Faculdades Santo Agostinho - FASA

**Faculdade de Ciências Exatas e Tecnológicas – FACET**

Curso de Sistemas de Informação

**ANE CRISTINE SILVA FREITAS**

**ARMANDO JÚNIOR**

**HENSLEY GOMES**

**JORDANA CAYRES GOMES**

**MICHELLE PÂMELA**

**PABLO ZITO**

**JSP**

**Montes Claros**

**2011**

**JSP**

JSP - é uma tecnologia baseada em Java que simplifica o processo de desenvolvimento de sites da web dinâmicos.Com JSP, os designers da web e programadores podem rapidamente incorporar elementos dinâmicos em páginas da web, utilizando Java embutido e algumas tags de marcação simples.Estas tag(s) fornecem ao designer de HTML um meio de acessar dados e lógica de negócio armazenados em objetos Java sem ter que dominar as complexidades do desenvolvimento de aplicações.

Comsidere a JSP, como um tipo de linguagem de criação de script no servidor, embora, como você verá mais tarde, ela opere de maneira bem diferente por detrás dos panos.JavaServer Pages são arquivos textos,normalmente com a extensão ".jsp"(mas é normalmente possível configurar o container JSP para reconhecer extensões de arquivos adicionais ou alternativas.), que substituem as páginas HTML tradicionais.Os arquivos JSP contêm HTML junto com o código embutido que permitem que o designer de páginas acessem dados do código Java rodando no servidor.Quando a página é solicitada por um usuário e processada pelo servidor HTTP (HyperText Transport Protocol), a parte HTML da página é então transmitida.No entanto, as partes de código das páginas são executadas no momento em que a solicitação é recebida e o conteúdo dinâmico gerado por este código é unido na página, antes de ser enviado para o usuário.Isto propicia uma separação dos aspectos de apresentação HTML da página, da lógica de programação contida no código.

Incorporar conteúdo dinâmico deve no final das contas envolver algum tipo de programação para descrever como aquele conteúdo é gerado.No entanto o código do programa tende a ser caro para criar e manter, logo, minimizar a necessidade de programação é freqüentemente um objetivo desejável.Combinar este objetivo com o objetivo da SUN por um suporte robusto e repleto de recursos para Java no servidor, um sistema modelo baseado em Java, JSP, foi um resultado natural.

Quem conhece servlets verá que o JSP não oferece nada que você não possa conseguir com os servlets puros. O JSP, entretanto, oferece a vantagem de ser facilmente codificado, facilitando assim a elaboração e manutenção de uma aplicação. Além disso, essa tecnologia permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), facilitando o desenvolvimento de aplicações mais robustas, onde programador e designer podem trabalhar no mesmo projeto, mas de forma independente.

JSP é um tanto quanto híbrido entre os sistemas de modelo, porque suporta dois estilos diferentes para adicionar conteúdo dinâmico a páginas da web.Assim com o ASP, PHP e SSJS (Server-Side Javascript), scripts podem ser embutidos em páginas JSP contendo o código de programação tipicamente Java.(O que quero dizer é que a especificação JSP permite a possibilidade de diferentes linguagens de criação de scripts).

JSP oferece diversos benefícios como um sistema para geração de conteúdo dinâmico.Primeiramente, sendo uma tecnologia baseada em Java, ela se aproveita de todas as características e vantagens desta linguagem como orientação a objetos, herança, encapsulamento, tratamento de exceções e gerenciamento de memória automática (Garbage Colection) conduzindo assim a um código mais robusto e flexível.

Pelo fato de haver uma API padrão publicada para JSP, e pelo fato do bytecode Java compilado ser portável através de todas as plataformas que aceitam uma JVM (Java Virtual Machine),o uso de JSP não restringe ao uso de uma plataforma de hardware ,sistema operacional ou software servidor específico.Se uma mudança de qualquer um destes componentes se tornar necessária, todas as páginas JSP e classes de Java associadas podem ser migradas no estado em que se encontram.

Como um elemento essencial de sites da web ricos de recursos e aplicações baseadas na web, a geração de conteúdo dinâmico continua sendo foco de muita atenção na comunidade de programação web.A tecnologia JSP tem agora um grande papel na evolução contínua desta tecnologia e esperamos ver a próxima geração de ferramentas, serviços e diversões on-line que serão habilitadas pela especificação JavaServer Pages, um exemplo disso é o novo conceito para desenvolvimento gráfico das interfaces,o JavaServer Faces, e o surgimento de frameworks que auxiliarão no deselvolvimento do projeto.

