Isaque Gomes de Oliveira

Marcelo Hernane Braga Coelho

Marcos Rodrigo Cabral

Robson de Jesus da Silva

Trabalho de JSP

Faculdades Santo Agostinho

7º Período de Sistemas de Informação

17/05/2011

1 - Modelo de Programação para Web

A teclonogia JSP é uma abreviação de *Java Server Pages*, que traduzindo seria como Páginas de Servidor Java. Portanto, é uma tecnologia orientada a criar páginas *web* com programação em *Java.*

O JSP possibilita a criação de apliações que podem ser executadas em diversos servidores servidores *web.* As páginas JSP contém códigos em HTML/XML juntamente com *tags* especiais para programar utilizando a sintaxe da linguaguem *java.*

Para que as páginas possam ser compiladas, ou executadas, os JSP se baseia em *servlets*, que são programas em *Java* que tem por objeitivo executar o aplicativo no servidor.

Em suma, as páginas em JSP, são criadas de maneira parecida com as páginas *ASP* ou *PHP.* São gerados arquivos .jsp que incluem, dentro da estrutura de etiquetas HTML, as sentenças Java a executar no servidor.

1.1 Descreva o protocolo HTTP

O HTTP (*Hypertext Transfer Protoco* – Protocolo de Transferência de Hipertexto) é um protocolo de comunicação da camada de aplicação do modelo OSI. Ele é responsável pelo tratamento de pedidos e respostas entre o cliente e o servidor. Como a maioria dos protocolos de rede, baseando-se no paradigma de requisição e resposta. Um programa requisitante (cliente) estabelece uma conexão com um outro programa receptor (servidor) e envia-lhe uma requisição, contendo a URL, a versão do protocolo, uma mensagem com os modificadores da requisição, informações sobre o cliente e, possivelmente, o conteúdo no corpo da mensagem.

1.1.1 Quais as diferenças entre GET e POST??

O protocolo HTTP define oito métodos que indicam a ação a ser realizada no recurso especificado. O método determina o que o servidor deve fazer com o URL fornecido no momento da requisição de um recurso.

O GETé o método mais comum. Ele solicita algum recurso como um arquivo ou *script* por meio do protocolo HTTP. Ou seja, ele pega o dado que está identificado pela URL e envia para o servidor.

O método GET signiﬁca recupere qualquer informação que esteja identiﬁcada pela URL de requisição. Se a URL se refere a um processo que produz dados, os dados produzidos serão retornados como resposta à requisição.

Quando acessamos uma página através de um navegador estamos executando o método GET para a obtenção desta página e os recursos pertencentes a ela (imagens, outras páginas, etc). Uma resposta pode ser passível de cache quando isto é sinalizado explicitamente pelo servidor através de metainformações embutidas na resposta.

Por sua vez, o método *POST*, envia dados para serem processados (por exemplo, dados de um formulário HTML) para o recurso especificado. Os dados são incluídos no corpo do comando. Sua utilização em uma requisição ocorre quando é necessário enviar dados ao servidor para serem processados, geralmente por um programa *script* identificado no *Request-URI*. A função exercida pelo método POST é determinada pelo servidor e geralmente é dependente da URL da requisição. As informações enviadas ao servidor são então tratadas, no servidor, pela URL, ou seja, esta URL pode ser um programa que saiba como tratar e o que fazer com as informações recebidas por ele.

1.2 Descreva os princípios da Arquitetura REST

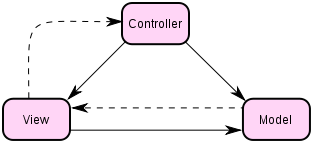
A Arquitetura REST ou *Representational Sate Transfer* – Transferência de Estado Rpresentacional – é uma técnica de engenharia de software para sistemas ditribuidos na *web.*

O termo *REST* se referia, originalmente, a um conjunto de princípios de arquitetura, usados para descrever qualquer interface web simples que utiliza XML e HTTP (ou YAML, JSON, ou texto puro), sem as abstrações adicionais dos protocolos baseados em padrões de trocas de mensagem como o protocolo de serviços web SOAP. Segue abaixo alguns de seus princípios:

* 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, 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 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 URL.
* 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 ou 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.

1.3 Descreva a arquitetura MVC (Model - View - Controller)

Model-view-controller (MVC) é um padrão de arquitetura de software que visa a separar a lógica de negócio da lógica de apresentação, permitindo o desenvolvimento, teste e manutenção isolado de ambos. A criação desse modelo se deu pelo aumento da complexidade das aplicações desenvolvidas. Por isso tornou-se relevante a separação entre os dados e a apresentação das aplicações.



