Java e WEB

Padrão JAVA para WEB

Fábio Henrique Barros

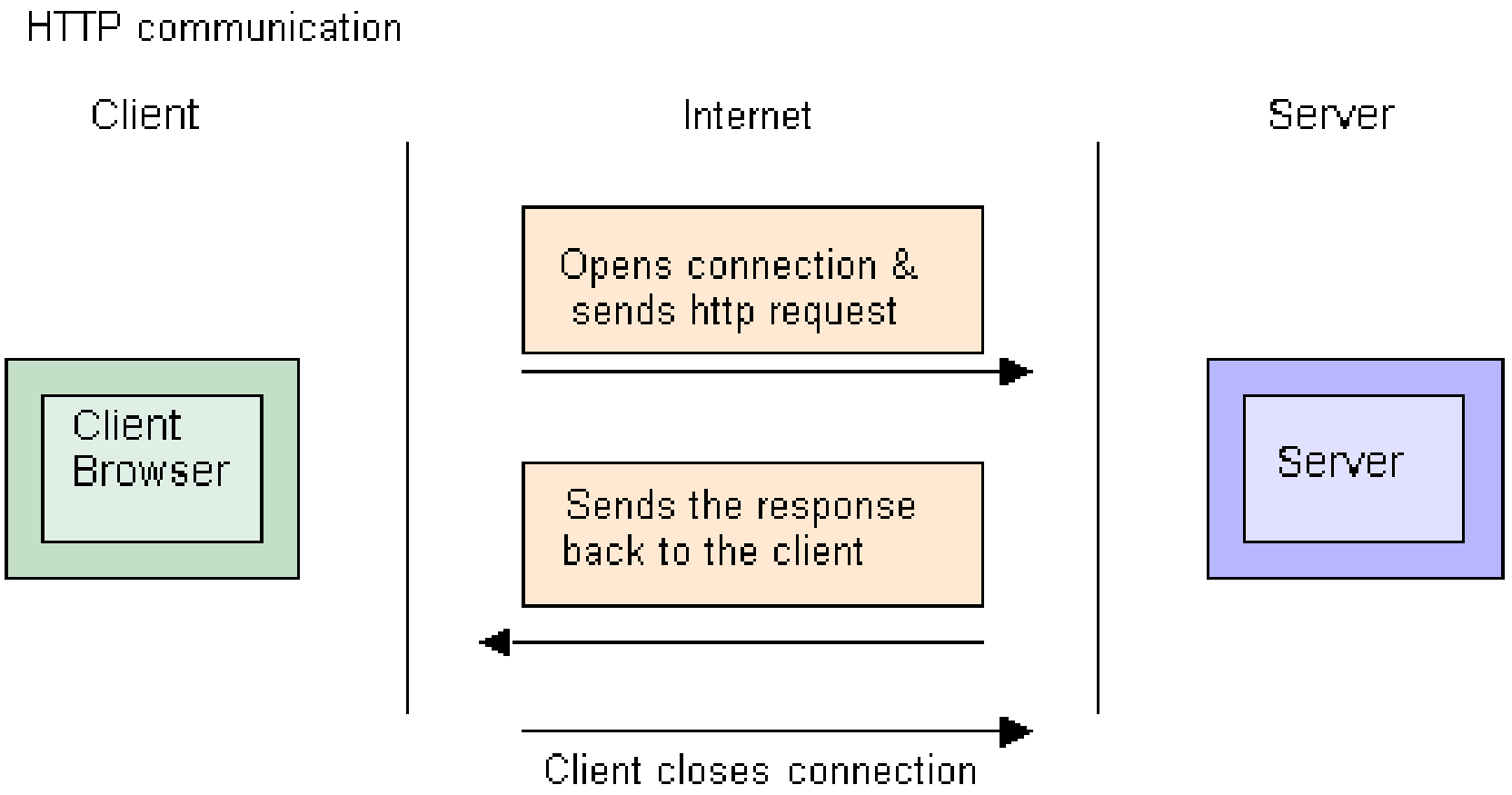
Conteúdo

- Java e WEB
  - Introdução
  - Arquitetura Servlet
  - JSP
  - EL, JSTL e Taglibs
  - Padrão MVC

Introdução

- Aplicações WEB
  - Acessada por meio de navegadores WEB
  - Facilidade de atualização da aplicação
  - Processamento centralizado
  - Interface com usuário muito poderosa
  - Diversas tecnologias disponíveis
    - CGI
    - PHP
    - ASP
    - ASP.NET
    - Java Servlets/JSP

Introdução


HTTP communication
Client
Internet
Server
Opens connection & sends http request
Client Browser
Server
Sends the response back to the client
Client closes connection

