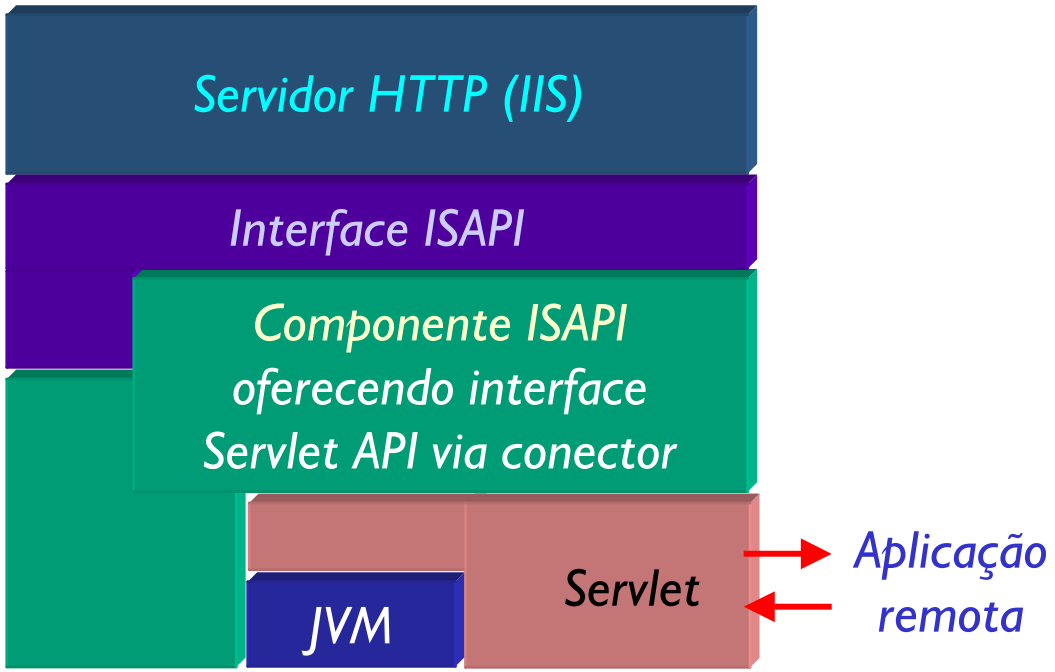
APIs do servidor

- *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*
- *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