Exemplo do Modelo MVC

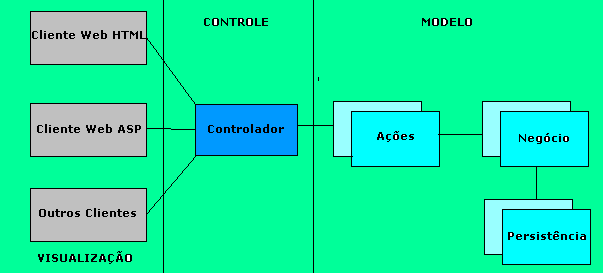
Como mostra a figura acima, a arquitetura MVC é composta por três tipos de objetos. O *model* é o objeto de aplicação, a *view* é a apresentação na tela e o *controller* define a maneira como a interface do usuário reage às entradas do mesmo.

O *MVC* tem como principal objetivo separar dados ou lógicsas de negócios (*Model*) da interface do usuário (*View*) e o fluxo da aplicação (*Controller*). A ideia é permitir que uma mensagem da lógica de negócios possa ser acessada e visualizada através de várias interfaces. Na arquitetura *MVC*, á lógica de negócios, ou seja, nosso *Model* não sabe quantas nem quais as interfaces com o usuário esta exibindo seu estado, a *view* não se importa de onde esta recebendo os dados, mas ela tem que garantir que sua aparência reflita o estado do modelo, ou seja, sempre que os estados do modelo mudam, o modelo notifica as *views* para que as mesmas atualizem-se.

1.3.1 Qual a função dos controladores (Controllers)? Com quem eles se comunicam?

Um controlador funciona como um intermediário entre a camada de apresentação e a camada de negócios, sua função como já diz, é controlar e coordenar o envio de requisições feitas entre a *view* e o *model*. O *controller* define o comportamento da aplicação, e é ele quem interpreta as solicitações (cliques, seleções de menus) feitas por usuários. Com bases nestes requerimentos o controlador comunica-se com o *model* que seleciona a *view* e a atualiza para o usuário, ou seja, o controlador controla e mapeia as ações.

Embora o *MVC* só contenha três camadas há outra camada fundamental para o bom andamento da arquitetura, esta é um mecanismo de eventos necessário a comunicação entre outros três elementos, este elemento permite uma comunicação assíncrona que é invocada quando algum evento interessante acontece, esta quarta camada contém os *beans* de entidade onde se localizam os métodos get e set das classes.



Exemplo da arquitetura MVC.

Em suma, o *controller* é o lugar onde serão processadas todas as requisições feitas através da interface (*View*). O controle também acessa o *model* afim de obter determinadas informações. Toda lógica da aplicação (validações, atribuições, etc) é feita no *controller*.

1.3.2 Qual a função das visões(Views)? Com quem elas se comunicam?

*View* ou visão é a camada de apresentação para o usuário, é a interface que proporcionará á entrada de dados e a visualização de respostas geradas. Nas aplicações web é representado pelo HTML que é mostrado pelo browser. Geralmente a visão contém formulários, tabelas, menus e botões para entrada e saída de dados.

A visão deve garantir que sua apresentação reflita o estado do *model*, quando os dados do *model* mudam, o mesmo notifica as *views* que dependem dele, cada *view* tem a chance de atualizar-se.

Desta maneira é possível ligar muitas *views* a um *model* podendo fornecer diferentes apresentações. Essa camada não contém códigos relacionados á lógica de negócios, ou seja, todo o processamento é feito pelo *model* e este repassa para a *view.*

1.3.3 Qual a função dos modelos(Models)? Com quem eles se comunicam?

É o modelo da aplicação onde são definidos propriedades e atributos. Ou seja, é a camada que contém a lógica da aplicação e é responsável pelas regras de negócio. Para sistemas persistentes, o *model* representa a informação (dados) dos formulários e as regras SQL para manipular dados do banco.

O*model* mantém o estado persistente do negócio e fornece ao *controller*a capacidade de acessar as funcionalidades da aplicação.O*model* é a camada principal. Toda aplicação deve se comunicarcom o *model,* pois se atuarem isoladamente não saberão quais informações ou interfaces que terá de atualizar.O*model* somente acessa á base de dados e deixa os dados prontos para o *controller,* este por sua vez encaminha para a *view* correta.

**2. Framework JSP**

**2.1 O que são Servlets e a classe HttpServlet? Para que servem? Quais**  
**os principais métodos? Dê exemplos.**

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.\*;

importjavax.servlet.\*;

importjavax.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>EnderecoipRemoto:" + 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";

   }

}

**2.1.1 Descreva a função e os principais métodos das classesHttpServletRequest e HttpServletResponse, com exemplos.**

Estas 2 classes são cruciais para o tratamento de requisições através de servlets  
 A classe HTTPServletRequest representa uma requisição e contém todas as informações do cliente solicitanteIP do cliente, cookies, header do HTTP, etc  
 A classe HTTPServletResponse é utilizada para preparar a resposta ao cliente  
Com esta classe podemos:

Adicionar novos cookies, adicionar uma nova entrada no cabeçalho HTTP, redirecionar uma requisição para tratamento por outro recurso, etc.

Alguns metodos:

HttpServletRequest  
String getParameter(String name) – Retorna parâmetros de solicitação de campos de um formulário;  
String getRemoteAddress() – Retorna o endereço IP da máquina cliente;  
Cookie [] getCookies() – Retorna um array de todos os cookies;  
String getHeader(String name) – Retorna o valor da entrada name no cabeçalho HTTP como uma string.  
  