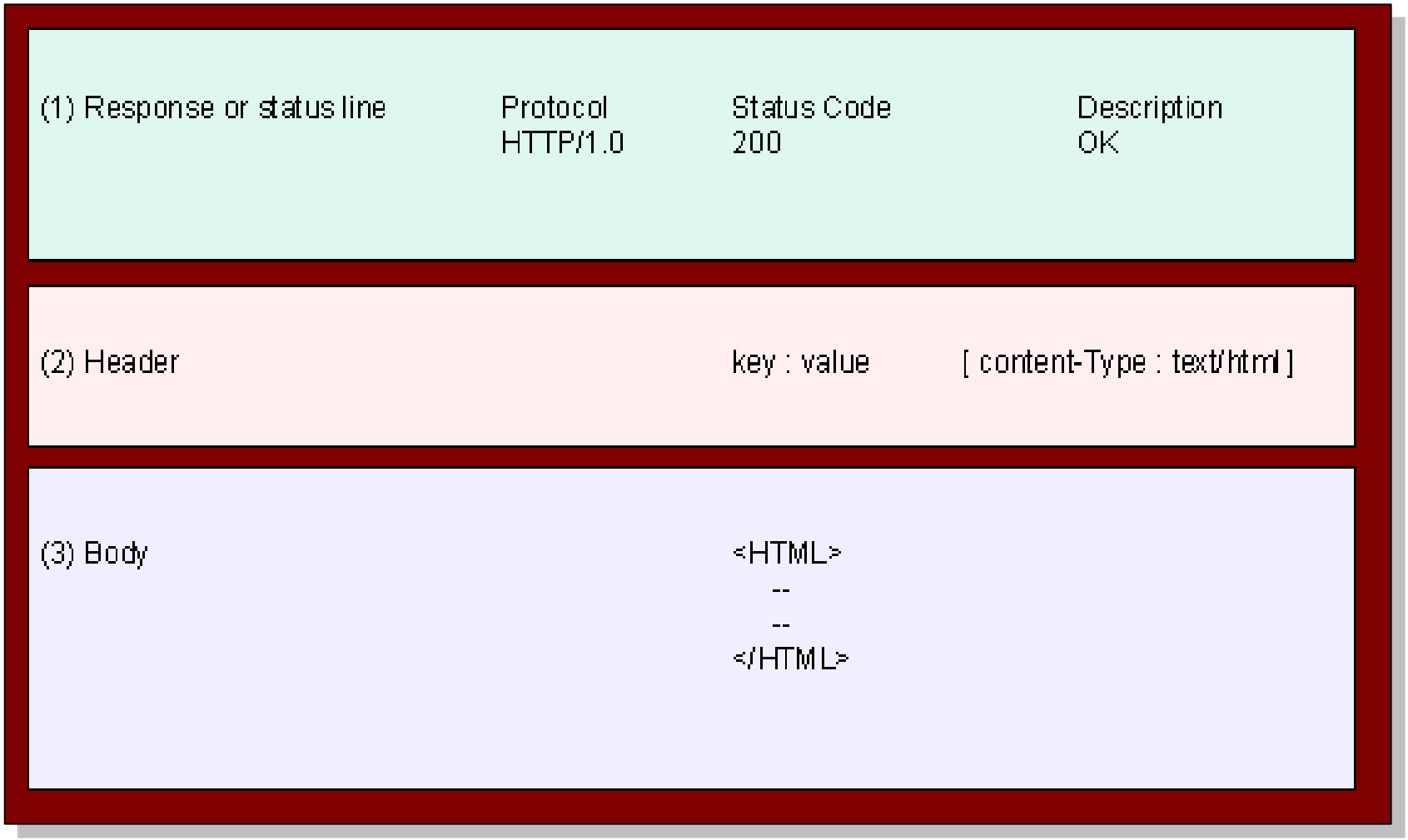
Introdução

- Requisição HTTP

(1) Request or Initial line	method GET, POST etc.	URI /product/Servlet/index.html	Protocol HTTP/1.0
(2) Header		key : value pair	
	For example	Accept: image/gif	
		User-Agent: MOZILLA/1/0	
		Content-Length: 23	
This is followed by an empty line			
(3) Body		name1=val1&name2=val2....	

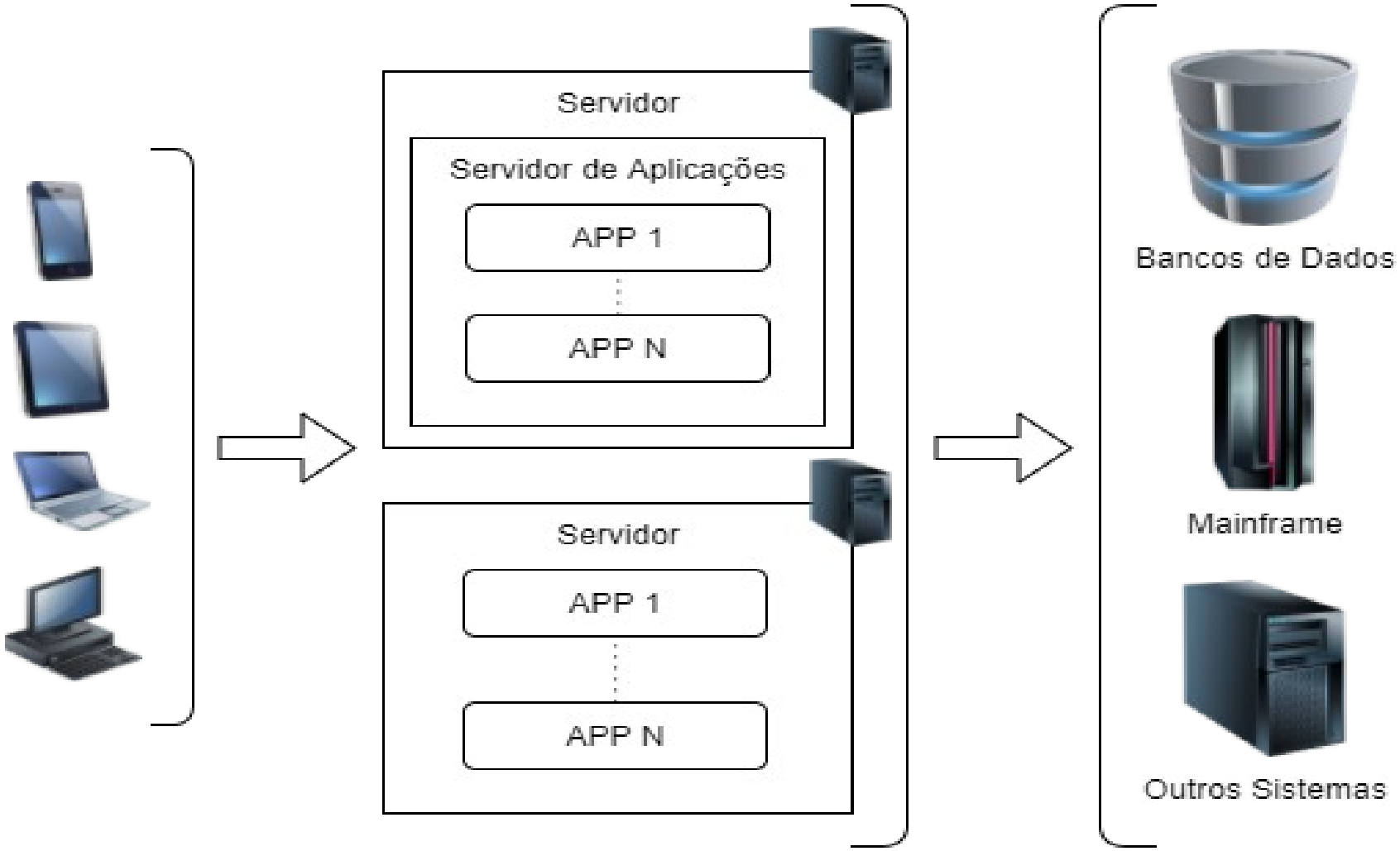
Introdução

- Resposta HTTP



Introdução

- Modelos de Arquitetura



Introdução

- Java e Web (JEE)
  - Servlets e JSP
  - Assim como a linguagem Java, são independentes de Plataforma
  - Acesso completo a API do Java
  - Melhor desempenho (na maioria dos casos) do que CGI, ASP e PHP
  - Adotado pelas grandes industrias de software mundial

