Arquitetura de Software na Web Atual: Processamento no Servidor

Edson Alves de Oliveira Junior
Renata Pontin de Mattos Fortes

Universidade de São Paulo (USP)
Instituto de Ciências Matemáticas e de Computação (ICMC)
Caixa Postal 668, 13560-970 – São Carlos, SP, Brasil
e-mail: edsonjr@icmc.usp.br

Resumo: Este trabalho apresenta uma visão geral dos conceitos de arquiteturas e padrões Web, bem como o mecanismo de processamento no lado servidor de dados provenientes de navegadores Web. Uma rápida discussão do padrão MVC e da arquitetura em três camadas é apresentada como base para o entendimento do funcionamento de tecnologias que permitem o processamento em servidores Web. Além disso, são apresentados os conceitos básicos sobre a tecnologia *JavaServer Pages* (JSP) para a construção de páginas Web com conteúdo dinâmico. Os principais elementos do JSP são apresentados, bem como alguns exemplos práticos ilustrando suas sintaxes e comportamentos. Uma discussão sobre a tecnologia JSP é apresentada ao final deste trabalho.

Palavras-chave: arquiteturas Web, Java, JavaServer Pages, padrões Web, processamento no servidor.

Abstract: This work presents the foundations on web architectures and patterns, as well the server-side processing mechanism for data provided by web browsers. A briefly discussion about the MVC pattern and the three-tier architecture is presented as the basis for understanding the technologies that allow server-side processing. Moreover, the basic concepts around JavaServer Pages (JSP) technology are presented to aim at the construction of web pages with dynamic content. The main JSP elements are presented as well practical examples illustrating its sintaxes and behaviour. At the end, a discussion around the JSP technology is presented.

Keywords: *Web architectures*, *Java*, *JavaServer Pages*, *Web patterns*, *server-side processing*.

Sumário

Capítulo

1

Introdução

O desenvolivmento de aplicações Web tem crescido exponencialmente na última década com a consolidação da Internet como um meio de divulgação e comercialização de produtos e serviços com baixo custo e ampla abrangência. Agreaga-se a este fato o avanço da capacidade de tréfego nas redes e as máquinas servidoras com maior capacidade de armazenamento e processamento.

Assim, novas tecnologias vêm surgindo e permitindo o desenvolvimento de aplicações Web cada vez mais robustas e com maior capacidade de interação com o usuário. Algunas tecnologias já conhecidas e consolidadas no mercado como, por exemplo, PHP, JSP e ASP vem sendo amplamente adotadas como solução para o desenvolvimento de aplicações que atendem à demanda e aos requisitos de clientes cada vez mais interessados em disponibilizar suas informações e produtos e serviços na Internet. Vários são os casos de sucesso de empresas de grande porte que adotam tais tecnologias para o desenvolvimento de suas aplicações Web.

Estas tecnologias baseiam-se, na maioria delas, em padrões e arquiteturas Web já difundidas e consolidadas na literatura como é o caso do padrão Modelo-Visão-Controlador (*Model-View-Controller* - MVC) e a arquitetura em três camadas (*Three-Tier Architecture*). Estes padrões e arquiteturas guiam o desenvolvedor a construir aplicações Web que facilitam a sua manutenção e evolução, uma vez que cada parte da aplicação possui um

papel bem definido e independente de acoplamento facilitando, assim, a manutenção da aplicação.

Dessa forma, este trabalho apresenta os conceitos fundamentais sobre o padrão MVC e a arquitetura em três camadas, bem como os fundamentos da tecnologia JSP para o desenvolvimento de aplicações Web com conteúdo dinâmico processado por servidores Web. Assim, o capítulo 2 apresenta os conceitos básicos sobre tais padrões e arquiteturas Web, além do mecanimso de processamento em servidores Web. O capítulo 3 apresenta os fundamentos da tecnologia JSP, assim como exemplos práticos de sua utilização. O capítulo 4 apresenta as conclusões acerca deste trabalho. Nos capítulos 2 e 3 são apresentadas considerações iniciais e finais sobre o assunto em questão.

Capítulo

2

Arquiteturas, Padrões e Processamento em Servidores Web

2.1 Considerações Iniciais

As aplicaçoes Web (*WebApps*, também conhecidas como aplicações baseadas na Web) se caracterizam por possuir um amplo conjunto de conceitos e terminologias associadas, além de possuírem características de aplicações hipermídia (Pressman, 2006).

Esse tipo de sistema não exige que a sua execução seja realizada mediante sua prévia instalação local, assim como é feito com aplicações *desktop*. Muito pelo contrário, a execução da grande maioria de aplicações Web acontece somente com o uso de um navegador Web como, por exemplo, o *Mozilla Firefox* (Mozilla, 2007). Aplicações Web são sistemas que possuem um alto grau de interação (Kerer e Kirda, 2001) além de atender simultaneamente diversos usuários, distribuídos em locais distintos fisicamente (Hendrickson e Fowler, 2002). Além disso, existe ainda a necessidade de disponibilização contínua e rápida de tais aplicações.

Para garantir estas características e o desenvolvimento apropriado desse tipo de aplicação, uma subárea da Engenharia de Software vem ganhando destaque, a Engenharia de Web (ou WebE - *Web Engineering*) (Pressman, 2006), sendo que uma de suas principais atividades é o Projeto Arquitetural gerando como artefato de saída uma arquitetura Web.

Além da arquitetura Web também é importante destacar o papel das tecnologias que tornam possível a comunicação entre o navegador Web, onde a aplicação está sendo apresentada ao usuário, e o servidor Web, onde os dados enviados pelo cliente (navegador Web) são processados produzindo, assim, uma resposta ao cliente. Atualmente, existem várias tecnologias que permitem tal interação.

Dessa forma, este capítulo apresenta os conceitos fundamentais sobre arquiteturas e padrões arquiteturais para a Web, bem como tecnologias que permitem a interação entre cliente e servidor por meio do processamento dos dados provenientes dos navegadores Web. É tomado como foco o processamento no lado servidor e a tecnologia *Java Server Pages (JSP)* para ilustrar os conceitos relacionados. Processos de desenvolvimento de Engenharia de Web não serão considerados, pois não é o foco deste trabalho.

2.2 Arquiteturas e Padrões Web

Atualmente a literatura apresenta um conjunto de vários padrões arquiteturais para sistemas distribuídos e interativos. As aplicações Web são consideradas sistemas interativos sendo que o padrão mais conhecido é o MVC (*Model-View-Controller*) (Buschmann et al., 1996; Pressman, 2006).

O MVC tem como objetivo desacoplar a interface da navegação e do comportamento da aplicação permitindo uma manutenção mais fácil e uma maior reutilização. O MVC é formado por três componentes principais, sendo eles:

- **Modelo (*Model*)**: engloba o conteúdo e a lógica de processamento específicos da aplicação, incluindo todos os objetos de conteúdo (persistentes) e dados externos de informação;
- **Visão (*View*)**: contém todas as funções específicas da interface e permite a apresentação do conteúdo e da lógica de processamento, incluindo todos os objetos de conteúdos, dados de informação e a funcionalidade de processamento requerida pelo usuário final;
- **Controlador (*Controller*)**: gera o acesso ao Modelo e à Visão e coordena o fluxo de dados entre eles.

Uma descrição básica do comportamento de aplicações que seguem o padrão MVC poderia ser: a Visão dispara eventos (baseados na entrada do usuário) ao Controlador que modifica o estado do Modelo e, em seguida, a Visão busca os dados do Modelo.

A Figura 2.1 ilustra o funcionamento básico do padrão MVC com base nos seus componentes principais.