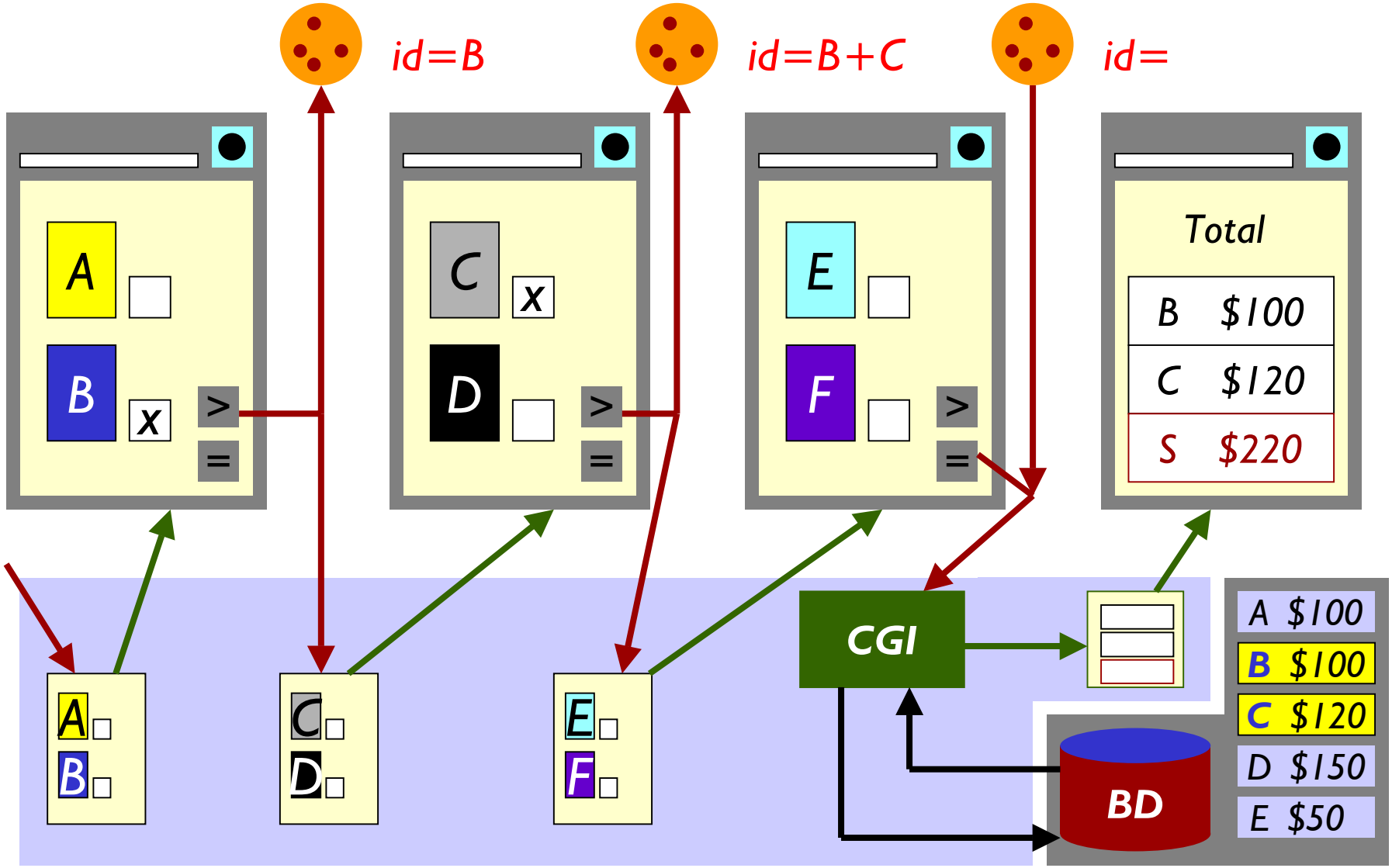
- Alguns dos mais populares:
  - Microsoft Active Server Pages (ASP)
  - Sun JavaServer Pages (JSP)
  - Macromedia Cold Fusion
  - PHP
- A página geralmente possui uma extensão de nome de arquivo diferente para que o servidor a identifique como um programa
- As página ASP, PHP, JSP, etc. são processadas 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)

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>
```

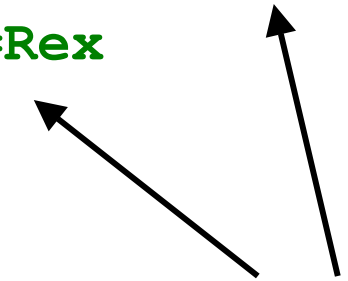
Página recebida no browser

- *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;
      }
}
```