Introdução

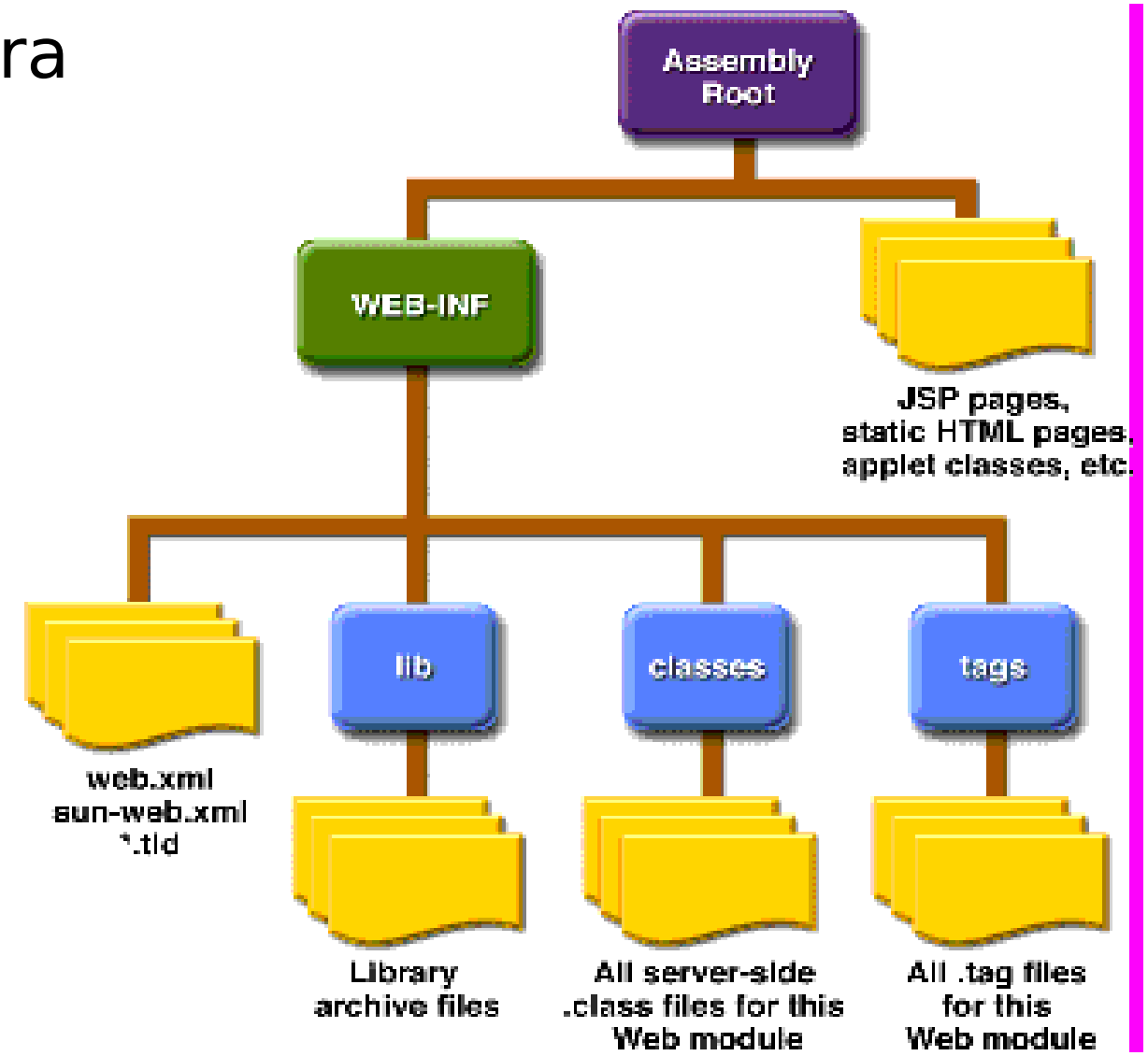
- Aplicação WEB em Java
  - Desenvolver o código dos componentes WEB
  - Configurar aplicação (web.xml)
  - Compilar aplicação
  - Empacotar binários em uma unidade instalável
  - Instalar a aplicação em um Servidor de Aplicação
    - Tomcat
    - JBoss/Wildfly
    - IBM WebSphere ...
  - Acessar a URL que referência a aplicação WEB

Introdução

- Estrutura
  - Raiz da Aplicação
    - Páginas JSP's, imagens, js, css, conteúdo estático, etc
    - Pode conter sub-diretórios
  - WEB-INF
    - web.xml – Configurações da Aplicação
    - Tld's – Tag Library Descriptor
    - classes – Contém as classes compiladas da aplicação
    - tags – Arquivos .tag, implementações de taglibs
    - lib – Contém os arquivos .jar com as bibliotecas necessárias a aplicação
  - META-INF (Opcional)
    - Informações sobre versão, etc

Introdução

- Estrutura



Introdução

- Maven como ferramenta de build. Exemplo: pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.cursoweb</groupId>
    <artifactId>cursoweb</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

</project>
```

Introdução

- Exemplo: WEB-INF/web.xml

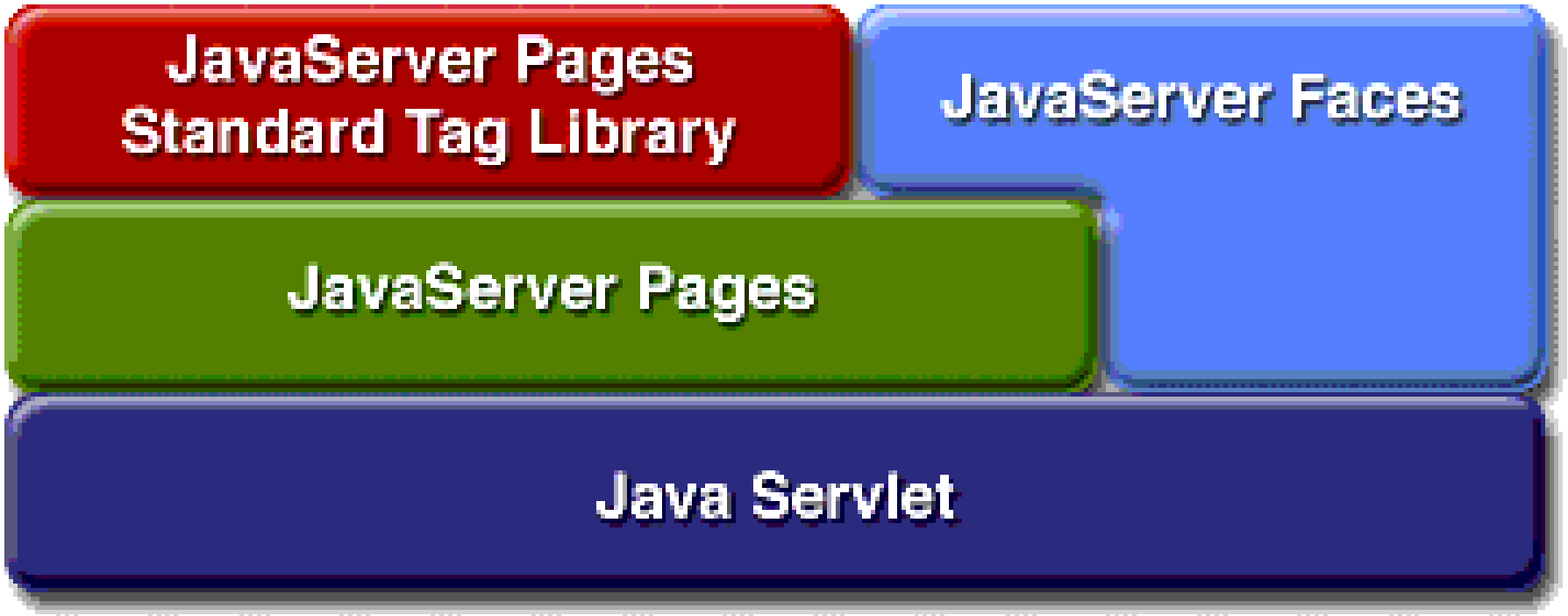
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    id="WebApp_ID" version="4.0">

    <display-name>Curso WEB</display-name>
    <module-name>cursoweb</module-name>

    <servlet>
        <servlet-name>Teste</servlet-name>
        <servlet-class>br.com.curso.TesteServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Teste</servlet-name>
        <url-pattern>/Teste</url-pattern>
    </servlet-mapping>

    <error-page>
        <error-code>505</error-code>
        <location>/geral/erro.jsp</location>
    </error-page>

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

</web-app>
```