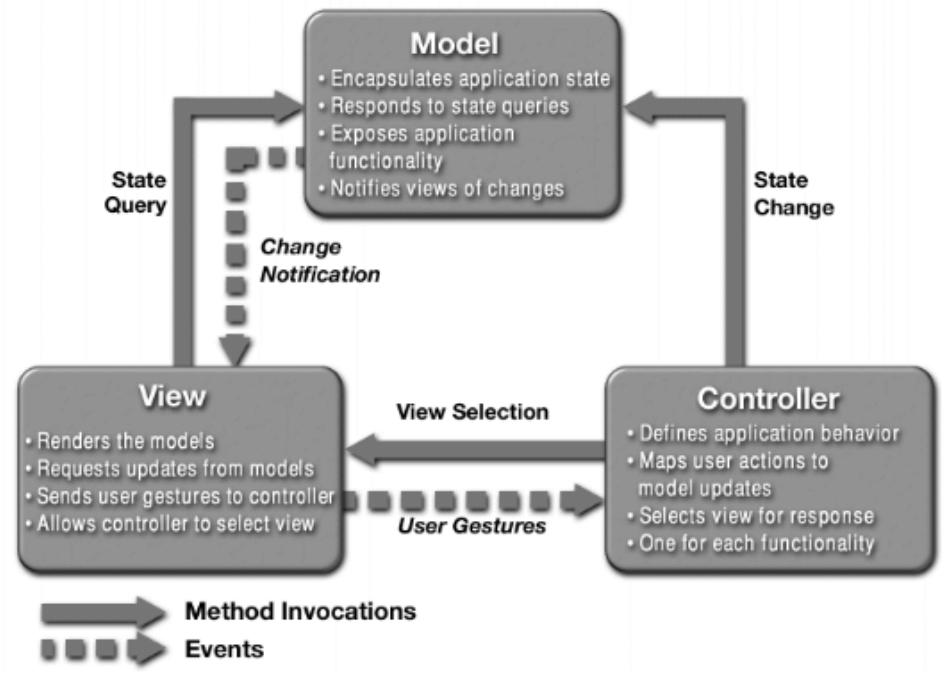

Figura 2.1: Representação gráfica do padrão MVC (Sun, 2007b).

Nesta figura podemos perceber que a Visão dispara um evento para o Controlador, que por sua vez chama um método que muda o estado do Modelo. Assim que o Modelo tem seu estado modificado, este informa à Visão, por meio do Controlador, sobre o seu novo estado (conjunto de valores dos atributos de um objeto, por exemplo). Em seguida, a Visão busca os dados diretamente do Modelo.

O Modelo pode ser formado pelas entidades que armazenam os dados que são apresentados pela Visão que, por sua vez, pode ser uma interface gráfica. O Controlador pode ser uma ou mais classes que possuem métodos que permitem que o Modelo seja atualizada a partir de eventos disparados por uma ou mais Visões, conforme mostra a Figura 2.2 com maiores detalhes dos componentes MVC.

Outro padrão de arquitetura bastante referenciado na literatura para sistemas Web é a Arquitetura em 3 Camadas (*Three-Tier Architecture*) (Reese, 2000; Sadoski e Comella-Dorda, 1997). A arquitetura em 3 camadas é um tipo de arquitetura cliente/servidor na qual a interface com o usuário, a lógica de processo, o armazenamento de dados e o acesso a dados são desenvolvidos e mantidos em módulos independentes. A forma como é concebida a arquitetura em 3 camadas permite que cada um dos módulos seja atualizado ou replicado de maneira independente com relação aos requisitos de tecnologias utilizadas.

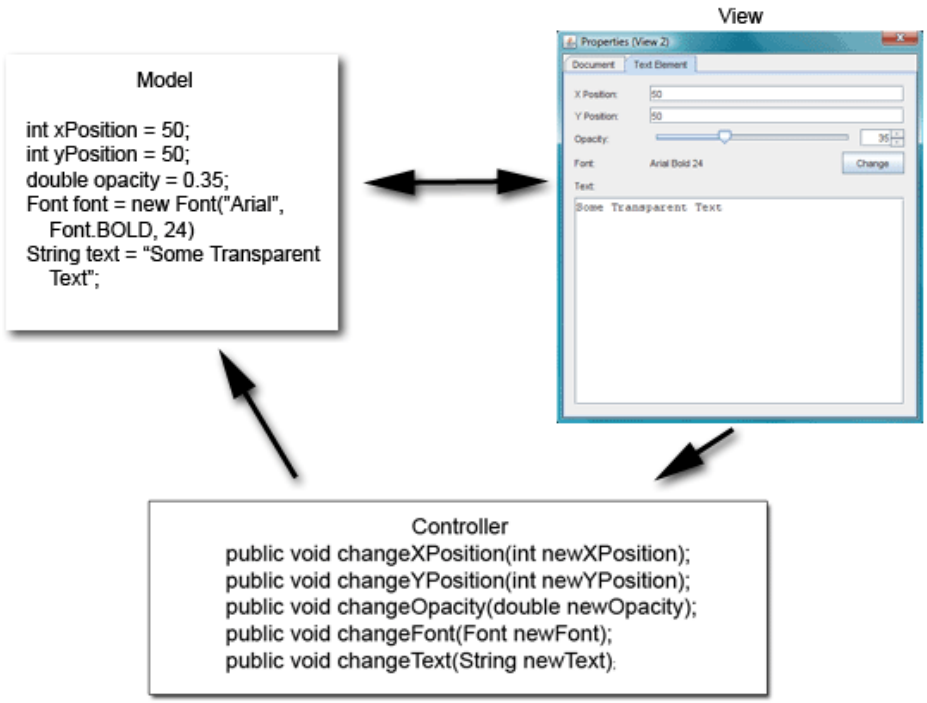

Figura 2.2: Exemplo de interação entre os componentes do padrão MVC em uma aplicação Java (Sun, 2007b).

Este tipo de arquitetura é formada pelas seguintes camadas:

- **Camada de Apresentação (*Presentation Tier*)**: contém toda a interface gráfica e permite a interação com o usuário por meio dos serviços disponíveis ao usuário (sessões e entrada de dados, por exemplo);
- **Camada Lógica (*Logic Tier*)**: contém toda a lógica do negócio, bem como a lógica de transações;
- **Camada de Dados (*Data Tier*)**: contém os dados que são manipulados pela aplicação, bem como o acesso a dados, atualização e persistência destes.

A Figura 2.3 apresenta uma ilustração gráfica da arquitetura em 3 camadas.

Fazendo uma breve comparação entre uma arquitetura em 3 camadas e uma arquitetura MVC por meio das figuras 2.3 e 2.1, respectivamente, percebemos que a primeira pode ser considerada linear, uma vez que a camada de apresentação não se comunica diretamente com a camada de dados, passando sempre pela camada lógica. Já as arquiteturas que seguem o padrão MVC são consideradas triangulares, uma vez que a Visão dispara eventos ao Controlador que por sua vez atualiza o Modelo, porém a Visão busca os dados diretamente do Modelo para exibí-los. Componentes de interface gráfica desenvolvidos sob o padrão MVC são comumente usados em aplicações com arquiteturas em 3 camadas.