HttpServletRequest  
RequestDispatcher getRequestDispatcher (String url) – Cria um “dispatcher” para o recurso url; assim podemos encaminhar a requisição utilizando o método forward() da classe RequestDispatcher  
HttpSession getSession() – Retorna uma referência para a sessão associada à requisição, ou cria uma nova caso a sessão não exista  
  
HttpServletResponse  
java.io.ServletOutputStream getOutStream() – Retorna o outputstream de forma que possamos gravar dados no cliente;  
void setContentType(String mime) – Configura o tipo de conteúdo de saída; Lista completa dos tipos MIME em  
  
  
HttpServletResponse  
void addCookie(Cookie cookie) – Adiciona um cookie ao cabeçalho da resposta;  
void addHeader(String name, String value) – Adiciona o cabeçalho especificado à resposta;  
void sendRedirect(String url) – Redireciona uma solicitação para outro recurso (servlet ou página www).  
  
  
**2.1.2 Descreva a função e os principais métodos das classe**  
**RequestDispatcher, com exemplos.**  
  
RequestDispatcher getRequestDispatcher (String url) – Cria um “dispatcher” para o recurso url; assim podemos encaminhar a requisição utilizando o método forward() da classe RequestDispatcher

1. RequestDispatcherrd = request.getRequestDispatcher(jsp);
2. nbsp;        rd.forward(request, response);

**2.1.3 O que são Listeners? Qual a sua função? Quais os principais**  
**métodos? Dê exemplos.**

Um listener é normalmente implementado em forma de interface.

Ele serve para escutar o que acontece em um objeto e avisar a outro.

Exemplo:

Eu tenho um sistema que vc tuna carro.

Ai quando vc altera o carro.cor vc tem que avisar a parte gráfica do programa pra ela trocar a cor do carro na tela então o carro pode ter um espaço para vc colocar um ouvido (listener) da tela nele então na tela ao criar o carro:

1. Carro carro = **new** Carro(Color.white);
2. carro.addColorChangeListener(**new** ColorChangeListener(){
3. colorChanged(ColorChangeEvent e)
4. {
5. panelCarro.setColor(e.getNewColor);

e no objeto no setColor, vc chama esse evento

1. setColor(Color color)
2. {
3. Color oldColor = **this**.color;
4. **this**.color = color;
5. fireColorChangeListener(oldColor, color);

**3.1.4 O que são Filters? Qual a sua função? Quais os principais**  
**métodos? Dê exemplos.**

Servlet Filters são os últimos componentes que são adicionados na Servlet 2.3 especificações. Esses filtros são utilizados basicamente para interceptar e modificar requisições e resposta do servidor. Considere um cenário onde você deseja verificar sessão da solicitação de todos os usuários e se ele é válido, então só você querer deixar o usuário acessar a página. Você pode conseguir isso, marcando sessões em todas as páginas do servlet (ou páginas JSP) consultas que os usuários ou você pode fazer isso usando o filtro.

pacote net.viralpatel.servlet.filters;  
  
importação java.io.IOException;  
importação java.util.Date;  
  
importação javax.servlet.Filter;  
importação javax.servlet.FilterChain;  
importação javax.servlet.FilterConfig;  
importação javax.servlet.ServletException;  
importação javax.servlet.ServletRequest;  
importação javax.servlet.ServletResponse;  
importação javax.servlet.http.HttpServletRequest;  
  
público de classe LogFilter implementa filtro {  
  
   pública anular doFilter (ServletRequest req, res ServletResponse,  
           FilterChain chain) throws IOException, ServletException {  
  
       HttpServletRequest request = (HttpServletRequest) req;  
  
       / / Obter o endereço IP da máquina cliente.  
       String = ipAddress request.getRemoteAddr ();  
  
       / / Log do endereço IP e timestamp atual.  
       System.out.println ( "IP" + + ipAddress "Time"  
                           + novo Date ()). toString ();  
  
       chain.doFilter (req, res);  
   }  
   pública void init (FilterConfig config) throws ServletException {  
  
       / / Pega o parâmetro init  
       testParam config.getInitParameter String = ( "test-param" );  
  
       / / Imprime o parâmetro init  
       System.out.println ( "Teste Param:" + testParam);  
   }  
   pública void destroy () {  
       / / Adiciona o código para liberar qualquer recurso  
   }  
}  
  
Neste exemplo de filtro, temos implementado uma interface javax.servlet.Filter e substituir seus métodos init, doFilter e destruir.  
O **init ()** método é usado para inicializar qualquer código que é usado pelo filtro. Observe também que, método init () vai ter um objeto de FilterConfig que contém informações de nível de filtro diferente, bem como parâmetros de inicialização que é passado da Web.xml (descritor de implantação).  
O **doFilter ()** método irá fazer o registro real de informações. Você pode modificar este método e adicione o seu código, que pode modificar request / session / resposta, adicione algum atributo no pedido etc  
O **destroy ()** método é chamado pelo container quando se quer coletar lixo no filtro. Isso geralmente é feito quando o filtro não é usado por muito tempo e servidor quer alocar memória para outras aplicações.  
  