Arquitetura Servlet

- Servlets (1ª Geração)
  - Classes Java que processam dinamicamente requisições e constroem respostas
  - Fundação de todas as tecnologias WEB
  - Os pacotes `javax.servlet.http` provê as classes e interfaces necessárias para escrever Servlets
  - A classe HttpServlet oferece os serviços HTTP através dos métodos doGet, doPost, doHead, etc
  - Cliclo de vida gerenciado pelo Container
  - Multi-thread

Arquitetura Servlet

- Servlets



Arquitetura Servlet

- Servlets
  - Classe deve herdar de `javax.servlet.http.HttpServlet`
  - A classe é instanciada e inicializada pelo Container Web
  - Um objeto do tipo `javax.servlet.ServletConfig` é passado ao Servlet (nome do Servlet e parâmetros de inicialização)
  - É passado uma referência ao contexto da aplicação, objeto do tipo `javax.servlet.ServletContext`
  - O método init é chamado
  - Quando chega um requisição ao Servidor, o método service() da classe `HttpServlet` chama o método doXXX() equivalente à requisição
  - É passado um objeto contendo as informações da requisição (`HttpServletRequest`)
  - É passado um objeto para se construir a resposta (`HttpServletResponse`)

Arquitetura Servlet

- Exemplo: UsuariosServlet.java

```
/**
 * Servlet implementation class UsuarioServlet.
 */
@WebServlet("/usuarioServlet")
public class UsuarioServlet extends HttpServlet {

    private static final long serialVersionUID = -3162419944663739724L;

    public UsuarioServlet() {
        super();
    }

    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
    }

    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
    }

}
```

Arquitetura Servlet

- Exemplo: doGet()

```
@Override
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    List<Usuario> usuarios = new ArrayList<Usuario>();
    usuarios.add(new Usuario("Fabio", "fabio@gmail.com"));
    usuarios.add(new Usuario("Maria", "maria@gmail.com"));
    usuarios.add(new Usuario("Jose", "jose@gmail.com"));
    usuarios.add(new Usuario("Pedro", "pedro@gmail.com"));
    usuarios.add(new Usuario("Ana", "ana@gmail.com"));
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Usuarios</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<ul>");
    for(Usuario usuario: usuarios) {
        out.println("<li>");
        out.println("<a href=\"mailto:" + usuario.getEmail() + "\">");
        out.println(usuario.getNome());
        out.println("</a>");
        out.println("</li>");
    }
    out.println("</ul>");
    out.println("<hr/><a href=\"/cursoweb/\">Voltar</a>");
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```

Arquitetura Servlet

- Exemplo: web.xml
  - Opcional a partir do Servlet 3.0

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <servlet>
        <servlet-name>UsuarioServlet</servlet-name>
        <servlet-class>br.com.cursoweb.servlet.UsuarioServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>UsuarioServlet</servlet-name>
        <url-pattern>/usuario</url-pattern>
    </servlet-mapping>
</web-app>
```

Arquitetura Servlet

- Exemplo: index.html

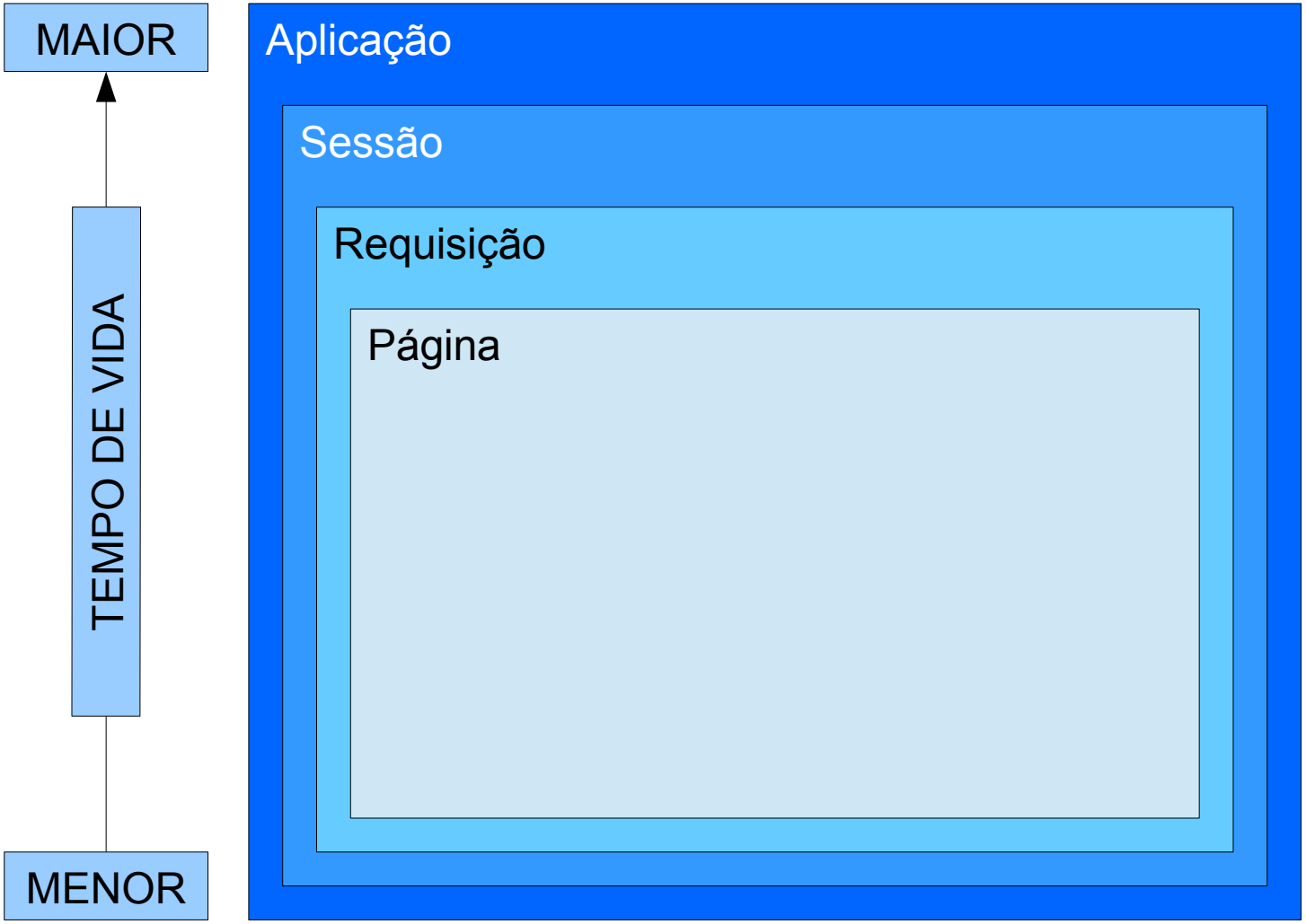
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Pagina Inicial</title>
</head>
<body>
    Ola Java EE!<br />
    <a href="usuarioServlet">Listar Usuarios</a>
</body>
</html>
```

Arquitetura Servlet

- Compartilhando Informações
  - Os vários componentes de uma aplicação Web trabalham juntos (JSP, Servlet, Filters, Listeners...)
  - É necessário compartilhar informações entre eles
  - As informações são compartilhadas através de objetos mantidos como atributos dos chamados Objetos de Escopo (Scope Objects) usando os métodos [get|set]Attribute:
    - Web Context - ServletContext
      Escopo da Aplicação, visível a toda Aplicação (Variáveis globais)
    - Session - HttpSession
      Informações relativas a sessão de um Cliente, vísivel a um conjunto de requisições
    - Request - HttpServletRequest
      Relativa a apenas uma requisição
    - Page - JspContext
      Criada pela página JSP, visível apenas ao JSP

Arquitetura Servlet

- Escopos



Arquitetura Servlet

- Compartilhando Informações
  - Acesso Concorrente
    - Processamento centralizado = várias requisições simultâneas
    - A mesma instância de um Servlet atende a diversas requisições diferentes (Controlado pelo Container)
    - Não se deve usar variáveis de instância na classe Servlet
    - Em casos onde possa ocorrer acesso simultâneo a recursos, potencial inconsistência, é necessário usar os recursos de sincronização da linguagem Java

Arquitetura Servlet

- Tratando Requisições
  - Métodos HTTP mais comuns: GET e POST
  - Extrair informações da requisição, acessar recursos externos, e montar a resposta baseado nessas informações
  - URL da requisição:
    - http://[host]:[port][request path]?[query string]
    - Context Path: "/nomeDoContexto" nome da aplicação
    - Servlet Path: "/Teste" corresponde ao caminho para um componente (Servlet)

Arquitetura Servlet

- Tratando Requisições (cont.)
  - Query String: parâmetros passados pela URL (GET), podem ser passados no corpo da requisição (POST)
  - HttpServletRequest
    - URL: **`getContextPath()`**, **`getServletPath()`**, **`getQueryString()`**
    - Aplicação: **`getSession()`**, **`getRealPath()`**, **`getRequestDispatcher()`**
    - Parâmetros: **`getParameter()`**, **`getParameterNames()`**, **`getParameterValues()`**, **`getParameterMap()`**, **`getInputStream()`**
    - HTTP: **`getMethod()`**, **`getHeader()`**, **`getHeaderNames()`**, **`getHeaders()`**, **`getCookies()`**

Arquitetura Servlet

- Tratando Requisições (cont.)
  - HttpServletResponse
    - HTTP: **`sendError()`**, **`sendRedirect()`**, **`addHeader()`**, **`setHeader()`**, **`addCookie()`**
    - Montar resposta: **`setContentType()`**, **`getWriter()`**, **`getOutputStream()`**

Arquitetura Servlet

- Exemplo: RequestServlet.java