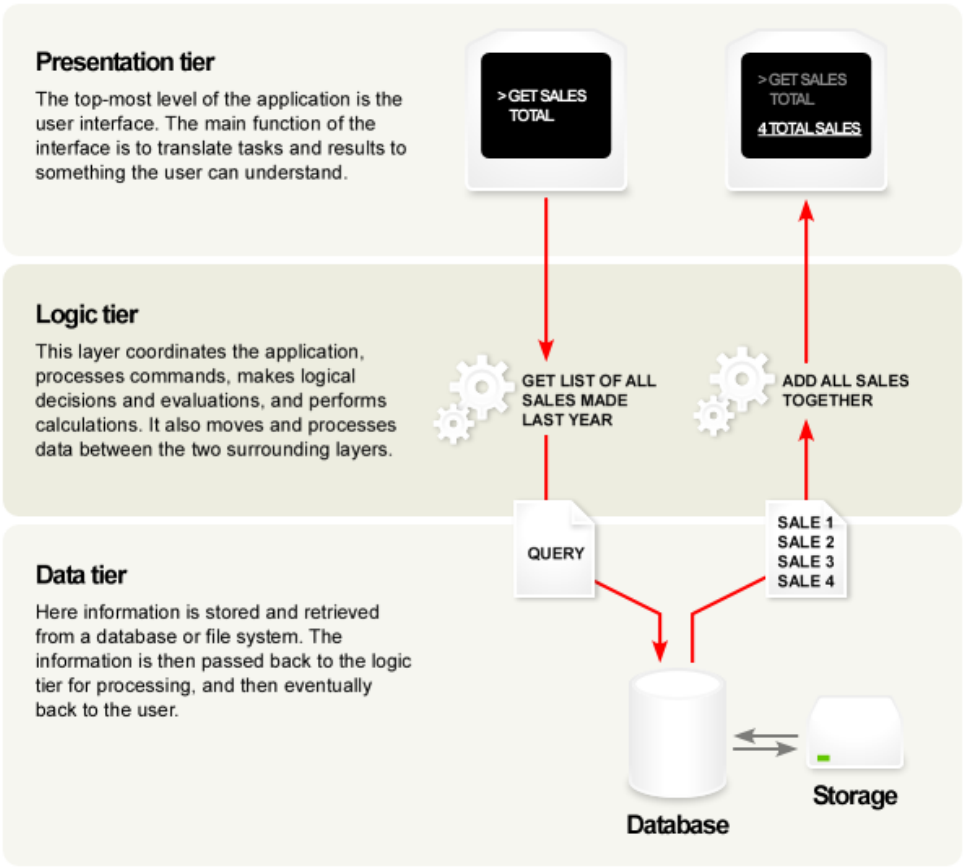

Figura 2.3: Representação gráfica de uma arquitetura em 3 camadas (Reese, 2000).

2.3 Processamento em Servidores Web

Servidores Web são aplicações que disponibilizam conteúdos e serviços para que aplicações Web possam acessar de forma distribuída e concorrente. Além dessas características, estes servidores devem ser providos de funcionalidades tais como: autenticação, autorização e controle de acesso, direcionamento de chamadas, filtro de dados de entrada e saída, segurança, alta performance e escalabilidade.

Entre os mais conhecidos servidores Web estão o *Apache HTTP Server* (Apache, 2007b), o *HP Web Server Suite* (HP, 2007) e o *IBM HTTP Server* (IBM, 2007).

O servidor Apache é amplamente o mais utilizado por não ser software proprietário e também pela estabilidade já alcançada em anos de desenvolvimento e recursos oferecidos. O Apache permite a instalação de módulos individuais que dão suporte ao processamento de dados enviados pelos clientes em uma grande variedade de tecnologias, dentre elas: *JavaServer Pages* (JSP) (Sun, 2007d), *ActiveServer Pages* (ASP) (Apache-ASP, 2007; Microsoft, 2007b), *Hypertext Preprocessor* (PHP) (PHP, 2007), *Common Gateway Interface* (CGI) (W3C, 2007) e Perl (Perl, 2007).

Os servidores Web tratam o processamento como uma seqüência de passos:

1. Recebem uma requisição HTTP do cliente;
2. Identificam o recurso desejado (imagem, página HTML, entre outros);
3. localizam o recurso e verificam a sua disponibilidade;
4. Se o recurso estiver disponível, este é enviado ao cliente que por sua vez exibe o conteúdo ao usuário. Caso contrário, um erro é gerado e enviado ao cliente.

Esse é o esquema de funcionamento padrão para servidores Web que servem somente conteúdo estático, ou seja, páginas HTML e derivadas e arquivos diversos como imagens, arquivos PDF, entre outros. A Figura 2.4 apresenta um esquema de processamento em um servidor Web.

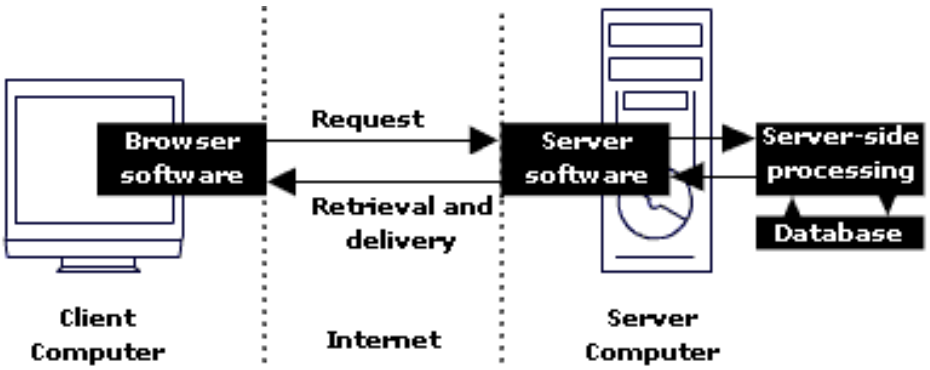

Figura 2.4: Esquema de Processamento em um Servidor Web.

Nesta figura podemos observar que o cliente (*Client Computer*) envia uma requisição (*Request*) (passo 1) para o servidor Web (*Server Computer*) que por sua vez processa o pedido (passos 2 e 3) e devolve uma resposta (*Response*) ao cliente (passo 4).

Porém, no caso de servidores Web que disponibilizam recursos dinâmicos como, por exemplo, páginas JSP, PHP e ASP, é adicionado um passo entre os passos 2 e 3. O servidor precisa identificar que o recurso requerido é um conteúdo dinâmico. No caso do Apache, isso é feito por meio da extensão do arquivo, por exemplo JSP. Assim, o servidor redireciona o pedido do recurso para o módulo responsável por interpretar o conteúdo de tal recurso. No Apache, é possível configurar os módulos para redirecionar o pedido ao *Container Web* Tomcat (Apache, 2007a) para páginas JSP. O *Container Web* interpreta o conteúdo e gera um arquivo HTML. Dessa forma, o *Container Web* devolve ao servidor Web somente o arquivo HTML gerado e o servidor por sua vez envia tal arquivo ao cliente como resposta. Para o cliente todo esse processamento é transparente independentemente

do recurso ser dinâmico ou estático, pois o navegador só recebe conteúdo estático como páginas HTML e figuras.

O Tomcat é um *Container Web* que pode ser utilizado como um servidor Web para aplicações Java desenvolvidas usando as tecnologias JSP e Servlet (Sun, 2007e). Pelo fato de servir tanto páginas dinâmicas em Java quanto páginas estáticas o Tomcat pode ser considerado um servidor Web e pode ser utilizado sem o apoio de um servidor como o Apache, por exemplo. Páginas ASP podem ser tratadas pelo Tomcat com a ajuda de um *plug-in* para o ambiente de desenvolvimento *Mainsoft for Java* chamado *Grasshopper* (Microsoft, 2007a), equanto páginas PHP podem ser tratadas com a instalação de um *patch* para o Tomcat (Apache, 2007c). Além disso, o Tomcat ainda possibilita a utilização do Axis (Apache, 2007d) que é uma implementação SOAP (*Simple Object Access Protocol*) para o desenvolvimento de serviços Web (*Web Services*) em Java. Por todas essas características, o Tomcat é um dos *Containers Web* mais utilizados no mercado para aplicações Java, assim como o Apache para aplicações em geral.

2.4 Considerações Finais

A literatura atual apresenta várias arquiteturas e padrões Web com o objetivo de facilitar tanto o desenvolvimento quanto a manutenção de WebApps. Dentre os mais conhecidos estão os padrões MVC e a arquitetura em três camadas. O mercado de desenvolvimento Web tem adotado e consolidado o uso do MVC como base para o desenvolvimento de seus produtos de software. Vários são os *cases* de empresas que relatam a adoção destes padrões com sucesso e ainda propostas de padrões derivados destes mais conhecidos.

Conhecer estes padrões e arquiteturas contribui no entendimento e no desenvolvimento de aplicações Web com o objetivo de melhorar a manutenção e ainda entender melhor o papel de cada tecnologia utilizada no desenvolvimento. O próximo capítulo trata da tecnologia JSP que desempenha um papel importante no desenvolvimento de aplicações Web como um componente Visão do modelo MVC.

Assim como as arquiteturas e os padrões, os servidores Web têm papel fundamental no desenvolvimento de WebApps, pois é neles que acontece todo o processamento das requisições enviadas pelo cliente e a transformação de conteúdo dinâmico em estático. Entender o funcionamento de um servidor Web é fundamental para o desenvolvimento de WebApps. O mercado de desenvolvimentode aplicações Web tem aceitado e adotado alguns servidores que se tornaram mais populares ao longo dos anos como é o caso do Apache com fins gerais e o Tomcat para aplicações Java principalmente.