**PROTOCOLO HTTP**

O protocolo HTTP (HyperText Transfer Protocol) é o protocolo mais utilizado na Internet desde 1990. A versão 0.9 destinava-se unicamente a transferir dados na Internet (em especial páginas Web escritas em HTML). A versão 1.0 do protocolo (a mais utilizada) permite doravante transferir mensagens com cabeçalhos que descrevem o conteúdo da mensagem utilizando uma codificação de tipo MIMO. O protocolo HTTP tem como objetivo permitir uma transferência de ficheiros (essencialmente no formato HTML) localizados graças a uma cadeia de caracteres chamada URL entre um navegador (o cliente) e um servidor Web (chamado de resto httpd nas máquinas UNIX).

**GET E POST**

O método **GET** é usado quando queremos pesquisar ou passar dados para outra página usando a URL da página.

O método GET tem como objetivo enviar uma requisição por um recurso. As informações necessárias para a obtenção do recurso (como informações digitadas em formulários HTML) são adicionadas à URL e, por conseqüência, não são permitidos caracteres inválidos na formação de URLs como:

·         Espaços em Branco.

·         Caracteres Especiais.

O método GET possui a vantagem dos servidores Web poderem assumir que a requisição pode ser repetida, sendo possível adicionar à URL resultante de uma pesquisa, mas o GET tem como desvantagem que as informações passadas via GET não pode ser muito longas, o número de caracteres permitidos é por volta de 2K.

O método **POST** é usado quando queremos enviar dados a serem gravados em um banco de dados ou uma pesquisa cujos dados sejam grandes o suficiente para não caber na URL da página.

JÁ as requisições POST, a princípio podem ter tamanho ilimitado. No entanto, elas não são idempotente, o que as tornam ideais para formulários onde os usuários precisam digitar informações confidenciais, como número de carão de crédito. Desta forma o usuário é obrigado a digitar a informação toda vez que for enviar a requisição, não sendo possível registrar a requisição em um *bookmark.*

**ARQUITETURA REST**

Um protocolo cliente/servidor sem estado: cada mensagem HTTP contém toda a informação necessária para compreender o pedido. Como resultado, nem o cliente e nem o servidor necessitam gravar nenhum estado das comunicações entre mensagens. Na prática, muitas aplicações baseadas em HTTP utilizam cookies e outros mecanismos para manter o estado da sessão (algumas destas práticas, como a reescrita de [URLs](http://pt.wikipedia.org/wiki/URL" \o "URL" \t "_blank), não são permitidas pela regra do REST).

Um conjunto de operações bem definidas que se aplicam a todos os recursos de informação: HTTP em si define um pequeno conjunto de operações, as mais importantes são POST, GET, PUT e DELETE. Com freqüência estas operações são combinadas com operações [CRUD](http://pt.wikipedia.org/wiki/CRUD) para a persistência de dados, onde POST não se encaixa exatamente neste esquema.

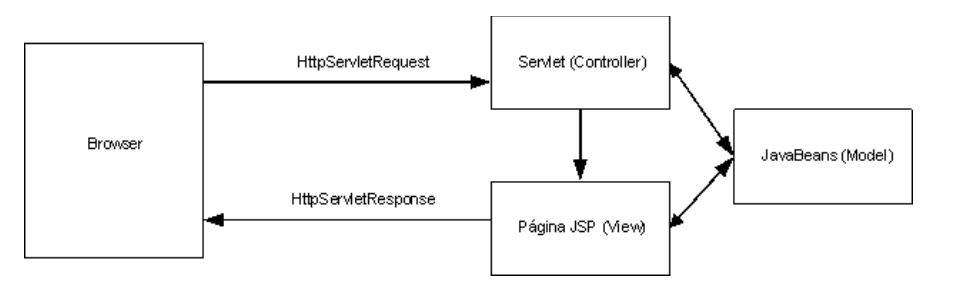
Uma sintaxe universal para identificar os recursos. No sistema REST, cada recurso é unicamente direcionado através da sua [URI](http://pt.wikipedia.org/wiki/URI).

O uso de hipermídia, tanto para a informação da aplicação como para as transições de estado da aplicação: a representação deste estado em um sistema REST são tipicamente [HTML](http://pt.wikipedia.org/wiki/HTML) ou [XML](http://pt.wikipedia.org/wiki/XML). Como resultado disto, é possível navegar com um recurso REST a muitos outros, simplesmente seguindo ligações sem requerer o uso de registros ou outra infraestrutura adicional.