```
@WebServlet("/requestServlet")
public class RequestServlet extends HttpServlet {
    private static final long serialVersionUID = 68020305307387277 45L;

    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {

        String nome = request.getParameter("nome");
        if (nome != null && !nome.equals("")) {
            request.setAttribute("NOME", nome);
            request.getSession().setAttribute("NOME", nome);
            request.getSession().getServletContext().setAttribute("NOME", nome);
        }

        String error = request.getParameter("error");
        if (error != null && !error.equals("")) {
            response.sendError(Integer.valueOf(error));
        }

        String redirect = request.getParameter("redirect");
        if (redirect != null && !redirect.equals("")) {
            response.sendRedirect(redirect);
        }
...
```

Arquitetura Servlet

- Exemplo: RequestServlet.java (cont.)

```
...
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Usuarios</title>");
        out.println("</head>");
        out.println("<body>");

        out.println("getRequestURL(): " + request.getRequestURL() + "<br/>");
        out.println("getContextPath(): " + request.getContextPath() + "<br/>");
        out.println("getQueryString(): " + request.getQueryString() + "<br/>");
        out.println("getMethod(): " + request.getMethod() + "<br/>");
        out.println("getProtocol(): " + request.getProtocol() + "<br/>");

        out.println("<br/>");

        out.println("NOME Request: "
           + request.getAttribute("NOME") + "<br/>");
        out.println("NOME Session: "
           + request.getSession().getAttribute("NOME") + "<br/>");
        out.println("NOME Context: "
           + request.getSession().getServletContext().getAttribute("NOME") + "<br/>");
...
```

Arquitetura Servlet

- Exemplo: RequestServlet.java (cont.)

```
...
        out.println("<hr/><a href=\"/cursoweb/\">Voltar</a>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

}
```

Arquitetura Servlet

- Exemplo: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
...
    <error-page>
        <error-code>500</error-code>
        <location>/error500.html</location>
    </error-page>
...
</web-app>
```

- Exemplo: error500.html

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Pagina de ERRO</title>
</head>
<body>
    [500] Erro interno no servidor!!!
    <hr />
    <a href="/cursoweb/">Voltar</a>
</body>
</html>
```

Arquitetura Servlet

- Sessão com o Cliente
  - Na maioria das aplicações é necessário que uma série de requisições estejam associadas
  - É necessário guardar informações sobre determinadas ações referentes a um cliente
  - Carrinho de compras, preferências de exibição, etc
  - O protocolo HTTP não guarda estado
  - A API do Java provê controle de sessões

Arquitetura Servlet

- Sessão com o Cliente
  - Acessando a Sessão
    - Objeto **`HttpSession`**
      - Acessado através do **`HttpServletRequest`**
        - **`getSession()`**, **`getSession(boolean create)`**
    - Adicionar objetos à Sessão
      - **`setAttribute()`**, **`getAttribute()`**, **`removeAttribute()`**
    - Invalidar/Destruir uma Sessão
      - **`invalidate()`**

Arquitetura Servlet

- Exemplo: configuração de sessão

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
    "http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>CursoWeb</display-name>

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

    <session-config>
        <session-timeout>10</session-timeout>
    </session-config>
</web-app>
```