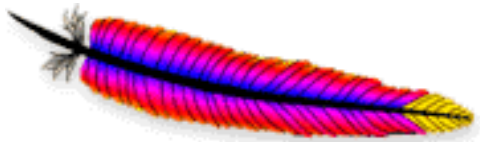
Primeiro Bean JSP

- *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>
```

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
- Arquivos importantes (nas instalações standalone)
  - $TOMCAT_HOME/conf/server.xml
    - Configuração do servidor (onde se pode configurar novos contextos)
  - $TOMCAT_HOME/common/lib/*.jar
    - Classpath global para todas as aplicações que rodam no container (use com cuidado para evitar conflitos)
- O Jakarta-Tomcat é embutido no servidor J2EE RI e integrado com algumas versões do JBoss



Clientes Web

- São aplicações J2EE que rodam em um Web Container que oferece serviços como repasse de requisições, segurança, concorrência (threads), gerência do ciclo de vida
- São compostos principalmente de componentes: servlets e páginas JSP empacotados em um arquivo WAR
- O WAR é essencial para implantar o cliente Web J2EE, mas opcional em servidor standalone.
- No Tomcat, há três formas de implantar um servlet ou JSP
  - 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)

1) Usar contexto existente

- *Exemplo: contexto /examples/*
  - *Coloque páginas Web, JSPs, imagens, etc. em* `$TOMCAT_HOME/webapps/examples/`
  - *Coloque servlets em* `$TOMCAT_HOME/webapps/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*

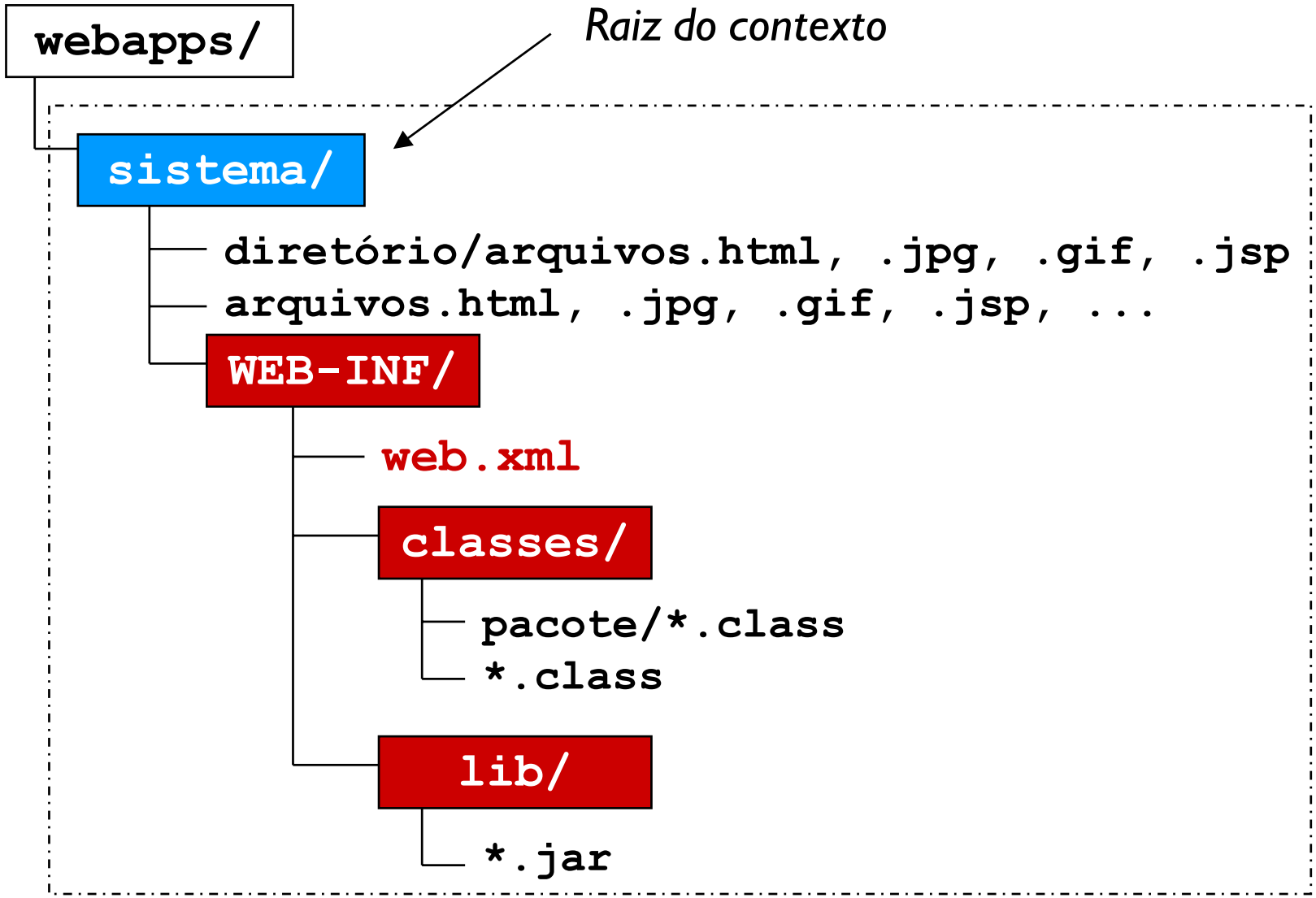