**ARQUITETURA BÁSICA DO MODELO MVC (MODEL – VIEW - CONTROLLER).**

A arquitetura básica do modelo MVC, ou Model-View-Controller, se vale do uso de Servlets, JavaBeans e páginas JSP: os Servlets controlam as requisições recebidas (Controller), os JavaBeans implemejntam a lógica da aplicação (Model), e as páginas JSP se encarregam da apresentação do resultado (View).

Podemos representar melhor essa arquitetura através da seguinte forma:



Toda vez que uma requisição é recebida, o Servlet de controle repassa a requisição para a página JSP responsável pela apresentação da resposta, sendo que o JavaBeans são utilizados pela página JSP para obter os dados dinâmicos da aplicação.

**CONTROLADORES, VISÕES E MODELOS**

O Controlador, Visão, Modelo,  é tipicamente utilizado para a criação de aplicações web e não só CodeIgniter o implementa, como também outra série de frameworks de desenvolvimento web, em PHP ou outras linguagens. É interessante porque separa em vários grupos as complexidades das diferentes partes que compõem uma página web, como a vista e a lógica, assim como o acesso à base de dados.

Talvez o que mais nos force a mudar nossos hábitos de programação em PHP seja o fato de termos que basear nossos scripts neste modelo de programação, que seguramente resultará uma novidade para a maioria dos leitores deste manual, porque fixa um novo estilo de desenvolvimento de aplicações, que nos obriga a separar código fonte segundo seu âmbito. Entretanto, como dizíamos anteriormente, estes novos costumes de codificação também ajudarão nossos programas a serem melhores e a desfrutarem de várias vantagens como ser mais organizados, extensíveis y entendíveis por outros desenvolvedores, reutilizáveis, de mais fácil manutenção, etc.

Para aqueles que não o conhecem, o Modelo - Visão - Controlador (em inglês Model - View - Controller) é um padrão de desenvolvimento ou um estilo de arquitetura de software que separa o código fonte das aplicações em três grupos:

**CONTROLADOR**

O controlador poderíamos dizer que é a parte mais importante, porque funciona como link entre o modelo, a visão e qualquer outro recurso que se tenha que processar no servidor para gerar a página web. Em resumo, no controlador salvamos a lógica de nossas páginas e realizamos todas as ações que sejam necessárias para gerá-las, ajudados pelo modelo ou pela visão.

Durante o desenvolvimento com CodeIgniter será muito recomendável seguir as normas do design Modelo - Visão - Controlador (MVC), mas realmente o framework é bastante flexível e permite que, se desejarmos, não sigamos o desenvolvimento atendendo a tal arquitetura. Neste caso, poderíamos ter simplesmente controladores e dentro deles realizar todas as ações de acesso à base de dados diretamente, sem fazer chamadas ao modelo, ou escrever texto na saída sem utilizar as visões. Obviamente, esta opção não aproveitará as vantagens de separação de código entre apresentação, lógica e modelo de base de dados, porém se virmos que nos resulta muito complexo o MVC, podemos deixá-lo de lado.

No caso de que não utilizemos os modelos, não ocorrerá nenhum efeito negativo no desempenho do framework, porém, no caso das visões, se não as utilizamos e escrevemos saída diretamente desde o controlador, como por exemplo, com sentenças echo de PHP no código dos controladores, perderemos algumas das vantagens que CodeIgniter realiza por nós para processar a saída antes de enviá-la ao usuário. Isto será detalhado mais adiante.

**VISÕES**

A visão codifica e mantém a apresentação final de nossa aplicação com vista ao usuário. Ou seja, na visão colocaremos todo o código HTML, CSS, Javascript, etc. que se deve gerar para produzir a página como queremos que seja vista pelo usuário. Na prática, a visão não só serve para produzir páginas web, como também qualquer outra saída que queiramos enviar ao usuário, em formatos ou linguagens diferentes, como podem ser feeds RSS, arquivos JSON, XML, etc.

**MODELOS:**

Todo o código que tem que ver com o acesso à base de dados. No modelo manteremos encapsulada a complexidade de nossa base de dados e apenas criaremos funções para receber, inserir, atualizar ou apagar informação de nossas tabelas. Ao se manterem todas as chamadas à base de dados em um mesmo código, desde outras partes do programa poderemos invocar as funções que necessitemos do modelo e este se encarregará de processá-las. No modelo nos poderão preocupar coisas como o tipo de base de dados com a que trabalhamos, ou as tabelas e suas relações, mas desde as outras partes do programa simplesmente chamaremos as funções do modelo sem nos importar com o que ele tem que fazer para conseguir realizar as ações invocadas.