Completando as duas questões:  
  
**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) { }  
}  
  
  
**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  
  
**3.1.5 Como se trabalha com sessões? E com cookies?**  
  
**Sessões:**  
A grande diferença entre fazer um código servlet ou JSP está no conteúdo em Java que se pretende introduzir. Se o objectivo é realizar uma página toda em Java com pouco ou nenhum conteúdo estático então poderá ser feito um servlet que consiste em fazer uma classe. Se o pretendido é realizar uma página com pouco conteúdo dinâmico e com muitos elementos estáticos então o JSP é mais apropriado. Normalmente um servidor que suporte servlets suporta também JSP o que torna a decisão exclusivamente do programador.   
  
Classes e interfaces especificas do JSP:

Package javax.servlet.jsp

Package javax.servlet.jsp.tagext

Ex.:  
Você pode colocar qualquer objeto Java em uma sessão e recuperá-lo nas páginas seguintes. Este exemplo mostra como criar um Vector de nomes, colocá-lo na sessão e exibir o valor de seus elementos na página seguinte:

public class Cliente implements java.io.Serializable{  
   public String nome;  
   public int idade;  
     
   public Cliente(String nome, int idade){  
     this.nome = nome;  
     this.idade = idade;  
   }  
}  
  
Compile esta classe a coloque-a no pacote estudos. Consulte outras dicas para saber como compilar classes a serem usadas em página JSP.  
  
página JSP que colocará os objetos na sessão:  
<%@ page import="java.util.Vector, estudos.Cliente" %>  
<html>  
<head><title>Aprendendo JSP</title></head>  
<body>  
  
<%  
 // cria um Vector de clientes  
 Vector clientes = new Vector();  
   
 // adiciona elementos ao Vector  
 clientes.add(new Cliente("Osmar J. Silva", 38));  
 clientes.add(new Cliente("Fernanda Lima", 21));  
   
 // coloca o Vector na sessão  
 session.setAttribute("clientes", clientes);  
%>  
  
</body>  
</html>  
  
3 - Crie a página JSP que recuperará os objetos na sessão e exibirá seus valores:  
<%@ page import="java.util.Vector, estudos.Cliente" %>  
<html>  
<head><title>Aprendendo JSP</title></head>  
<body>  
  
<%  
 // obtém o Vector guardado na sessão  
 Vector clientes = (Vector)session.getAttribute("clientes");  
   
 // obtém os clientes contidos no Vector  
 for(int i = 0; i < clientes.size(); i++){  
     Cliente cliente = (Cliente)clientes.elementAt(i);  
     out.println("Cliente: " + cliente.nome +   
         " - Idade: " + cliente.idade + "<br>");  
 }  
%>  
  
**Cookies**  
  
 Cookie é um mecanismo padrão fornecido pelo protocolo HTTP e que permite gravarmos pequenas quantidades de dados persistentes no navegador de um usuário. Tais dados podem ser recuperados posteriormente pelo navegador. Esse mecanismo é usado quando queremos recuperar [informações](http://www.htmlstaff.org/ver.php?id=1585) de algum usuário. Com os cookies, pode-se reconhecer quem entra num [site](http://www.htmlstaff.org/ver.php?id=1585), de onde vem, com que periodicidade costuma voltar. Para se ter uma idéia de como eles fazem parte da sua vida, dê uma olhada na sua máquina. Se você usa o Internet Explore 5.1, vá a c:\windows\cookies. No Communicator 4.7, os cookies ficam em c:\[arquivos](http://www.htmlstaff.org/ver.php?id=1585) de programas\netscape\users. Os cookies em si não atrapalham ninguém, se propriamente usados.

Como padrão, os cookies expiram tão logo o usuário encerra a navegação naquele site, porém podemos configurá-los para persistir por vários dias. Além dos dados que ele [armazena](http://www.htmlstaff.org/ver.php?id=1585), um cookie recebe um nome; um servidor pode então definir múltiplos cookies e fazer a identificação entre eles através dos seus nomes.

Os cookies são associados ao URL da página que os manipula.

Ex.:  
<html>  
<head><title>Aprendendo JSP</title></head>  
<body>  
  
<%  
 // Para criar um cookie você fornece o nome do cookie e o  
 // seu valor (ambos do tipo String)  
   
 // cria um cookie chamado cliente  
 String valor = "Osmar J. Silva";  
 Cookie cookie = new Cookie("cliente", valor);  
   
 // define o tempo de vida do cookie para um mês  
 cookie.setMaxAge(60 \* 60 \* 24 \* 30);  
   
 // coloca o cookie na resposta enviada ao cliente  
 response.addCookie(cookie);  
%>  
  
</body>  
</html>  
  
  
**2.2 O que são páginas JSP? O que são Scriptlets?**

As páginas JSP, ou Java Server Pages, foram criadas para contornar algumas das limitações no desenvolvimento com Servlets: se em um Servlet a formatação da página HTML resultante do processamento de uma requisição se mistura com a lógica da aplicação em si, dificultando a alteração dessa formatação, em uma página JSP essa formatação se encontra separada da programação, podendo ser modificada sem afetar o restante da aplicação.