2) Criar novo contexto (depende de servidor)

- *No Tomcat: acrescente um elemento <Context> no* **`$TOMCAT_HOME/conf/server.xml`** *(global)*

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

- *docBase também pode conter endereço relativo a $TOMCAT_HOME/webapps*
- *É preciso reiniciar o servidor*

Estrutura de um contexto WebApp



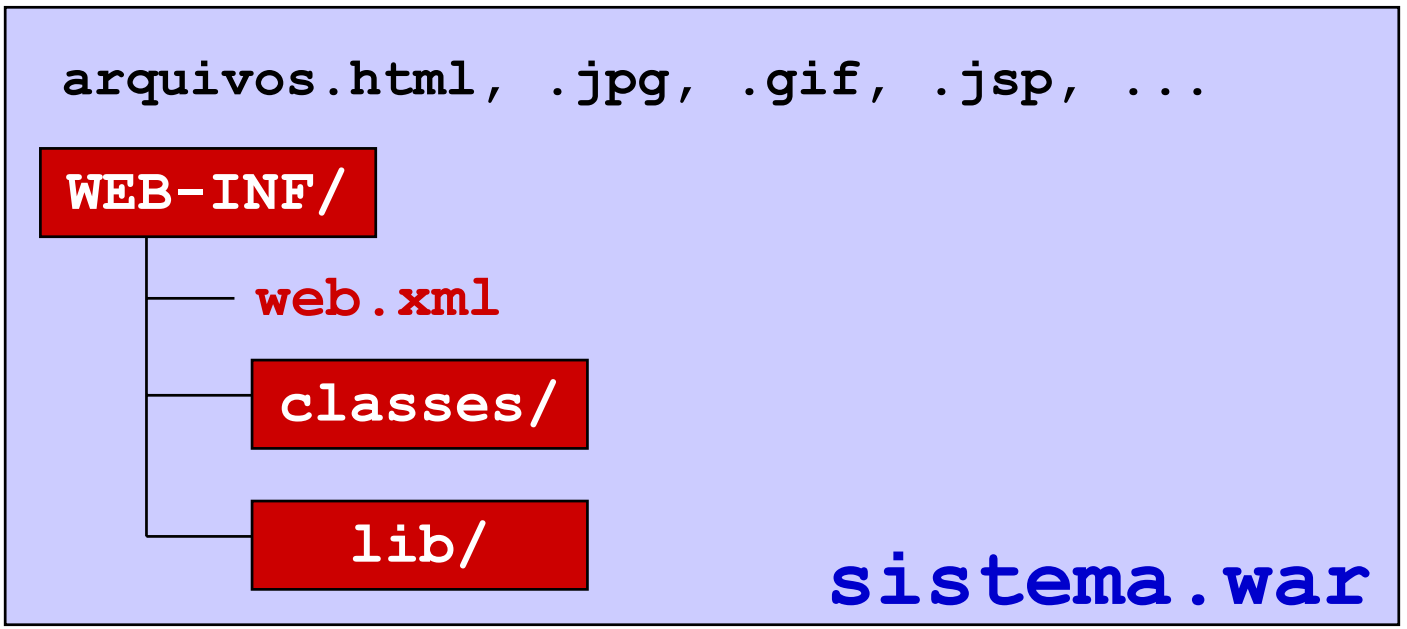
Componentes do contexto

- *Raiz geralmente define o nome do contexto.*
  - *Pode ser alterada em server.xml*
- **{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*

3) Web Archive

- *Utilizável no Tomcat e também em servidores de aplicação*
- *Não precisa criar novo contexto em server.xml*
- *Coloque JAR contendo estrutura do contexto no dir. 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: veja WCC3.html no J2EE Tutorial: (também explica como adicionar um WAR a um EAR)
  - Um aplicativo tipo WinZip
  - Ferramenta JAR: jar -cf arquivo.war -C diretorio_base .
  - Ferramenta packager: packager -webArchive <opções>
  - 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="meta/metainf.xml">
   <fileset dir="${build}/${bookstore2}" >
      <include name="*.tld />
      <exclude name="web.xml" />
   </fileset>
   <classes dir="${build}" >
      <include name="database/*.class" />
   </classes>
   <lib dir="bibliotecas" />
   <webinf dir="outros.xml" />
</war>
```