**FRAMEWORK JSP**

**SERVLETS E HTTPSERVLET**

A classe HttpServlet é uma classe abstrata (que, obviamente, não pode ser instanciada, ou seja, não podemos criar objetos desta classe) que extende a classe GenericServlet. Sempre que queremos criar um servlet que atende requisições HTTP, deveremos extender a classe HttpServlet. Veja sua posição na hierarquia de classes Java:

java.lang.Object  
 javax.servlet.GenericServlet  
 javax.servlet.http.HttpServlet

Esta classe extende as seguintes interfaces: Serializable, Servlet e ServletConfig.

Este é o método que geralmente sobrescrevemos quando queremos fornecer a funcionalidade desejada ao nosso servlet. Tenha em mente que, quando estamos tratando requisições específicas de GET ou POST, é recomendável deixar o método service() em paz e sobrescrever os métodos doGet() e doPost().

O trecho de código abaixo mostra como sobrescrever o método doGet() em um servlet (fique atento ao fato de que nossa classe servlet extende a classe HttpServlet):

package estudos;  
  
import java.io.\*;  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
  
public class EstudosServlet extends HttpServlet{  
 public void doGet(HttpServletRequest req,   
 HttpServletResponse res) throws ServletException,   
 IOException{  
   
 PrintWriter pw = res.getWriter();  
 pw.println("Recebi uma requisição GET");  
 }  
}

Principais métodos: doGet() e doPost()

Servlets são classes Java, desenvolvidas de acordo com uma estrutura bem definida que quando instaladas e configuradas em um Servidor que implemente um Servlet Container, podem tratar requisições recebidas de clientes Web, como por exemplo os Browsers (Internet Explorer® e Mozilla Firefox®).

Ao receber uma requisição, um Servlet pode capturar os parâmetros desta requisição, efetuar qualquer processamento inerente a uma classe Java, e devolver uma página HTML. A listagem abaixo, retorna uma página para o cliente WEB, com o endereço IP do servidor.

package org.computer.br;

import java.io.\*;

import java.net.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class EnderecoIpRemoto extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse

response)throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet EnderecoIpRemoto</title>");

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

out.println("<body>");

out.println("<h1>Endereco ip Remoto:" + request.getRemoteAddr() +

"</h1>");

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

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

out.close();

}

protected void doGet(HttpServletRequest request, HttpServletResponse

response)throws ServletException, IOException {

processRequest(request, response);

}

protected void doPost(HttpServletRequest request, HttpServletResponse

response)throws ServletException, IOException {

processRequest(request, response);

}

public String getServletInfo() {

return "Short description";

}

}

**CLASSES HttpServletRequest e HttpServletResponse**

Fornece uma classe abstrata para ser uma subclasse de criar um servlet HTTP apropriado para um site. Uma subclasse de HttpServlet deve substituir pelo menos um método, normalmente um desses:

doGet , se o servlet aceita solicitações HTTP GET

doPost , para solicitações HTTP POST

doPut , para solicitações HTTP PUT

doDelete , DELETE para pedidos HTTP

init e destroy , para gerir os recursos que são detidos para a vida do servlet

getServletInfo , que o servlet utiliza para fornecer informações sobre si próprio

Quase não há razão para substituir o service método. service lida com requisições HTTP padrão, enviando-lhes os métodos de manipulação para cada tipo de solicitação HTTP (o do métodos XXX listadas acima).

Da mesma forma, não há quase nenhuma razão para substituir o doOptions e doTrace métodos.

Servlets normalmente são executados em servidores multithreaded, portanto estar ciente de que um servlet deve tratar solicitações simultâneas e ter cuidado para sincronizar o acesso a recursos compartilhados. recursos compartilhados incluem dados em memória, como instância ou classe variáveis ​​e objetos externos, como arquivos, conexões de banco de dados e conexões de rede. Consulte o Tutorial Java em programação multithread para obter mais informações sobre o manuseio múltiplos threads em um programa em Java.