Arquitetura Servlet

- Exemplo: SessionServlet

```
@WebServlet("/sessionServlet")
public class SessionServlet extends HttpServlet {
    private static final long serialVersionUID = 68020305307387277 45L;
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Usuarios</title></head>");
        out.println("<body>");

        String invalidate = request.getParameter("invalidate");
        if (invalidate != null && invalidate.equals("true")) {
               request.getSession().invalidate();
               out.println("Session invalidada!");
        } else {
               out.println("Session não invalidada!");
        }

        out.println("<hr/><a href=\"/cursoweb/\">Voltar</a>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

Arquitetura JSP

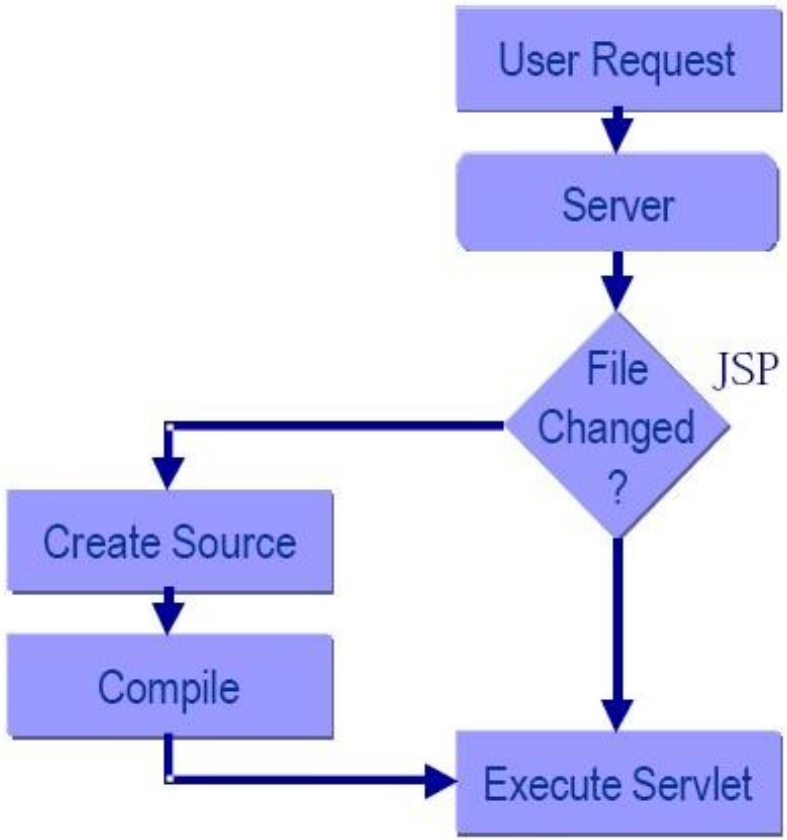
- JSP (2ª Geração)
  - Assim como a linguagem Java, é independente de plataforma
  - Código Java
  - Fácil edição de páginas HTML
  - É instalado automaticamente pelo container
  - É compilado para um Servlet
    - Recompilado quando á página é modificada
  - Vantagens em relação ao Servlets
    - Não é necessário usar println()
    - Fácil edição e manutenção de páginas HTML

Arquitetura JSP

- JSP
  - Objetos Implícitos
    - Criados pelo container e disponibilizados ao JSP
    - Equivale aos objetos passados como parâmetro para o Servlet
    - application – ServletContext
    - session – HttpSession
    - request – HttpServletRequest
    - reponse – HttpServletResponse
    - out – JspWriter
    - config – ServletConfig
    - pageContext – JspPageContext
      - Contexto válido apenas na página JSP

Arquitetura JSP

- JSP
  - As páginas JSP ficam na raiz da aplicação
  - Seguem o seguinte fluxo (controlado pelo contêiner)
  - Fase de Tradução
  - Fase de Compilação
  - Fase de Execução



Arquitetura JSP

- Elementos de Scripting JSP
  - Possibilita a inserção de código Java no servlet gerado
  - Usar o mínimo possível
  - Três maneiras
    - Expressões: <%= Expressão%>
    - Scriplets: <% Código%>
    - Declarações <%! Variáveis e Métodos%>

Arquitetura JSP

- Elementos de Scripting JSP – Expressões
  - Durante a fase de Execução
    - A Expressão é verificada e convertida para String
    - Equivalente ao out.println()
    - Possível o uso de objetos implícitos
  - Formato
    - <%=usuario.getEmail()%>
    - Não é permitido ponto e vírgula

Arquitetura JSP

- Elementos de Scripting JSP – Expressões
  - Exemplos
    - Hora atual:
      - <%= new java.util.Date() %>
    - Nome Host:
      - <%= request.getRemoteHost() %>
    - Nome Contexto:
      - <%= application.getServletContextName() %>
    - Informações da Sessão:
      - <%= session.getAttribute("bla") %>

Arquitetura JSP

- Elementos de Scripting JSP – Scriplets
  - Usada para inserir código Java arbitrário ao fluxo de Execução do JSP
  - Operações que as Expressions não executam
    - Alterar headers do response
    - Escrever informações no LOG do servidor
    - Atualizar um base de dados
    - Executar loops e código condicional
  - Possível o uso de objetos implícitos
  - Formato:
    - <% código Java %>

Arquitetura JSP

- Exemplo: usuarios.jsp

```
<%@page import="java.util.List"%> <%@page import="br.com.cursoweb.model.Usuario"%>
<%@page import="java.util.ArrayList"%> <%@page import="java.util.Date"%>
<%@page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%
    List<Usuario> usuarios = new ArrayList<Usuario>();
    usuarios.add(new Usuario("Fabio", "fabio@gmail.com", "M"));
    usuarios.add(new Usuario("Maria", "maria@gmail.com", "F"));
    usuarios.add(new Usuario("Jose", "jose@gmail.com", "M"));
    usuarios.add(new Usuario("Pedro", "pedro@gmail.com", ""));
    usuarios.add(new Usuario("Ana", "ana@gmail.com", "F"));
%>
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Lista de Usuarios</title>
</head>
<body>
    <ul>
        <% for (Usuario usuario : usuarios) { %>
        <li><a href="mailto:<%= usuario.getEmail() %>"><%= usuario.getNome() %></a></li>
        <% } %>
    </ul>
</body>
</html>
```

Arquitetura JSP

- Elementos de Scripting JSP – Declarações
  - Usada para adicionar métodos e variáveis à Classe Servlet gerada
  - Pode ser usado para inicialização e limpeza nas páginas JSP
    - `jspInit()`
    - `jspDestroy()`
  - Formato:
    - `<%! declaração de método ou variável %>`

Arquitetura JSP

- Exemplo: usuarios.jsp

```
<%@page import="java.util.List"%>
<%@page import="br.com.cursoweb.model.Usuario"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.util.Date"%>
<%@page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
...
<%!
    private String toUpper(String valor) {
        return valor.toUpperCase();
    }
%>
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
...
<body>
    <ul>
        <% for (Usuario usuario : usuarios) { %>
        <li>
            <a href="mailto:<%= usuario.getEmail() %>">
                <%= toUpper(usuario.getNome()) %>
            </a>
        </li>
        <% } %>
    </ul>
</body>
</html>
```

Arquitetura JSP

- Diretivas JSP
  - São mensagens ao Container JSP que afetam a geração do Servlet
  - Não produzem nenhuma saída na resposta
  - Podem configurar o ContentType dá página, o encoding (UTF-8, ISO-8859-1, etc), página de erro, etc
  - Sintaxe
    - `<%@ diretiva atributo="valor"%>`

Arquitetura JSP

- Diretivas JSP
  - page
    - `<%@ page language="java"`
      `contentType="text/html"`
      `pageEncoding="ISO-8859-1"%>`
    - `<%@page import="java.util.List"%>`
  - include – usada para incluir conteúdo
    - `<%@ include file="/topo.jsp" %>`
  - taglib – indica ao container que se está usando determindada taglib
    - `<%@ taglib uri="" prefix="" %>`

EL, JSTL e Taglibs

- Expression Language (EL)
  - Possibilita o acesso a propriedades dos Java Beans e de Map's
    - Exemplo: `${usuario.nome}`
  - Reconhecido como saída de texto e como atributo para qualquer ação
  - Suporta Operadores
  - Reduz o uso de Scriptlets
  - Suporte a funções EL personalizadas
    - Extensível via taglibs
    - Exemplo: `${fn:endsWith(filename, '.txt')}`
    - O JSTL 1.1 já vem com 16 funções pré-definidas

EL, JSTL e Taglibs

- Expression Language (EL)
  - Exemplo
    - Usando Scriptlets