Neste capítulo foram apresentados os conceitos gerais sobre padrões e arquitetutas Web, bem como os fundamentos do processamento Web no lado servidor. O capítulo a seguir está focado na apresentação da tecnologia JSP já consolidada e bem aceita pelo mercado de desenvolvimento de WebApps assim como pela comuinidade acadêmica.

Capítulo

3

Java Server Pages (JSP): Processamento Java no Servidor

3.1 Considerações Iniciais

Como já visto no capítulo anterior, é possível desenvolver aplicações Web com uma variedade de tecnologias e servidores Web disponíveis no mercado. Cada uma destas tecnologias possui características particulares, assim como os servidores. Porém, o objetivo principal de todas elas é permitir a disponibilzação de conteúdos dinâmicos na Web aumentando, assim, a interatividade entre os usuários e os sistemas.

Com base nestes conceitos, este capítulo apresenta uma visão geral da tecnologia JSP com o objetivo de ilustrar os conceitos de processamento de dados no lado servidor provenientes de um cliente. Além disso, são apresentados os elementos que compõem uma página JSP, bem como o seu papel nos padrões e arquiteturas Web discutidas. Exemplos de códigos JSP são vistos ao longo da apresentação dos conceitos básicos de JSP.

3.2 Desenvolvimento de WebApps com Java e MVC

O desenvolvimento de aplicações Web com Java é realizado com base em duas tecnologias principais: Servlets e JSP. A primeira delas representa o Controlador no padrão MVC e é responsável por encapsular a lógica de negócio da aplicação. Um servlet nada mais é do que uma classe Java que fica à disposição no servidor Web e recebe e responde por requisições do tipo HTTP. Já JSP possui elementos que permitem o desenvolvimento de componentes que representam a Visão no padrão MVC, uam vez que as páginas construídas com esta tecnologia são páginas HTML com código Java embutido e delimitado por tags bem definidas.

Em uma aplicação baseada em MVC, a Visão (página JSP ou HTML) gera um evento (um clique em um botão de um formulário, por exemplo). Esse evento (conhecido como requisião HTTP) é enviado a um servidor ou *container* Web (Tomcat, por exemplo) que repassa os dados enviados a um Controlador (no caso um servlet). O servlet por encapsular a lógica de negócio processa a requisição e dependendo da situação muda o estado de um Modelo. Dessa forma, os papéis de cada uma destas tecnologias fica bem definido facilitando a correta aplicação de cada uma delas.

É muito comum desenvolvedores iniciantes trocarem os papéis destas duas tecnologias ou até mesmo fazerem uso de somente uma delas para o desenvolvimento de aplicações Web. Seria bastante árduo gerar conteúdo dinâmico em um servlet, uma vez que os servlets precisam instanciar objetos específicos para oferecer uma saída em HTML para o cliente. Da mesma forma aconteceria se a lógica de negócio fosse colocada inteiramente em trechos de códigos Java embutidos em uma página HTML, o que prejudicaria a legibilidade e a manutenção da aplicação.

A Figura 3.1 apresenta um esquema de um servidor para uma aplicação Web Java.

Na figura podemos perceber a existência de um cliente Web (*Web Client*) que pode ser um navegador, por exemplo, que envia requisições HTTP (*HTTP Request*) para o servidor. O servidor verifica se o recurso solicitado é um component Web (*Web Components*) como um JSP ou um servlet, ou um conteúdo estático (uma página HTML ou até mesmo uma imagem). Os componentes Web podem se comunicar com um Sistema Gerenciador de Banco de Dados (SGBD) ou um repositório peristente ou até mesmo com componentes JavaBeans (*JavaBeans Components*) (Sun, 2007c). Os componentes JavaBeans representam os Modelos no padrão MVC e, além disso, os objetos persistentes de uma aplicação Java. Dessa forma, é possível identificar o papel de cada elemento Java no padrão MVC. Os componentes Web são responsáveis por gerar uma resposta (*HTTP Response*) ao cliente.

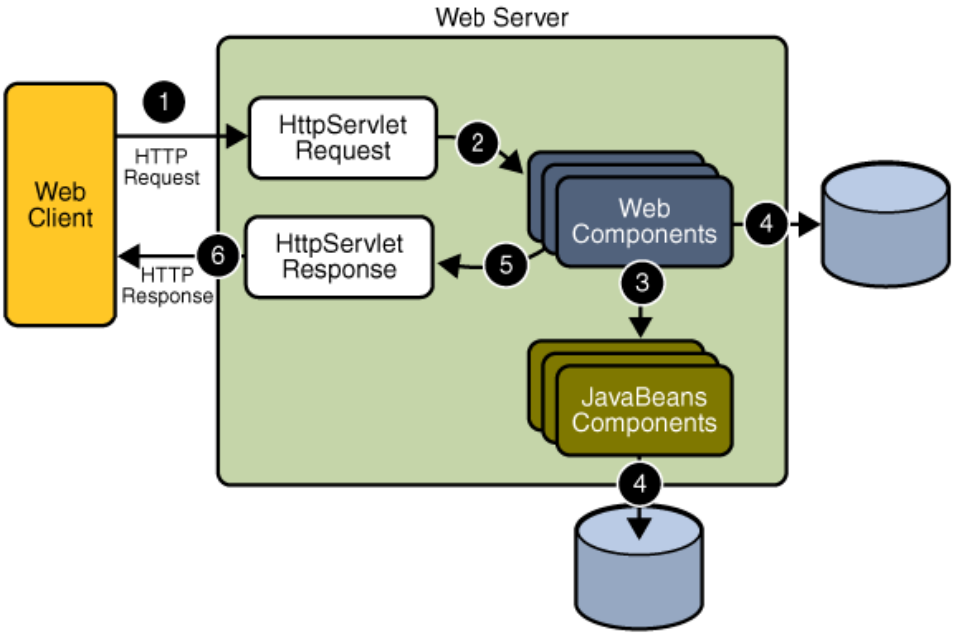

Figura 3.1: Esquema de um Servidor para uma Aplicação Web em Java (Sun, 2007f).

O desenvolvimento e a distribuição de aplicações Web em Java devem seguir uma estrutura pré-definida segundo a especificação da Sun. Para tanto, as páginas JSP e HTML, classes servlets e JavaBeans e bibliotecas de terceiros possuem lugares específicos em tal estrutura. A figura 3.2 apresenta a estrutura de uma aplicação Web.

Nesta figura temos uma pasta raiz chamada *Assembly Root* também conhecida como contexto de uma aplicação Web. Cada WebApp dá um nome diferente para essa pasta a qual representa o ponto de partida em um endereço URL para a busca de recursos em um servidor Web. Por exemplo, em `http://www.icmc.usp.br/coteia` a pasta *Assembly Root* é "coteia" e representa o contexo da aplicação. Essa pasta contém as páginas HTML e JSP, além de classes Applet. Tais arquivos são recursos disponíveis e acessíveis diretamente pelo cliente. No mesmo nível encontra-se a pasta *WEB-INF* que por sua vez é responsável por armazenar:

- **Arquivo web.xml**: arquivo de configuração onde são definidos parâmetros de inicialização, níveis de autenticação e autorização, mapeamento de servlets, entre outras coisas;
- **Arquivos .TLD**: são arquivos que descrevem a criação de tags personalizadas e que podem ser acessadas por páginas JSP.

Ainda na pasta *WEB-INF* temos a pasta *lib* que armazena todas as bibliotecas (arquivos JAR e ZIP) de terceiros e que podem ser referenciados por servlets e páginas JSP.