**CLASSES RequestDispatcher**

classe RequestDispatcher é usado principalmente para "passar" a solicitação atual para outro programa (servlet) e, portanto, permite "encadeamento" dos programas. A RequestDispatcher contém principalmente dois métodos include () e forward (). Incluir) método (inclui a resposta de um outro programa, enquanto método () encaminha o pedido para a frente do programa em curso para outro.

forward ()

O atacante () de frente RequestDispatcher e objetos de pedido de resposta ServletRequest ServletResponse e interface, respectivamente, para o caminho especificado na getRequestDispatcher (String caminho). A resposta é enviada de volta para o cliente, portanto, o cliente não sabe sobre esta mudança de recurso no servidor. Este método ajuda para a comunicação entre os recursos do servidor (servlet para servlet). Desde objetos de requisição e de resposta são encaminhados a outro recurso, portanto, todos os parâmetros da requisição são mantidos e disponíveis para uso. Qualquer código escrito depois para a frente (, a resposta do pedido) o método não executará como o pedido já está encaminhado. Como o cliente não tem conhecimento sobre essa frente no servidor, portanto, sem história será armazenado no cliente e, como resultado frente e para trás os botões não funciona. Este método é mais rápido em comparação ao uso sendRedirect porque nenhum de ida e volta de rede para o servidor e de volta é obrigatório.

include ()

Este método inclui o conteúdo de um recurso na resposta. O recurso pode ser um servlet ou uma página JSP ou um arquivo HTML. Este método também permite programático server-side includes. O servlet incluído não pode alterar o código de status ou um conjunto de cabeçalhos a resposta, qualquer tentativa de fazer isso é ignorado. Este método pode ser chamado a qualquer momento. Ele só pode usar ServletOutputStream ou Writer do objeto de resposta para gravar as informações.

Exemplo:

RequestDispatcher dispatcher =

request.getRequestDispatcher("/inicio.jsp");

dispatcher.forward(request, response);

**LISTENERS E FILTERS**

Permitem que tenhamos maior controle do uso das nossas aplicações web Listeners serão usualmente utilizados para observar o ciclo de vida de atributos (criação, atualização e remoção) para os escopos disponíveis

Filters, por sua vez, permitem que as requisições possam ser interceptadas antes ou depois de alcançar o recurso desejado (um servlet, por exemplo)

Ambos funcionam como funções de callback, ou seja, após a criação, o contêiner é quem faz chamadas a estes métodos.

Usando a API de servlets (javax.servlet), podemos criar listeners de eventos para aplicações web.

Estes listeners estão definidos como interfaces nesta API Estão disponíveis para os escopos:

Escopo de aplicação: ServletContextListener, ServletContextAttributeListener

Escopo de sessão: HTTPSessionListener, HTTPSessionAttributeListener

Escopo de requisição: ServletRequestListener, ServletRequestAttributeListener

public class EscutadorAplicacao implements ServletContextListener {

public void contextDestroyed(ServletContextEvent arg0) {}

public void contextInitialized(ServletContextEvent arg0) {}

}

public class EscutadorAtributoAplicacao implements

ServletContextAttributeListener {

public void attributeAdded(ServletContextAttributeEvent arg0) { }

public void attributeRemoved(ServletContextAttributeEvent arg0) {}

public void attributeReplaced(ServletContextAttributeEvent arg0) {}

}

public class EscutadorAplicacao implements ServletContextListener,

ServletContextAttributeListener {

public void contextDestroyed(ServletContextEvent arg0) {

System.out.println("Aplicação encerrada");

}

public void contextInitialized(ServletContextEvent arg0) {

System.out.println("Aplicação inicializada");

Calendar cal = new GregorianCalendar();

arg0.getServletContext().setAttribute("iniciou", cal);

}

public void attributeAdded(ServletContextAttributeEvent arg0) {

System.out.println(arg0.getName() + ": ");

System.out.println(arg0.getValue().toString());

}

public void attributeRemoved(ServletContextAttributeEvent arg0) { }

public void attributeReplaced(ServletContextAttributeEvent arg0) { }

}

Os filtros permitem a interceptação de uma solicitação antes/após que esta atinja o recurso solicitado**.** Em outras palavras, um filtro dá acesso aos objetos HTTPServletRequest e HTTPServletResponse antes destes serem passados a um servlet**.** Um filtro pode ser um ponto ideal para: log de requisições, criptografia, autenticação, compressão de dados, validação do usuário, etc

**PÁGINAS JSP E SCRIPTLETS**

Os principais elementos de uma página JSP são: declarações, expressões, scriptlets e diretivas. Os três primeiros têm sintaxes e usos similares, mas têm diferenças importantes. Vamos explicar, com alguns exemplos, as semelhanças e diferenças entre eles.