```
<center>
    <%= usuario.getNome() %>
</center>
```

    - Usando EL

```
<center>
    ${usuario.nome}
</center>
```

EL, JSTL e Taglibs

- Expression Language (EL)
  - Exemplo
    - Usando Scriptlets

```
<center>
<%  Map m = (Map)pageContext.getAttribute("estados" );
    if (m != null) {
        Estado e = ((Estado)m.get( "CE" ));
        if( e != null ) {%>
            <%= e.getCapital() %>
<%      }
    }
 %>
</center>
```

    - Usando EL

```
<center>
${estados["CE"].capital}
</center>
```

EL, JSTL e Taglibs

- Expression Language (EL)
  - Variáveis
    - Para a expressão `${variavel}` o container procura um objeto com nome “variavel” nos vários escopos na seguinte ordem
      - page, request, session, application (Objetos Implícitos)
      - Propriedades e variáveis podem ser acessadas usando o operador ". "
    - Expressão `${variavel.nome}` é equivalente a `${variavel["nome"]}`
    - Dependendo do objeto, uma expressão `${variavel[valor]}` é interpretada diferentemente
      - Map – retorna `variavel.get(valor)`
      - List ou array – converte valor para int e retorna `variavel.get(valor)` ou `variavel[valor]`
      - Java Bean - `variavel.getValor()`

EL, JSTL e Taglibs

- Expression Language (EL)
  - Objetos Implícitos
    - `pageContext`
      - `${pageContext.servletContext}`
      - `${pageContext.request}`
      - `${pageContext.response}`
      - `${pageContext.session}`
    - `param` – Parametros da requisição
    - `header` – Cabeçalhos HTTP da requisição
    - `cookie` – Cookies
    - `initParam` – Parâmetros de inicialização do Contexto
    - É possível procurar por variáveis em escopos específicos
      - `pageScope, requestScope, sessionScope e applicationScope`

EL, JSTL e Taglibs

- Expression Language (EL)
  - Elementos
    - Literais
      - Booleano – `true` e `false`
      - Números – semelhante a Java (`${10}, ${10.5}`)
      - Strings – usando aspas duplas (`${"string"}`)
      - Nulo – `null`
    - Operadores
      - Aritiméticos: `+`, `-`, `*`, `/`, `div`, `%` e `mod`
      - Lógicos: `and`, `&&`, `or`, `||`, `not`, `!`
      - Comparadores: `==`, `eq`, `!=`, `ne`, `<`, `lt`, `>`, `gt`, `<=`, `ge`, `>=`, `le`
      - Vazio: `empty` (determina se um valor é nulo ou vazio)
      - Condicional: `cond ? A: B`

EL, JSTL e Taglibs

- Java Standart Tag Lib (JSTL)
  - Faz parte da especificação JEE
  - Encapsula funcionalidades comuns a muitas aplicações JSP
  - Conjunto de tags padrão
  - Não é preciso escrever as Tags
  - Portabilidade
- Instalação (pom.xml)

```
<dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

EL, JSTL e Taglibs

- JSTL - Bibliotecas/Declaração
  - Core (prefixo: c)

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

  - Formatação (prefixo: fmt)

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

  - Funções (prefixo: fn) – usado na EL

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

  - XML (prefixo: x)

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

  - Banco de Dados (prefixo: sql)

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

EL, JSTL e Taglibs

- JSTL – Core
  - Manipulação de variáveis

```
<c:set>
<c:remove>
```

  - Controle de Fluxo

```
<c:if>
<c:choose>
    <c:when>
    <c:otherwise>
<c:forEach>
<c:forTokens>
```

  - Manipulação de URLs

```
<c:import>
    <c:param>
<c:redirect>
    <c:param>
<c:url>
    <c:param>
```

  - Propósito geral

```
<c:out>
<c:catch>
```

EL, JSTL e Taglibs

- JSTL – Core
  - Controle de Fluxo <c:if> e <c:choose>
    - Controle condicional sem uso de Scriptlets
    - `<c:if test="..."></c:if>`
      - Execução condicional do código no corpo da tag dependendo do valor do atributo test
    - `<c:choose></c:choose>`
      - Funciona com um if-then-else
      - Usa as tags `<c:when>` e `<c:otherwise>` aninhadas para controle condicional

EL, JSTL e Taglibs

- JSTL – Core
  - Controle de Fluxo <c:forEach>
    - Iteração sobre coleções sem uso de Scriptlets
    - Atributos
      - items: a coleções de itens
      - var: item corrente
      - varStatus: status da iteração
      - begin, end, step: intervalo e numero de passos
    - Tipos de coleções
      - java.util.Collection
      - java.util.Map (java.util.Map.Entry)
      - Arrays primitivos e de objetos

EL, JSTL e Taglibs

- Exemplo:

```
<%@ page language="java" contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@ page import="br.com.curso.service.facade.GerenciadorUsuarioFacade"%>
<%@ page import="br.com.curso.service.FacadeFactory"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%
    List<Usuario> usuarios = new ArrayList<Usuario>();
    usuarios.add(new Usuario("Fabio", "fabio@gmail.com", "M"));
    usuarios.add(new Usuario("Maria", "maria@gmail.com", "F"));
    usuarios.add(new Usuario("Jose", "jose@gmail.com", "M"));
    usuarios.add(new Usuario("Pedro", "pedro@gmail.com", ""));
    usuarios.add(new Usuario("Ana", "ana@gmail.com", "F"));
    pageContext.setAttribute("usuarios", usuarios);
%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Usu&aacute;rios</title>
</head>
<body>
<c:if test="${not empty usuarios}">
  <ul>
  <c:forEach var="usuario" items="${usuarios}">
    <li><a href="mailto:${usuario.email}">${usuario.nome}</a></li>
  </c:forEach>
  </ul>
</c:if>
</body>
</html>
```

Padrão MVC

- Aplicações Web - Desafios
  - Interface Web muda com frequência
    - Necessidade de separar lógica de negócios da lógica de apresentação
  - Páginas Complexas
    - HTML Extensos
    - JavaScript
    - Conteúdo dinâmico geralmente representam uma pequena porção da página
    - A separação de papéis pode ser uma boa escolha

Padrão MVC

- Aplicações Web - Desafios
  - Não segue modelo tradicional de resposta a estímulos do MVC (Sistemas Desktop)
    - Modelo requisição-resposta
    - Quando um componente muda de estado, a página não é automaticamente atualizada
  - Requisições HTTP podem carregar somente strings como parâmetros
    - Necessidade de conversão para objetos Java
    - Tendência a geração de Erros