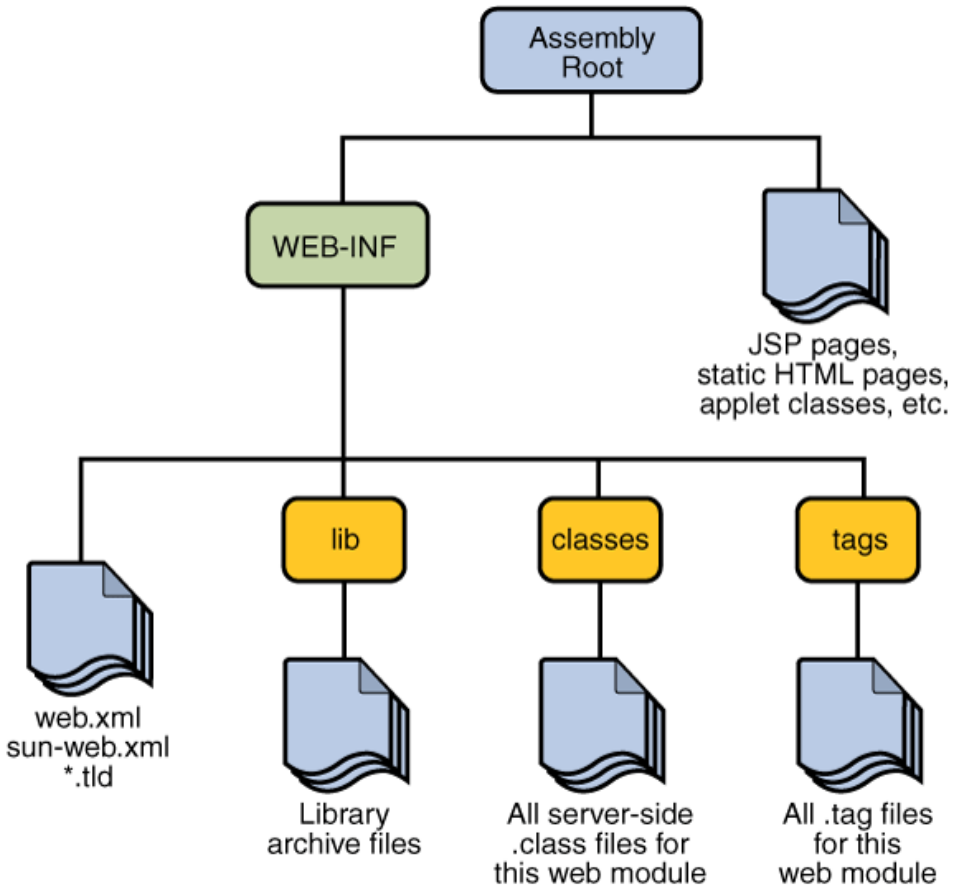

Figura 3.2: Estrutura de uma aplicação Web em um Servidor (Sun, 2007f).

A pasta *classes* armazena toda a estrutura de pacotes e classes implementadas para uma aplicação como, por exemplo as classes de servlets e JavaBeans. A pasta *tags* armazena todas as bibliotecas de tags personalizadas criadas para uma aplicação e que são descritas nos arquivos .TLD na pasta *WEB-INF*.

Os recursos definidos na pasta *WEB-INF* e hierarquicamente abaixo dela não podem ser acessados por um cliente de forma direta através de um endereço URL. Tais recursos só podem ser acessados por páginas JSP e classes servlets e JavaBeans por meio de um mapeamento definido no arquivo web.xml. Isso garante a integridade e a segurança na manipulação dos dados da aplicação. Caso contrário, os navegadores clientes poderiam acessar tais recursos e recuperar, por exemplo, uma arquivo .java e ver o seu conteúdo conhecendo, assim, a implementação da classe.

Desde o momento em que uma página JSP é escrita, esta passa por um ciclo de vida para que possa estar disponível ao cliente ou algum outro recurso da aplicação. Os itens a seguir apresentam cada uma das etapas do clico de vida das páginas JSP, desde a sua criação até a sua disponibilização (Basham et al., 2004):

1. **Tradução do JSP**: o servidor Web recupera o conteúdo de uma página JSP e traduz para uma classe servlet em Java (arquivo .java);

2. **Compilação do Servlet**: o servidor Web compila o servlet gerando um arquivo .class;

3. **Carga e Inicialização do Servlet**: o servlet é carregado e, então inicializado como um objeto Java que fica disponível e aguardando requisições HTTP em *threads* separadas para cada requisição. Em seguida chama o método *service()* do servlet.

Com base no ciclo de vida percebe-se que uma página JSP acaba sendo traduzida em um servlet. É possível imaginar, também, os tipos de elementos que uma página JSP pode conter. Na verdade existem vários tipos de elementos e cada um deles possui um local e uma forma correta de estar presente em uma página JSP. As seções a seguir apresentam os elementos básicos que uma página pode conter, bem como o local onde cada um deles pode aparecer.

3.3 Elementos Básicos do JSP

Páginas JSP podem ser formadas por vários tipos de elementos que nos permitem melhorar a apresentação dos dados ao cliente, bem como diminuir os esforços de manutenção. É importante conhecer os elementos do JSP para que seja possível otimizar o código implementado, uma vez que vários destes elementos permitem um aumento considerável de produtividade em projetos complexos.

A seguir são apresentados os elementos básicos da tecnologia JSP, dando destaque às diretivas, aos *scriptlets*, aos objetos implícitos, e às ações (Basham et al., 2004).

3.3.1 Diretivas (*Directives*)

Uma diretiva é uma maneira simples de colocar instruções em uma página JSP para que o servidor possa entender em tempo de tradução da página.

As diretivas são delimitadas pelos caracteres "<%@" e "%>" As duas principais diretivas são: *page* e *include*. Cada uma delas possui um conjunto de atributos e valores como apresentado a seguir.

A diretiva *page*

Esta diretiva permite informar ao servidor em tempo de tradução da página algumas propriedades específicas da página quando esta estiver disponível. A maneira como isso é realizado é por meio dos atributos da diretiva. A seguir é apresentada uma lista com os atributos da diretiva *page*.

- ***import***: este atributo permite fazer a importação dos pacotes para que seja possível acessar classes destes pacotes em uma página JSP.

  Exemplo de uso:

  ```
  <%@ page import=‘‘br.usp.icmc.hipermidia.beans.*, java.util.*’’ %>
  ```

  Nesse exemplo estão sendo importados dois pacotes: *br.usp.icmc.hipermidia.beans* e *java.util*. Uma página JSP pode ter várias diretivas *page* com o atributo *import*. Porém, é possível fazer o import de todos os pacotes em uma única diretiva separando os pacotes por uma vírgula como no exemplo.

- ***contentType***: define o tipo de conteúdo MIME para a resposta da página JSP. O valor padrão é “text/html”, ou seja, se o este atributo não for definido o tipo de conteúdo de resposta será este.

  Exemplo de uso:

  ```
  <%@ page contentType=‘‘xml’’ %>
  ```

  É posível definir outro tipo de conteúdo de resposta da página como XML neste exemplo.

- ***session***: define se os objetos de uma sessão podem ser acessados pela página JSP. O valor padrão é *true* e indica que os objetos da sessão podem ser acessados.

  Exemplo de uso:

  ```
  <%@ page session=‘‘true’’ %>
  ```

- ***errorPage***: define uma página JSP para onde será enviada qualquer exceção lançada na página JSP sendo executada.

  Exemplo de uso:

```
<%@ page errorPage=‘‘errors/error.jsp’’ %>
```

- ***isErrorPage***: define se uma página JSP pode exibir uma exceção enviada para ela pelo atributo *errorPage*. O valor padrão é *false* e indica que o objeto implícito *exception* não pode ser acessado. Objetos implícitos serão vistos na seção 3.3.4.

  Exemplo de uso:

  ```
  <%@ page isErrorPage=‘‘true’’ %>
  ```

  Nesse caso, é possível exibir a exceção ocorrida acessando o objeto *exception*.

- ***isELIgnored***: define se uma página JSP suporta ou não o uso linguagens de expressão (seção 3.3.5). O valor padrão é *false* e indica que a página suporta o uso destas linguagens.