**AS DECLARAÇÕES**

As declarações situam-se entre os delimitadores <%! e %>. São usadas para definir variáveis e métodos específicos para uma página JSP. Os métodos e variáveis declaradas podem então ser referenciados por outros elementos de criação de scriptlets na mesma página. A cada declaração deve ser finalizada ou separada por "ponto-e-vírgula". Vejamos alguns exemplos:

<%! int i = 0; %>

<%! int a, b; double c; %>

<%! Circle a = new Circle(2.0); %>

Você deve declarar uma variável ou um método antes de usá-lo. O escopo de uma declaração é geralmente o arquivo JSP, mas se for incluído outros arquivos com a diretiva include, o escopo se expande para o cover do arquivo incluído.

**EXPRESSÕES**  
  
 As expressões situam-se entre os delimitadores <%= e %>. Podem conter alguma expressão válida da linguagem de script usada nessa página (o padrão é que a Linguagem seja Java), MAS SEM PONTO-E-VíRGULA.   
  
Veja alguns exemplos:

<%= Math.sqrt(2) %>

<%= items[i] %>

<%= a + b + c %>

<%= new java.util.Date() %>

A expressão Java é avaliada (da esquerda para a direita), convertida em String e depois inserida na página. Essa avaliação é feita em tempo de execução (quando a página é solicitada) permitindo fácil e rápido acesso a informação que foi requisitada. Por exemplo, uma exibição de data e hora em que a página é acessada.

Para construir uma expressão em JSP você pode colocar entre as tags qualquer expressão definida na Especificação da Linguagem Java. Ao contrário dos scriptlets (que veremos a seguir), uma expressão não aceita ponto e vírgula e define somente uma expressão da Linguagem.

**SCRIPTLETS**

Os scriplets situam-se entre os delimitadores <% e %>. Permitem você escrever trechos de código da Linguagem usada na página. Veja o exemplo abaixo:

<html>

<body>

.

.

.

<%

resposta = request.getParameter("opcao");

if (resposta.equals("sim")){

respostas[0]++;

}else{

respostas[1]++;

}

numSim = respostas[0];

numNao = respostas[1];

try{

output = new ObjectOutputStream( new FileOutputStream(f));

output.writeObject(respostas);

output.flush();

output.close();

}

catch(IOException io){

io.printStackTrace();

}

%>

.

.

.

</html>

</body>

Lembre-se que em um script você deve finalizar as expressões através do uso de ponto-e-vírgula. Quando você escreve um script, você pode usar algum dos objetos implícitos do JSP ou das classes importadas através da diretiva page, variáveis ou métodos (declarados entre as tags <%! e %> ou objetos nomeados através da tag <jsp:useBean>.

**DIRETIVAS**  
  
 As diretivas são usadas para fornecer informações especiais ao container JSP sobre a página JSP quando esta é compilada para servlet. As diretivas JSP afetam a estrutura global da classe servlet. Existem dois tipos principais de diretivas:

* page: permite situações como importação de classes, customização de super classes servlet entre outras;
* include: permite que seja inserido o conteúdo de um arquivo no servlet no momento em que o arquivo JSP é traduzido para servlet.

Diretiva page: A diretiva page tem a seguinte sintaxe:

<%@ page attribute1=valor1 attribute2=valor2 attribute3=... %>   
  
 Alguns dos atributos de page são import, que especifica qual o pacote a ser importado e session (com os valores true/false), que indica se a variável predefinida session (do tipo HttpSession) deve estar ligada a sessão existente.

Diretiva include:A diretiva include permite que sejam incluídos arquivos na hora em que a página JSP é traduzida no servlet. Uma directive include é algo como:

<%@ include file="relative url" %>

Esta diretiva pode, então, ser usada para incluir arquivos que sejam usadas em diversas páginas como, por exemplo, um arquivo que represente um menu de navegação que está presente em todas as páginas do site; assim, o menu é editado apenas uma vez e incluído em todas as páginas por meio da diretiva include. Isso facilita a manutenção do site.