Padrão MVC

- Aplicações WEB - Desafios
  - Dificuldade em validar/restringir ações do usuário
    - Pouco controle sobre o navegador
    - Dependência de JavaScript
  - Pouca Quantidade de componentes gráficos
  - Interoperabilidade
    - As páginas devem aparecer e ser comportar de forma semelhante em todos os navegadores
  - Difícil de realizar testes automáticos

Padrão MVC

- Padrão MVC
  - Model
    - Objetos de dados (Entidades de Negócio)
  - View
    - Apresenta dados ao cliente
    - Captura ações do cliente
  - Controller
    - Reage a ações do usuário, atualizando o Model da maneira apropriada
    - Determina o fluxo de navegação

Padrão MVC

- Padrão MVC
  - O cliente interage com o View e envia uma requisição
  - O Controller recebe a requisição e a delega, geralmente, para camada de negócios
  - O Model é atualizado e o fluxo volta para o Controller
  - O Controller decide o que será mostrado e qual view será mostrado
  - O View exibe ao cliente os dados do Model disponibilizados pelo Controller

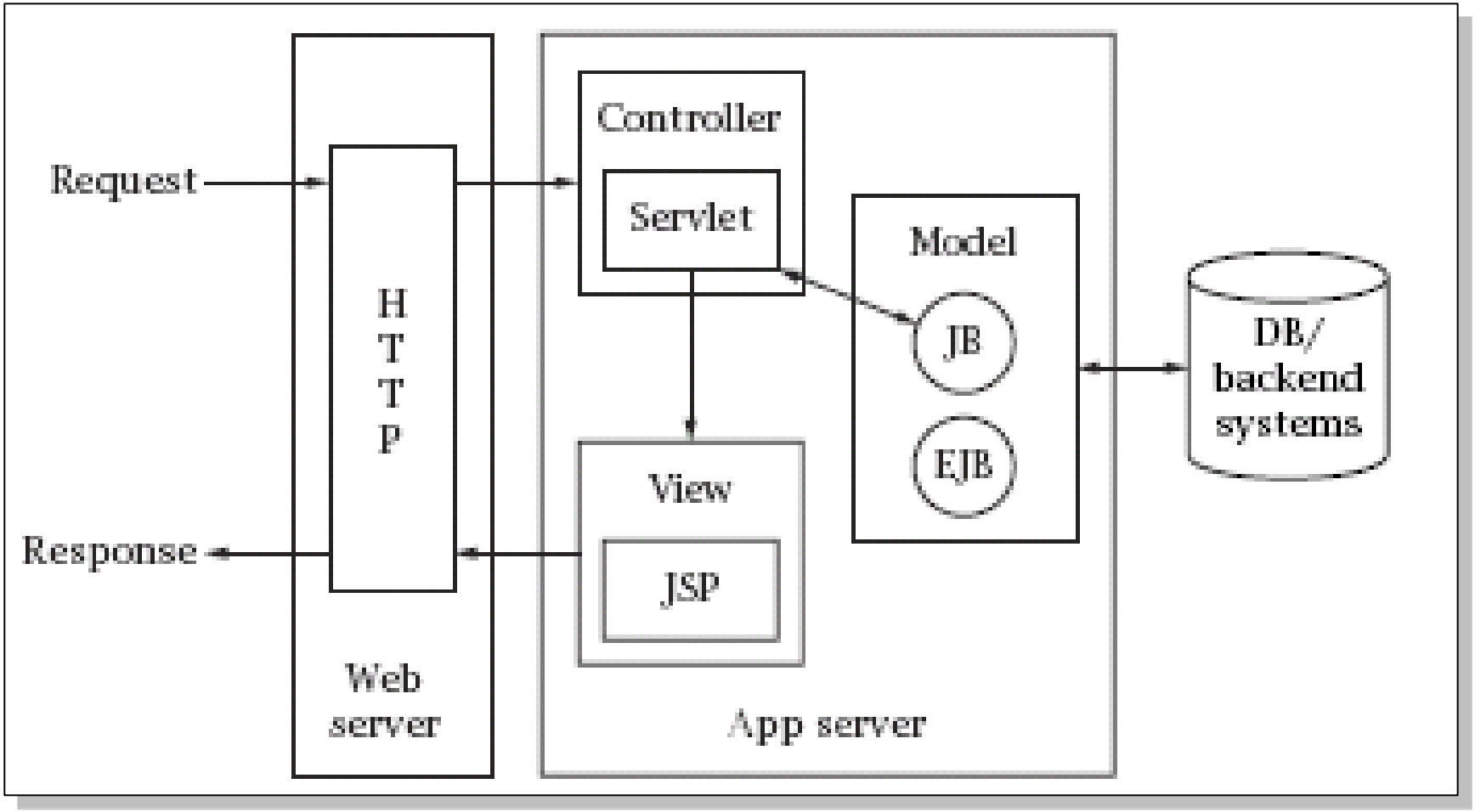
Padrão MVC

- Padrão MVC em Java
  - Servlets
    - Classe Java
    - Código HTML gerado pelo programador
    - Difícil escrita
    - Difícil manutenção
  - JSP
    - Página de texto
    - Mais fácil de desenvolver
    - Código Java misturado com código HTML
    - O número de JSP's pode crescer muito, difícil manutenção

Padrão MVC

- Padrão MVC em Java
  - Servlets (Controller)
    - Classes Java devem conter apenas código Java
    - Tem acesso a Camada de Negócio
    - Podem direcionar a requisição para outros JSP's
  - JSP (View)
    - São páginas e devem conter apenas lógica de apresentação
  - Camada de Negócio e Entidades de Negócio (Model)

Padrão MVC


Request
Response
H
T
T
P
Web
server
Controller
Servlet
View
JSP
Model
JB
EJB
App server
DB/
backend
systems

Padrão MVC

- Exemplo: controler - UsuarioMvcServlet

```
@WebServlet("/usuarioMvcServlet")
public class UsuarioMvcServlet extends HttpServlet {
    private static final long serialVersionUID = 68020305307387277 45L;

    @Override
    protected void doGet(HttpServletRequest request,
       HttpServletResponse response) throws ServletException, IOException {

        List<Usuario> usuarios = new ArrayList<Usuario>();
        usuarios.add(new Usuario("Fabio", "fabio@gmail.com"));
        usuarios.add(new Usuario("Maria", "maria@gmail.com"));
        usuarios.add(new Usuario("Jose", "jose@gmail.com"));
        usuarios.add(new Usuario("Pedro", "pedro@gmail.com"));
        usuarios.add(new Usuario("Ana", "ana@gmail.com"));

        request.setAttribute("usuarios", usuarios);
        request.getRequestDispatcher("/usuarioMvc.jsp")
               .forward(request, response);

    }
}
```

Padrão MVC

- Exemplo: view – usuarioMvc.jsp

```
<%@ page language="java" contentType="text/html"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>Usu&aacute;rios</title>
</head>
<body>
   <c:if test="${not empty usuarios}">
      <ul>
         <c:forEach var="usuario" items="${usuarios}">
            <li><a href="mailto:${usuario.email}">${usuario.nome}</a></li>
         </c:forEach>
      </ul>
   </c:if>
</body>
</html>
```