  Exemplo de uso:

  ```
  <%@ page isELIgnored=‘‘true’’ %>
  ```

A diretiva *include*

A diretiva *include* é utilizada para incluir na página JSP corrente, em tempo de tradução, um texto ou código de outra página como, por exemplo, um menu ou uma barra de navegação comum à várias páginas de uma aplicação. Assim, é possível disponibilizar o código de tal menu em um arquivo e somente incluí-lo em páginas JSP que fazem uso deste sem ter que replicar tal código em cada página.

O único atributo desta diretiva é o *file* que define o arquivo que será incluído.

Exemplo de uso:

```
<%@ include file=‘‘menuBar.jsp’’ %>
```

3.3.2 *Scriptlets*: Comentários, Declarações e Expressões

Os *scriptlets* são códigos Java que podem ser colocados em qualquer parte de uma página JSP, mas que seguem a delimitação de caracteres especiais. Dentre os principais *scriptlets* temos: comentários, declarações e expressões.

Comentários

Os comentários em páginas JSP podem ser de dois tipos: comentários HTML e comentários JSP. Os comentários HTML são aqueles que o tradutor da página inclui na resposta ao cliente, porém como comentário. Dessa forma, se o usuário exibir o código-fonte da página HTML sendo exibida no navegador, este verá tais comentários.

Exemplo de comentário HTML:

```
<!-- comentário HTML -->
```

Já os comentários JSP são aqueles utilizados pelos desenvolvedores e, por isso, não aparecem no código-fonte de uma página HTML quando devolvolvida como resposta ao cliente.

Exemplo de comentário JSP:

```
<%-- comentário JSP --%>
```

Declarações

As declarações permitem a definição de variáveis e métodos em qualquer local de uma página JSP. É possível fazer dois tipos de declarações: locais e globais.

Exemplo de declaração de variável local:

```
<% int x = 10; %>
```

Nesse caso, a variável pode ser acessada em qualquer local abaixo dessa declaração, uma vez que ao traduzir está página, essa variável será local ao método *service()*. É possível declará-la de forma global (como atributo da classe servlet gerada na tradução) como mostra o exemplo a seguir:

Exemplo de declaração de variável global:

```
<%! int x = 10; %>
```

A única diferença é a "!" a mais na declaração da variável. Dessa forma, a variável pode ser acessada em qualquer local ou método da página JSP e não só localmente.

Expressões

As expressões são usadas para evitar o uso da instrução "*out.print("algum texto")*" que é usada para escrever alguma saída na página JSP. O *out* é um objeto implícito que será visto na seção 3.3.4.

As expressões são utilizadas combinando as tags "<%" e "%>" com o símbolo "=". Assim, é possível escrever uma saída para a página JSP como no exemplo a seguir:

```
<%= x %>
```

Nesse caso, será escrito o valor da variável x declarada no exemplo anterior. É possível também escrever o resultado da execução de métodos como, por exemplo:

```
<%= Contador.getCount() %>
```

As tags utilizadas para *sriptlets*, diretivas e expressões são bastante parecidas, portanto deve-se tomar muito cuidado ao utilizar cada uma delas. Um resumo de cada tag é apresentado a seguir:

- *Scriptlet*: <% %>
- Diretivas: <%@ %>
- Expressões: <%= %>

3.3.3 Exemplo de Página JSP com Diretivas e *Scriptlets*

O exemplo a seguir apresenta uma página JSP com as diretivas *page* e *include*, além de vários *scriptlets*.

```
<!-- página JSP exemplo01.jsp -->
<%@ page import="java.util.*" %>
<%@ page contentType="text/html;charset=ISO-8859-1" session="true"
isErrorPage="false" %>

<head> <title>Exemplo de Diretivas e Scriptlets</title> </head>

<html>
<%-- Declaracao de uma variavel global. --%>
<%! String texto = "Texto de uma variável global"; %>

<!-- Valor da variavel texto -->
Imprimindo o valor da váriavel Global "texto":  <%= texto %> <br><br>

<%-- Declaracao de uma variavel local. --%>
<% int cont = 0; %>

Imprimindo o valor da váriavel local "cont":  <%= cont++ %> <br>
Imprimindo o valor da váriavel local "cont":  <%= cont++ %> <br>

Incluindo o texo de uma página HTML:
<%@ include file="inclusao.html" %> <br><br>

<%-- Scriptlet --%>
<%
int z = 10;
z = z + cont;
out.print("Valor de x: " + z + " - Valor de cont: " + cont);
%>
</html>
```

É possível perceber no código anterior que o código Java mistura-se com o código HTML por meio do uso de um conjunto de tags específicas para cada elemento. Ainda, percebe-se o uso do objeto implícito *out* que permite a escrita em respostas ao cliente.

A seguir é apresentado o código HTML enviado ao navegador cliente como resposta ao processamento da página JSP pelo servidor Web.

```
<!-- página JSP exemplo01.jsp -->
<head>
<title>Exemplo de Diretivas e Scriptlets</title>
</head>

<html>
<!-- Valor da variavel texto -->
Imprimindo o valor da váriavel Global "texto":  Texto de uma variável
global<br><br>

Imprimindo o valor da váriavel local "cont":  0<br>
   Imprimindo o valor da váriavel local "cont":  1<br>
Imprimindo o valor da váriavel local "cont":  2<br><br>

Incluindo o texo de uma página HTML:
um texto qualquer a ser incluído....<br><br>

Valor de x: 13 - Valor de cont: 3
</html>
```

A Figura 3.3 apresenta o resultado do processamento da página JSP pelo servidor como resposta à requisição do cliente.

Figura 3.3: Resultado do Processamento da Página exemplo01.jsp Apresentado ao Cliente.

3.3.4 Objetos Implícitos (*Implicit Objects*)

Objetos implícitos são instâncias de classes fundamentais para a execução de um servlet, pois dentre várias funcionalidades permite a escrita de conteúdo de resposta ao cliente e gerenciamento de sessões. Para tanto, páginas JSP podem acessar tais objetos sem ter que instanciá-los, uma vez que no momento da tradução da página JSP em um servlet estes objetos são instanciados como variáveis locais ao método *service()*.

A tecnologia JSP possui nove objetos implícitos provenientes da API de servlets, sendo eles:

- ***application***: instância da classe *javax.servlet.ServletContext*. Representa o contexto da aplicação para componentes JSP e servlets. Permite acessar, por exemplo, parâmetros da aplicação definidos no arquivo web.xml;
- ***config***: instância da classe *javax.servlet.ServletConfig*. Armazena as informações de inicializaçao do servlet gerado após a tradução da página JSP;
- ***exception***: instância da classe *java.lang.Throwable*. Representa a exceção ocorrida e só é acessado em páginas cujo atributo *isErrorPage* da diretiva *page* é *true*;
- ***out***: instância da classe *javax.servlet.jsp.JspWriter*. Representa um *stream* de saída para a escrita de conteúdo na resposta enviada ao cliente;
- ***page***: instância da classe *java.lang.Object*. Representa a instância da página JSP no servlet gerado;
- ***pageContext***: instância da classe *javax.servlet.jsp.PageContext*. Representa o contexo da página JSP. Fornece uma API para a manipulação e armazenamento de objetos em vários escopos;
- ***request***: instância da classe *javax.servlet.http.HttpServletRequest*. Representa a requisição enviada à página JSP;
- ***response***: instância da classe *javax.servlet.http.HttpServletResponse*. Representa a resposta enviada ao cliente após o processamento da página JSP;
- ***session***: instância da classe *javax.servlet.http.HttpSession*. Representa a sessão atual do cliente.