**ESTRUTURA DE UM PROJETO JSP**

**WEB-INF**

Diretório dentro do seu contexto que será responsável por conter todas as classes da aplicação e armazenamento de código java(JavaBeans,servlets entre outros..).É criado por convenção e padrão da especificação um diretório chamado ?classes?, para organizar suas classes (\*.class e \*.java) nas chamadas packages, que por sua vez podem ser criadas quantas forem necessárias para uma melhor organização de seu projeto.Observação: seus arquivos ?\*.jsp? podem se encontrar em qualquer lugar dentro de seu diretório de contexto,exceto neste diretório ou quaisquer um de seus subdiretórios.

**Web.xml**

Este arquivo descreverá a aplicação segundo as normas da versão 2.2 das especificações de Servlets e JavaServer pages.Ao contrário de aplicações simples que só utilizarão JSP, a personalização deste arquivo somente é necessária se você precisar adicionar servlets e/ou parâmetros de inicialização.Em colunas futuras, estaremos descrevendo melhor este arquivo.

**LIB**

Este diretório irá conter todo e qualquer arquivo (biblioteca) referente à aplicação.Formado principalmente por arquivos \*.jar (java arquive) e .war (web arquives), e \*.tdl (Tag Libraries).

Exemplo: Drives JDBC.( mysql-connector-java-3.0.6-stable-bin.jar).Jar´s e War´s de sua aplicação.

**Tag Libraries (estados.tdl)**

Você poderá criar diretórios optativos para melhor organizar seus arquivos.Por exemplo um diretório de documentação: percebam que geralmente em ferramentas IDE para Java como o JBuilder, por exemplo, vários diretórios são criados em seu contexto.Um deles é o de documentação.

**WEB.xml**

O arquivo chave para implantação de um servlet é o “web.xml” (descritor de implantação). Este arquivo descreverá a aplicação segundo as normas da versão 2.2 das especificações de Servlets e JavaServer pages.

<web-app 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" version="2.4">

<servlet>

<servlet-name>primeiro</servlet-name>

<servlet-class> br.com.siriusnet.cursojava.PrimeiroServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>primeiro</servlet-name>

<url-pattern>/alou</url-pattern>

</servlet-mapping>

</web-app>

**FRAMEWORK JSF – JAVA SERVER FACES**

Mais que um framework para desenvolver aplicações web de forma ágil, JSF foi incorporado ao especificação J2EE.

**Quais os "módulos" que compõem o JSF?**

* Componentes
* Eventos
* Valiação e conversões
* Navegabilidade
* Backbeans

**Divisão de papéis com JSF?**

* Web-Design: Quem cria a "cara" da aplicação, em nosso caso usando HTML
* Desenvolvedores de aplicação: Quem é responsável por programar objetos, validadores, classes de suporte etc
* Desenvovedores de componentes: Normalmente, o cara que tem uma boa experiência em programar interface com o usuário e prefere criar UIComponents customizados.
* Arquitetos de aplicação: O responsável por planejar a aplicação se preocupando com: Escalabilidade, comunicação entre os módulos, Persistência, Transação etc. (Nosso conhecido plano arquitetural).
* Fornecedores de ferramentas: São os que fornecem ferramentas para auxiliar na produtividade do time.
* Embora haja a definição de papéis é muito comum a acumulação de papéis e até mesmo a divisão de um papel.

**O que JSF trás de bom?**

* MVC para aplicações web
* "fácil" de usar
* componentes extensíveis
* Boa demanda do mecado e conseqüentemente dos desenvolvedores
* Código aberto

**O que temos de diferente com JSF ?**

* Se tornou um padrão
* Uso de componentes (cuidado, não confundir com componentes de negócio)
* Conceito de evento para os componentes (temos um conceito muito parecido com os componentes swing)
* Gerencia estados dos componentes

**Curiosidade**

* Struts e JSF podem ser usados juntos em um mesmo projeto.
* Até mesmo em uma mesma página
* Dentro até mesmo de um mesmo formulário (isso fede, mas é possível)

A principal diferença entre JSP e JSF é que JSF elabora portais de internet com foco em componente, enquanto JSP (tirando ele com o uso de Facelets) possui uma abordagem mais parecida com o PHP ou ASP.

Ambos são padrões Java, mas JSP está entrando em "desuso" por conta das boas características do JSF 2.0 (que tem suporte Ajax de forma nativa!).

**COMUNICAÇÃO ENTRE SERVLETS E PÁGINAS JSP.**

Para que um Servlet Controller possa repassar a requisição recebida para uma página JSP, é necessário utilizar um método específico da classe javax.httpServletRequest.