Assim, um JSP consiste de uma página HTML com alguns elementos especiais, que conferem o caráter dinâmico da página. Esses elementos podem tanto realizar um processamento por si, como podem recuperar o resultado do processamento realizado em um Servlet, por exemplo, e apresentar esse conteúdo dinâmico junto à página JSP.

Scriptlets São blocos de [código Java](http://www.htmlstaff.org/ver.php?id=653) escritos em qualquer parte do JSP estando sempre contidos pelos TAGS "<%" e "%>". O código de um scriptlet pode acessar variáveis e métodos definidos em declarações do JSP e objetos externos tais como JavaBeans e Enterprise JavaBeans.

Um scriptlet pode ainda utilizar alguns objetos já disponíveis para o [JSP](http://www.htmlstaff.org/ver.php?id=653).

**2.2.1 O que são as Server Tags?? Explique as Declaration Tags,Expression Tags,Directive Tags, Scriptlet Tags e Action Tags e dêexemplos.**

**<%%>***Um bloco de código incorporado é o código do servidor que executa durante a fase de renderização de páginas do. O código no bloco pode executar instruções de programação e funções de chamada na classe da página atual*

**2.2.2 Descreva a estrutura de um projeto JSP, em especial o conteúdo**  
**da pasta WebContent.**

Este deverá ser a sua estrutura para sua aplicação.Embaixo da pasta webapps (contexto padrão) você deverá construir a estrutura para sua aplicação, conforme mostrado acima.

Vejamos a descrição de cada item:

/usr/java/tomcat/webapps/Diretório de conteúdo: Este será o seu contexto, o diretório de sua aplicação.Os sub-diretórios acima citados e abaixo descritos devem estar embaixo desta pasta (web-inf, lib, docs).

**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.

Bibliotecas de classes de terceiros.

**Tag Libraries (estados.tdl)**

DOCS: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.

**2.2.3 Qual a função do arquivo web.xml? Qual sua estrutura? Como se**  
**fazem mapeamentos? Como definir a páginal inicial de um projeto? E a**  
**página de erro?**

**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>  
  
**2.3 O que é o framework JSF - Java Server Faces ? Quais as diferenças com o JSP?**  
  
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.
  + exemplos de ferramentas:
    - Borland JBuilder 2005 Enterprise
    - Eclipse with Exadel Studio Pro
    - Eclipse with Nitrox JSF IDE
    - IBM Rational with JSF support
    - Macromedia Dreamweaver with JSTL and JSF extensions
    - MyEclipse with JSF Developer
    - Oracle JDeveloper
* 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!).

Parte 3: O Gerenciador de Entidades

**- O que é o EntityManager? Para que ele serve? Dê exemplos.**

**EntityManager**

Na nova Java Persistence specification, o EntityManager é o serviço central para todas as ações de persistência. Entidades são objetos de Java claros que são alocados como qualquer outro objeto Java. Eles não ficam persistentes explicitamente até seu código interagir com o EntityManager para os fazer persistente. O EntityManager administra o O/R que o mapea entre uma classe de entidade e uma fonte de dados subjacente. O EntityManager provê APIs para criar consultas, buscando objetos, sincronizando objetos, e inserindo objetos no banco de dados. Também pode prover caching e pode administrar a interação entre uma entidade e serviços transacionais em um ambiente Java EE como JTA. O EntityManager é firmemente integrado com Java EE e EJB mas não é limitado a estes ambientes;  pode ser usado em programas de Java SE.

**Persistence Contexts**

Começaremos através de uma introdução das condições do core do JPA. Uma unidade de persistência ou persistence-unit é uma configuração nomeada de classes de entidade. Um contexto de persistência é um gerenciamento de Entidades onde, todo contexto de persistência é associado com uma unidade de persistência. Entendendo o contexto de persistência é a chave para entender o EntityManager. Uma inclusão de entidade ou exclusão de um contexto de persistência, determinarão no resultado de qualquer operações persistentes nisto.

Se o contexto de persistência participa em uma transação, que está na memória,

o estado das entidades gerenciadas será sincronizado ao banco de dados. Ainda apesar do papel importante que faz, o contexto de persistência nunca é realmente visível à aplicação. Sempre é acessado indiretamente pelo entity manager e assumido que estar lá quando nós precisarmos dele.

**Persistence Unit**

Um EntityManager mapea um conjunto de classes a um banco de dados particular. Este conjunto de classes é chamado de *persistence unit* (unidade de persistência).

Antes de você poder pensar até mesmo em criar ou consultar entidades com um entity manager, você tem que aprender a empacotar uma unidade de persistência para uso, dentro de um ambiente  Java SE ou Java EE.