O exemplo a seguir ilustra a utilização dos objetos implícitos em uma página JSP.

```
<!-- página JSP exemplo02.jsp -->
<%@ page contentType="text/html;charset=ISO-8859-1" session="true"
 isErrorPage="false" %>

<head> <title>Exemplo de Objetos Implícitos</title> </head>
<html>
<%-- Armazena e recupera um objeto na sessao. --%>
<% session.setAttribute("textoSessao","Texto gravado na sessão.");
String textoRecuperado = "Recuperando objeto da sessão:   " +
(String)session.getAttribute("textoSessao");
out.print(textoRecuperado); %>

<br><br>

<%-- Algumas informacoes a partir de objetos implicitos. --%>
<% out.print("Objeto implícito request -- Login recuperado da string
URL: " + request.getParameter("login") + "<br>");
out.print("Objeto implícito session -- ID da sessão: " +
session.getId() + "<br><br>");
out.print("Objeto implícito application -- Parâmetro de contexto: " +
 application.getInitParameter("parametroContexto") + "<br>");
out.print("Objeto implícito application -- Informações do servidor
Web: " +
 application.getServerInfo() + "<br>");
out.print("Objeto implícito application -- Nome do contexto descrito
 no arquivo web.xml: " + application.getServletContextName()
+ "<br><br>");
out.print("Objeto implícito config -- Nome do servlet gerado: " +
config.getServletName() + "<br>"); %>

<%-- Encerra uma sessão e destroi todos os objetos. Bastante usado
para logout. --%>
<br><br> Encerrando a sessão do usuário... <br><br>
<% session.invalidate();
out.println("sessão encerrada...."); %>
</html>
```

Neste exemplo podemos perceber o acesso na forma de *scriptlet* a vários objetos implícitos, sendo que cada um deles com sua prórpia API (Sun, 2007a).

A Figura 3.4 apresenta o resultado apresentado ao cliente após o processamento do código anterior.

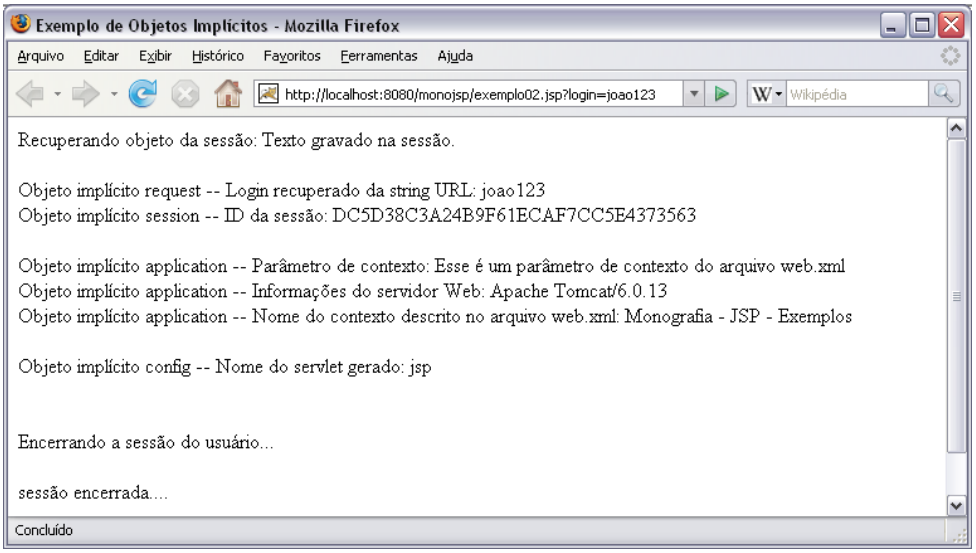

Figura 3.4: Resultado do Processamento da Página exemplo02.jsp Apresentado ao Cliente.

Percebe-se que a utilizaçãoo dos objetos implícitos em páginas JSP facilita ao desenvolvedor, pois este não precisa instanciá-los a cada nova página JSP criada, uma vez que tais objetos são criados no momento em que a página JSP é traduzida em um servlet.

3.3.5 Linguagem de Expressão (*Expression Language - EL*)

A EL é uma linguagem que tem por objetivo facilitar o acesso a objetos que seguem as regras de um componente JavaBean (Sun, 2007c) e seus atributos por meio de uma sintaxe própria e fácil de ser entendida e utilizada. Dessa forma, o desenvolvedor não precisa obrigatoriamento saber a sintaxe do Java. A EL já é habilitada por padrão em páginas JSP, porém é possível desabilitá-la definindo o atributo *isELIgnored* da diretiva *page* como *true*.

A grande vantagem da EL é o acesso direto aos objetos disponíveis nos vários escopos existents em uma aplicação Java Web sem ter que instanciar os objetos que permitem tal acesso nem usar *scriptlets*, o que torna o código de uma página JSP mais legível e de fácil manutenção. Além disso, objetos com ponteiros nulos (*null*) quando acessados não geram exceções, mas simplesmente não são exibidos como resposta ao cliente. A seguir temos

uma comparação entre um código *scriptlet* e um código EL. Ambos possuem o mesmo objetivo que é recuperar um objeto da sessão.

Código em *Scriptlet*:

```
<% String txt = (String)session.getAttribute("textoSessao"); %>
```

Código em EL:

```
${sessionScope.textoSessao}
```

Existe uma grande diferença entre os dois trechos de código acima. A primeira é que usando EL não é necessário usar as tags "<%" e "%>". A segunda é que em EL não é necessário chamar os métodos "get" do atributo que se deseja recuperar o valor. Isso só é possível em objetos que respeitam as regras de um componente JavaBean e, dessa forma, é referenciado somente o atributo. No caso do exemplo acima, a sessão armazena os objetos na forma de *Map*, ou seja, cada objeto é referenciado por um valor (um nome de identificação). A EL permite a busca por objetos em estruturas do tipo *Map* por meio da indicação direta do nome atribuído ao objeto. No exemplo acima foi utilizado o nome *textoSessao* para recuperar um objeto do tipo *String* armazenado na sessão.

No exemplo é possível perceber a utilização de um elemento chamado *sessionScope*. A EL, assim como em JSP, possui um conjunto de objetos implícitos e estes não possuem os mesmo nomes dos objetos implícitos de JSP (seção 3.3.4). A seguir são apresentados os objetos impícitos da EL:

- ***pageScope***: mapeia os objetos do escopo da página aos seus nomes;
- ***requestScope***: mapeia os objetos do escopo da requisição aos seus nomes;
- ***sessionScope***: mapeia os objetos do escopo da sessão aos seus nomes;
- ***applicationScope***: mapeia os objetos do escopo da aplicação aos seus nomes;
- ***param***: mapeia um nome de parâmetro da requisição para um valor;
- ***paramValues***: mapeia um nome de parâmetro da requisição para um *array* de valores;
- ***header***: mapeia um nome de cabeçalho para um valor;
- ***headerValues***: mapeia um nome de cabeçalho para um *array* de valores;
- ***cookie***: mapeia um nome de *cookie* para um valor;

- ***initParam***: mapeia um parâmetro de inicialização do contexto para um nome;
- ***pageContext***: é o único objeto implícito que não faz mapeamento, porém permite acesso direto aos atributos: *servletContext*, *session*, *request* e *response*.

O exempo a seguir apresenta o uso de EL em substituição a uma parte do código dá página exemplo02.jsp.

```
<!-- página JSP exemplo03.jsp -->
<%@ page contentType="text/html;charset=ISO-8859-1" session="true"
 isErrorPage="false" %>
<head>
<title>Exemplo de EL</title>
</head>

<html>
<%session.setAttribute("textoSessao", "Texto gravado na sessão.");%>

<%-- Recuperando o objeto armazenado na sessao. --%>
Recuperando objeto da sessão: ${sessionScope.textoSessao}
<br><br>

Login recuperado da string URL: ${param.login} <br>
ID da sessão: ${pageContext.session.id} <br><br>

Parâmetro de contexto: ${initParam.parametroContexto}
</html>
```

A Figura 3.5 mostra o conteúdo gerado e apresentado ao cliente após o processamento da página com EL.

Percebe-se, nesse exemplo, que o trecho de código em EL comparado ao trecho de código da página exemplo02.jsp, respectivo, fica mais legível e menor. Dessa forma, o desenvolvedor de páginas JSP implementa um código mais ”enxuto”, o que facilita a sua manutenção.