Assinatura do Método “getRequestDispatcher ()” da classe HttpServletRequest

public javax.servlet.RequestDispatcher getRequestDispatcher

(javax.lang.String p\_path);

Esse método retorna um referência para um objeto que implementa a interface javax.servlet.RequestDispatcher e que atua como um “wrapper” para o recurso indicado no path passado como parâmetro para a função. Assim se você passar como parâmetro, por exemplo, o caminho relativo de uma página JSP, esse método retornará uma referência a um “wrapper” dessa página JSP.

Esse “wrapper”, por sua vez, disponibiliza método forward () que permite que você repasse a requisição para o recurso “encapsulado” pelo “wrapper”.

Assinatura do método “forward ()” da interface RequestDispatcher

Public void forward(HttpServletrequest p\_request,

HttpServletResponse, p\_response);

Podemos, dessa forma, implementar um Servlet de exemplo que não faz nada, apenas repassa todas as requisições recebidas para uma página JSP.

Exemplo de Servlet que repassa requisições para uma página JSP.

Import java.io.\*;

Import javax.servlet.\*;

Import javax.servlet.http.\*;

// Servlet simples que repassa requisições recebidas para a

// págins JSP “”OiPessoas.jsp”

public class ServletRepassaReqs extends HttpServlet {

public void do Get()httpServletRequest p-request,

HttServletResponse p\_response)

throws IOException {

// Repassando a requisição para a página JSP OiPessoas.jsp try {

p\_request.getRequestDispatcher(”/OiPessoas.jsp”).forward

(p\_request,

p\_response);

} catch(ServletException p\_e) {}

}

}

No Servlet anterior, existe um ponto importante que deve ser observado: o path passado como parâmetro para a função getRequestDispatcher () referência um recurso contido na mesma , aplicação WEB do Servlet, e, dessa forma, deve excluir a parte referente ao diretório virtual da aplicação (ou seja, esse path não deve ser escrito como livroservlets/OiPessoas.jsp ). Além disso, é importante observar que o método forward poderá lançar uma exceção: umasd das cuasas pode ser uma exceção lançada pelo próprio recurso referenciado pelo RequestDispatcher.

Podemos também implementar a página JSP referenciada no servlet anterios como:

Exemplo de página JSP “OiPessoas.jsp”

<HTML>

<BODY>

<% out.println (“OiPessoas!”);%>

</BODY>

</HTML>

Assim, a cada requisição recebida, o Servlet ServletRepassaReqs repassa a requisção para a página JSP OiPessoas.jsp, que por sua vez retorna para o Browser a seguinte página HTML

Resposta do Servlet “ServletRepassaReqs”

<HTML>

<BODY>

OI Pessoas!

</BODY>

</HTML>

**Referências Bibliográficas**

<http://www.criarweb.com/artigos/modelo-visao-codeigniter.html>- Acessado em 16/05/2011

<http://www.icmc.usp.br/~mello/livro-j2ee.pdf>- Acessado em 16/05/2011

<http://www.roseindia.net/help/java/d/requestdispatcher.shtml>- Acessado em 16/05/2011

<http://www.devmedia.com.br/post-3573-Fundamentos-de-Servlets.html>- Acessado em 16/05/2011

<http://www.arquivodecodigos.net/dicas/java-servlets-entendendo-a-classe-httpservlet-2338.html>- Acessado em 16/05/2011

<http://www.arquivodecodigos.net/dicas/jsp-java-server-pages-aprenda-a-criar-cookies-em-jsp-812.html>- Acessado em 16/05/2011

<http://www.htmlstaff.org/ver.php?id=1585>- Acessado em 16/05/2011

<http://www.arquivodecodigos.net/dicas/jsp-java-server-pages-aprenda-a-colocar-objetos-java-em-sessoes-jsp-391.html>- Acessado em 16/05/2011

<http://www.dei.isep.ipp.pt/~i020525/trabalhos/2ano-1sem-lpg2-tp1/jsp.pdf>- Acessado em 16/05/2011

<http://maniish.wordpress.com/2009/11/10/server-tags/>- Acessado em 16/05/2011

<http://www.htmlstaff.org/ver.php?id=653>- Acessado em 16/05/2011

<http://javafree.uol.com.br/viewtopic.jbb?t=1407>- Acessado em 16/05/2011

<http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/jsf/jsf.htm> - Acessado em 16/05/2011

<http://viralpatel.net/blogs/2009/01/tutorial-java-servlet-filter-example-using-eclipse-apache-tomcat.html> - Acessado em 16/05/2011