Uma unidade de persistência está definida em um arquivo chamado *persistence.xml*. Este arquivo é um descriptor de desenvolvimento exigido no JPA. Um arquivo de persistence.xml pode definir um ou mais unidades de persistência. Este arquivo fica situado no diretório META-INF em:

Ambientes SE, esta pasta fica no classpath da aplicação. Em uma aplicação web, este diretório fica localizado nas pasta WEB-INF\classes\META-INF

Segue um exemplo de um arquivo persistence.xml:

**Listagem 01.**

<?xml version="1.0" encoding="UTF-8"?>

<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">

**<persistence-unit name="novoExemploPU" transaction-type="RESOURCE\_LOCAL">**

<provider>oracle.toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider

</provider>

**<class>teste.Empresas</class>**

**<class>teste.Cartoes</class>**

<properties>

  <property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/devmedia"/>

  <property name="toplink.jdbc.user" value="root"/>

  <property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>

  <property name="toplink.jdbc.password" value="123456"/>

</properties>

 </persistence-unit>

</persistence>

Você deve especificar suas classes de Entidades dentro no arquivo persistence.xml(persistence unit). Como mostradado na listagem 01 em negrito.

Vamos analizar como obter acesso de um EntityManager,  para poder inserir, atualizar,  remover e consultar um bean de entidade no seu bancos de dados. Embora você possa usar uma interface de fábrica em Java EE, esta plataforma provê algumas características adicionais que fazem isto mais fácil e menos áduo de administrar instancias de EntityManager.

**EntityManagerFactory**

EntityManagers podem ser criado ou podem ser obtidos de um EntityManagerFactory. Em uma aplicação Java  SE, você tem que usar um EntityManagerFactory para criar instancias de um EntityManager. Usar o factory não é uma exigência em um ambiente Java EE.

Obtendo um EntityManagerFactory em um ambiente Java SE.

Em um ambiente Java SE, a classe javax.persistence.Persistence é responsável por amarrar um EntityManagerFactory.

**Listagem 02.**

public class Persistence {

  public static EntityManagerFactorycreateEntityManagerFactory(

       String unitName

  );

  public static EntityManagerFactorycreateEntityManagerFactory(

       String unitName,

       java.util.Map

properties

  );

A classe javax.persistence.Persistence procura descriptors da distribuição  persistence.xml dentro de seu classpath de Java. O parâmetro do unitName que você passa dentro permitirá que a execução da persistencia encontre um EntityManagerFactory que combine o nome dado.

EntityManagerFactory factory =

Persistence.createEntityManagerFactory("novoExemploPU");

factory.close( );

Em Java EE, obter um EntityManagerFactory. Pode ser injetado diretamente em um campo ou em um método setter do seu EJBs usando a anotação @javax.persistence.PersistenceUnit

**Listagem 03.**

importjavax.persistence.\*;

importjavax.ejb.\*;

@Stateless

publicMyBean implements MyBusinessInterface {

  @PersistenceUnit(unitName="novoExamploPU")

  privateEntityManagerFactory factory;

  privateEntityManagerFactory factory2;

  @PersistenceUnit(unitName="novoExemploOutroDB")

  public void setFactory2(EntityManagerFactory f) {

  this.factory2 = f;

  }

**- O que é JPQL? Dê exemplos.**

O **Java Persistence Query Language** ( **JPQL** ) é independente de plataforma [orientada a objetos](http://en.wikipedia.org/wiki/Object-oriented) [linguagem de consulta](http://en.wikipedia.org/wiki/Query_language) definida como parte da [Java Persistence API](http://en.wikipedia.org/wiki/Java_Persistence_API) especificação.

JPQL é usada para fazer consultas com entidades armazenados em um banco de dados relacional. É fortemente inspirado no [SQL](http://en.wikipedia.org/wiki/SQL) , e suas pesquisas se assemelham a consultas na sintaxe SQL, mas operam com objetos de entidade JPA e não directamente com as tabelas de banco de dados.

Além de recuperar objetos ( SELECT consultas), JPQL suporta a granel UPDATE e DELETE consultas.

Exemplos:

Suponha que temos classes de entidade JPA definida como esta (métodos get e set omitidos para simplificar):

@ Entity

**público classe** Autor {

  @ Id

**privada** Integer id ;

**privada**String firstName ;

**privada**String lastName ;

  @ ManyToMany

**privada** List < Livro > livros ;

}

@ Entity

**público classe** Book {

  @ Id

**privada** Integer id ;

**privada**String title ;

**privada**String isbn ;

  @ ManyToOne

**privada** publisher ,

  @ ManyToMany

**privada** List <autor>autores ;

}

@ Entity

**públicoclasse** Publisher {

  @ Id

**privada**Integer id ;

**privada**String nome ;

**privada**String endereço ;

  @ OneToMany ( mappedBy = "editor" )

**privadas** List < Livro > livros ;

}

Então, uma simples consulta para recuperar a lista de todos os autores, ordenados alfabeticamente, seria:

**SELECT** um **DA** Autor de um **ORDER BY** um . firstName , um . lastName

Para recuperar a lista de autores que já foram publicados pela XYZ imprensa:

**SELECT DISTINCT** uma **DA** Autor de um **INNER JOIN** um . b livros **WHERE** b . editor . nome = 'XYZ Imprensa'

JPQL suporta parâmetros nomeados, que começam com dois pontos ( : ). Poderíamos escrever uma função que retorna uma lista de autores com o sobrenome é dado como segue:

**importação** javax.persistence.EntityManager ;

**importação** javax.persistence.Query ;

**importação** org.apache.commons.lang.StringUtils ;

... @ SuppressWarnings( "unchecked" )

**público** List <autor>GetAuthorsByLastName ( String lastName ) {

  String queryString = "Selecione um autor de um" +

                       "WHERE: lastName IS NULL ou inferior (a.lastName) =: lastName" ;

  consulta Query = getEntityManager ( ) . createQuery ( queryString ) ;

  . consulta setParameter ( "lastName" , StringUtils. minúsculas ( lastName ) ) ;

**retorno** . consulta getResultList ( ) ;

}

**- O que é Criteria? Dê exemplos.**

Critérios é uma versão simplificada de API para recuperar entidades compondo objetos critério. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set. Esta é uma abordagem muito conveniente para a funcionalidade como "pesquisa" telas onde há um número variável de condições para ser colocado sobre o conjunto de resultados.

A sessão é uma fábrica de Critérios. Instâncias Critério são normalmente obtidos através dos métodos da fábrica em Restrições. eg. por exemplo.

List cats = session.createCriteria(Cat.class) Lista de gatossession.createCriteria = (Cat.class)

   .add(Restrictions.like("name", "Iz%") ) . Adicionar (Restrictions.like ("name", "% Iz"))

   .add( Restrictions.gt( "weight", new Float(minWeight) ) ) . Adicionar (Restrictions.gt ("peso", novo Float (minWeight)))

   .addOrder(Order.asc("age") ) . AddOrder (Order.asc ("idade"))

   .list(); . List ();

You may navigate associations using createAlias() or createCriteria() . Você pode navegar usando associações createAlias () ou createCriteria ().

List cats = session.createCriteria(Cat.class) Lista de gatos session.createCriteria = (Cat.class)

   .createCriteria("kittens") . CreateCriteria ("gatinhos")

       .add(Restrictions.like("name", "Iz%") ) . Adicionar (Restrictions.like ("name", "% Iz"))

   .list(); . List ();

List cats = session.createCriteria(Cat.class) Lista de gatossession.createCriteria = (Cat.class)

   .createAlias("kittens", "kit") . CreateAlias ("gatinhos", "kit")

   .add(Restrictions.like("kit.name", "Iz%") ) . Adicionar (Restrictions.like ("kit.name", "% Iz"))

   .list(); . List ();

Você pode especificar a projeção e agregação usando instâncias de projeção obtida através dos métodos de fábrica em projeções.

List cats = session.createCriteria(Cat.class) Lista de gatossession.createCriteria = (Cat.class)

   .setProjection(Projections.projectionList() . SetProjection (Projections.projectionList ()

       .add(Projections.rowCount() ) . Adicionar (Projections.rowCount ())

       .add(Projections.avg("weight") ) . Adicionar (Projections.avg ("peso"))

       .add(Projections.max("weight") ) . Adicionar (Projections.max ("peso"))

       .add(Projections.min("weight") ) . Adicionar (Projections.min ("peso"))

       .add(Projections.groupProperty("color") ) . Adicionar (Projections.groupProperty ("cor"))

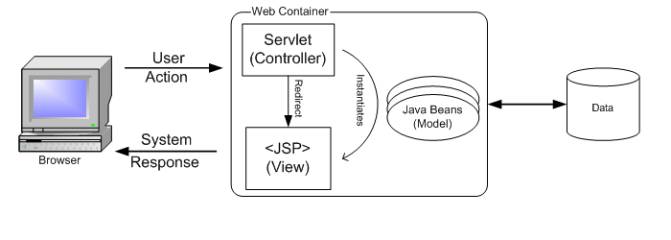
   ) )

   .addOrder(Order.asc("color") ) . AddOrder (Order.asc ("cor"))

   .list(); . List ();

Arquitetura MVC

Também chamada "Model 2 Architecture" em J2EE quem permite separar a aplicação em três camadas: Model, View e Controller.