3.4 Considerações Finais

A demanda por aplicações Web vem crescendo a cada ano e o mercado de desenvolvimento tem se mostrado bastante competivivo possibilitando, assim, a proposta de novas

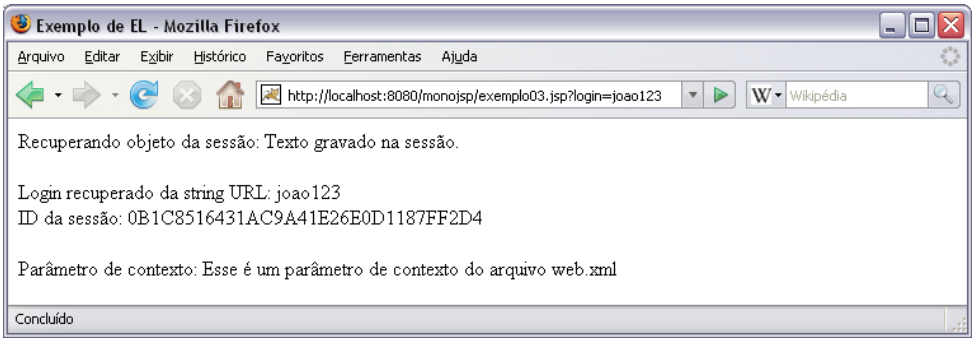

Figura 3.5: Resultado do Processamento da Página exemplo03.jsp Apresentado ao Cliente.

tecnologias. A tecnologia JSP vem sendo utilizada de forma já bastante consolidada no mercado tendo muita aceitação. A cada nova versão da tecnologia, novos recursos vão sendo incorporados, bem como alguns vão sendo melhorados.

Neste capítulo foram vistos os fundamentos básicos da tecnologia JSP com o objetivo de contextualizar o leitor e apresentar os recursos que a tecnologia dispõe para o desenvolvimento de WebApps. Percebe-se que JSP exige um pouco de conhecimento dos conceitos básicos de Orientação a Objetos e da sintaxe da linguagem Java para a criação de páginas por meio de *scriptlets*. Já o uso de diretivas e de linguagens de expressão não necessitam de tal conhecimento facilitando, dessa forma, o desenvolvimento e tornando o código implementado mais simples de se realizar manutenções.

Foi feita, também, uma exploração prática básica da utilização destes conceitos, porém um estudo mais aprofundado sobre a tecnologia e padrões de desenvolvimento Web devem ser realizados com o objetivo de fornecer um treinamento adequado ao desenvolvedor, uma vez que este trabalho está focado na apresentação geral dos conceitos sobre a tecnologia JSP.

Capítulo

4

Conclusões

Este trabalho apresentou vários conceitos sobre desenvolvimento de aplicações Web com destaque ao padrão MVC e a arquitetura em três camadas, assim como processamento no lado servidor. MVC e arquitetura em três camadas são conceitos já consolidados tanto na literatura quanto na indústria no desenvolvimento de aplicações Web visando diminuir os esforços em termos de manutenção.

As tecnologias existentes para o desenvolvimento deste tipo de aplicação têm permitido maior interação e disponibilização de conteúdo na Internet facilitando, assim, a divulgação de seus produtos e serviços aos mais variados tipos de clientes e suas localizações geográficas.

A tecnologia JSP vem sendo bastante utilizada para o desenvolvimento de WebApps tendo um papel fundamental no padrão MVC: a visão. Outras tecnologias Java existentes cooperam com a tecnologia JSP como é o caso dos servlets que possuem o papel do controlador no padrão MVC e os JavaBeans que representam os modelos.

Este trabalho apresentou os conceitos básicos da tecnologia JSP bem como exemplos práticos de sua utilização e sintaxe dos seus principais elementos, dentre eles: objetos implícitos, *scriptlets* e linguagem de expressão. Tais elementos são imprescindíveis para o desenvolvimento utilizando JSP e, também, para facilitar a manutenção das aplicações. Além disso, foi apresentada uma discussão a respeito de JSP e as tecnologias relacionadas, bem como os padrões e arquiteturas Web.

Este trabalho serve como um documento básico sobre os conceitos relacionados ao desenvolvimento de aplicações Web, bem como um tutorial básico sobre a tecnologia JSP e seus principais elementos. Conceitos mais avançados e específicos sobre JSP e as tecnologias relacionadas não foram apresentados neste trabalho. Dessa forma, o leitor deve buscar outros documentos para tal finalidade. Assim, o intuito deste trabalho foi contextualizar o leitor sobre o desenvolvimento de WebApps e fornecer uma noção prática e básica sobre o assunto.

Referências

Apache Apache Tomcat. 2007a.
Disponível em `http://tomcat.apache.org` (Acessado em 26/06/2007)

Apache The Apache HTTP Server Project. 2007b.
Disponível em `http://httpd.apache.org` (Acessado em 26/06/2007)

Apache Using PHP. 2007c.
Disponível em `http://wiki.apache.org/tomcat/UsingPhp` (Acessado em 26/06/2007)

Apache WebServices - Axis. 2007d.
Disponível em `http://ws.apache.org/axis/java/install.html#Tomcat4.xAndJava1.4` (Acessado em 26/06/2007)

Apache-ASP Web Applications with Apache and Perl. 2007.
Disponível em `http://www.apache-asp.org` (Acessado em 26/06/2007)

Basham, B.; Sierra, K.; Bates, B. *Head first servlets and jsp - passing the sun certified web component developer exam.* O'Reilly, 2004.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern-oriented software architecture: a system of patterns.* New York, NY, USA: John Wiley & Sons, Inc., 1996.

Hendrickson, E.; Fowler, M. The software engineering of internet software: Guest editors' introduction. *IEEE Softw.*, v. 19, n. 2, p. 23–24, 2002.

HP Web Servers for HP-UX 11i - Overview and Features. 2007.
Disponível em `http://www.hp.com/products1/unix/webservers/index.html` (Acessado em 26/06/2007)

IBM IBM HTTP Server - Product Overview. 2007.
Disponível em `http://www-306.ibm.com/software/webservers/httpservers` (Acessado em 26/06/2007)

Kerer, C.; Kirda, E. Layout, content and logic separation in web engineering. In: *Web Engineering, Software Engineering and Web Application Development*, London, UK: Springer-Verlag, 2001, p. 135–147.

Microsoft Microsoft Mainsoft Grasshopper. 2007a.
Disponível em `http://dev.mainsoft.com/Default.aspx?tabid=130` (Acessado em 26/06/2007)

Microsoft The Official Microsoft ASP.NET 2.0 Site. 2007b.
Disponível em `http://www.asp.net` (Acessado em 26/06/2007)

Mozilla Home of the Firefox web browser and Thunderbird email client. 2007.
Disponível em `http://www.mozilla.com` (Acessado em 26/06/2007)

Perl The Perl Directory. 2007.
Disponível em `http://www.perl.org` (Acessado em 26/06/2007)

PHP PHP: Hypertext Preprocessor. 2007.
Disponível em `http://www.php.net` (Acessado em 26/06/2007)

Pressman, R. S. *Software engineering: A practitioner's approach*. McGraw-Hill Higher Education, 2006.

Reese, G. *Database programming with jdbc and java*. O'Reilly, 2000.

Sadoski, D.; Comella-Dorda, S. Three tier software architectures. 1997.
Disponível em `http://www.sei.cmu.edu/str/descriptions/threetier.html` (Acessado em 25/06/2007)

Sun Java EE 5 SDK API. 2007a.
Disponível em `http://java.sun.com/javaee/5/docs/api` (Acessado em 28/06/2007)

Sun Java SE Application Design with MVC. 2007b.
Disponível em `http://java.sun.com/developer/technicalArticles/javase/mvc/index.html` (Acessado em 26/06/2007)

Sun JavaBeans. 2007c.
Disponível em `http://java.sun.com/products/javabeans` (Acessado em 28/06/2007)

Sun JavaServer Pages Technology. 2007d.
Disponível em `http://java.sun.com/products/jsp` (Acessado em 26/06/2007)

Sun Servlet Technology. 2007e.
Disponível em `http://java.sun.com/products/servlet` (Acessado em 26/06/2007)

Sun The Java EE Tutorial. 2007f.
Disponível em `http://java.sun.com/javaee/5/docs/tutorial/doc` (Acessado em 27/06/2007)

W3C CGI - Common Gateway Interface. 2007.
Disponível em `http://www.w3.org/CGI` (Acessado em 26/06/2007)