WEB-INF/web.xml

raiz do WAR

WEB-INF/classes

WEB-INF/lib

WEB-INF/

  - Veja o manual do Ant para outros exemplos e detalhes

Três níveis de configuração

- Nível da aplicação
  - Configuração definida fora do WAR: Em arquivos de configuração do fabricante (ex: jboss-web.xml) ou na configuração do EAR (caso o WAR faça parte de um)
  - Configuração do nome do contexto raiz
- Nível do WAR (contexto) - web.xml
  - Configuração aplicada a todos os componentes do WAR
  - Lista de componentes, mapeamentos, variáveis compartilhadas, recursos e beans compartilhados, serviços compartilhados, timeout da sessão, etc.
- Nivel do componente (servlet, JSP) - web.xml
  - Configuração de servlets, filtros e páginas individuais
  - Parâmetros iniciais, regras de carga, etc.

Configuração

- Use o deploytool para gerar automaticamente o seu WAR já com um deployment descriptor ...
  - Siga o deployment wizard passo-a-passo e defina os parâmetros desejados
- ... ou escreva manualmente seu web.xml e inclua-o no diretório WEB-INF do seu JAR
  - Use o DTD web-app_2_3.dtd e a especificação servlet 2.3 para incluir as informações necessárias
- Nome da raiz do contexto não é definido no web.xml
  - Por default, é o nome do WAR
  - Pode ser alterado se WAR estiver dentro de um EAR ou através de configuração proprietária do servidor (o deploytool o configura apenas para o servidor J2EE RI)

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

Servlet

Parâmetro que pode ser lido pelo servlet

Ordem para carga prévia do servlet

Declaração opcional 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.Myervlet 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)

```
...
  <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>

  <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>
```

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

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

Variáveis compartilhadas pelo ambiente

Configuração externa do WAR

- *Configuração externa ao WAR pode ser feita quando WAR é acrescentado ao EAR*
  - *Neste caso, usa-se o deployment descriptor application.xml do EAR (que fica no diretório META-INF do arquivo EAR)*

```
<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
```

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

Deployment e execução

- Depende do servidor
  - No J2EE Reference Implementation Server, use o deployment wizard
  - No JBoss, copie o WAR ou EAR para o diretório deploy do servidor
- Para executar, 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

raiz do contexto

Exercícios

1. Manipule com os exemplos do diretório cap05
   - Acrescente novos servlets e JSPs
   - Defina parâmetros de inicialização no deployment descriptor
   - Gere um novo WAR
2. Crie um WAR para cada componente do cap05
   - Cada um deve ter seu próprio web.xml
   - Coloque todos em um EAR com único contexto raiz e subcontextos, por exemplo: app/aplic1, app/aplic2
   - Faça com que todos os JSPs sejam pré-compilados

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@ibpinet.net

www.argonavis.com.br