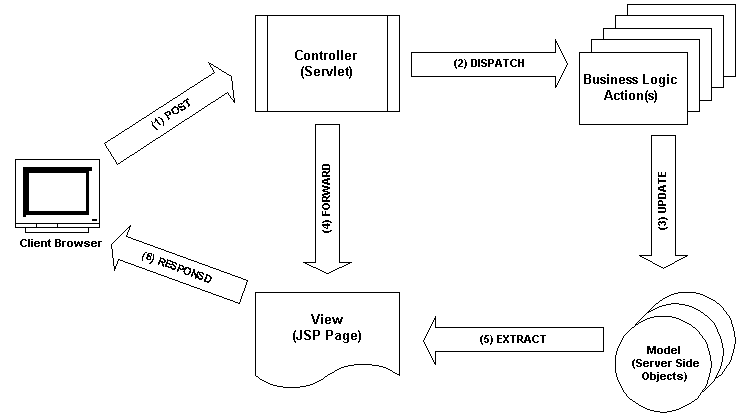


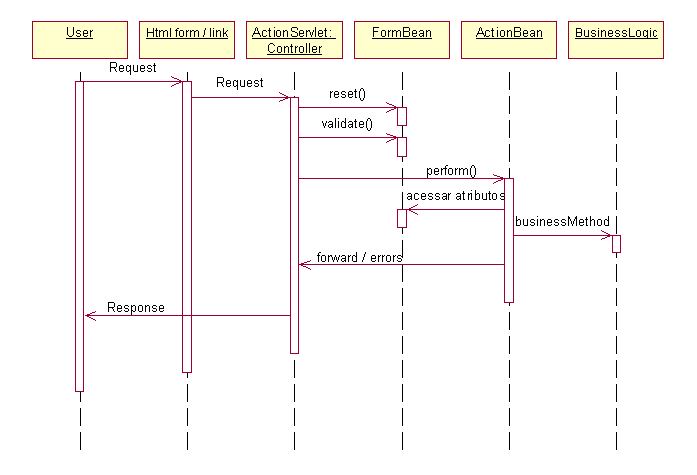
O Model é tudo que envolve o Business Logic, sem considerações de interface. Geralmente é um objeto Java (JavaBean).

O View representa numa interface o estado corrente do Model. Geralmente uma página HTML ou JSP

O Controller é usado para alimentar o Model quando ocorrem entradas na interface par o usuário.

O funcionamento da arquitetura MVC se dá como ilustrado nas figuras abaixo.





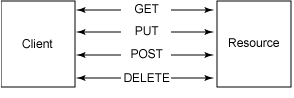
Definição REST

O [Representation State Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer) OU REST é um Estilo de arquitetura de software para sistemas distribuídos. A motivação para implementar serviços REST é porque apresentam protocolos menos complexos, mais poder e flexibilidade nas comunicações, arquitetura amplamente disponível nas empresas e menos overhead de protocolo.

Mas não é recomendado implementar o REST quando há Integrações com produtos fechados WS-\*, quando [WS-Security](http://en.wikipedia.org/wiki/WS-Security" \t "_blank) fizer sentido e quando não houver API HTTP razoável no servidor e/ou clientes-alvo.

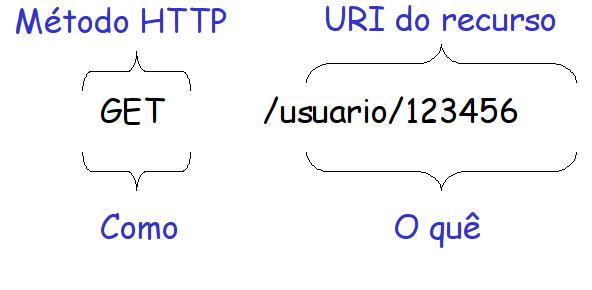
Arquitetura

Vamos identificar os recursos envolvidos e utilizar extensamente os recursos do HTTP para definir um bom protocolo de interação com estes recursos. O REST utiliza o seguinte estilos de acesso aos serviços.



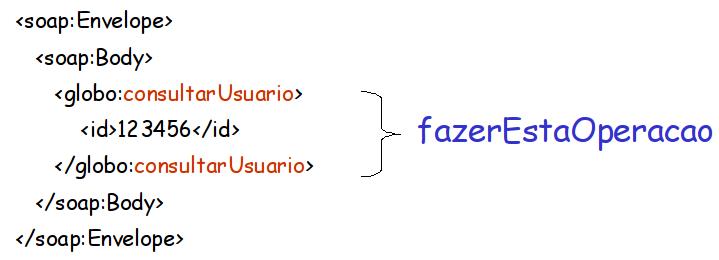
**No REST** clientes interagem com os Recursos através de requisições HTTP GET, PUT, POST e DELETE.

Estilo Declarativo x Imperativo



A URI deve indicar oque você está manipulando e o método (ou verbo) HTTP indicará comovocê está manipulando. Neste exemplo, a URI /usuario/123456 nos indica que estamos manipulando um usuário específico. Sabendo que estamos usando o método HTTP GET, temos a clara indicação de que estamos buscando os dados deste usuário. Este estilo de invocação de serviços pode ser considerado Declarativo.

Em bons protocolos REST, a junção do método HTTP com a URI do recurso já nos indica na maioria dos casos qual é a operação sendo realizada. Assim, com um pequeno trecho do cabeçalho HTTP já somos capazes de compreender a comunicação.



Comunicação entre Servlets e páginas JSP

A comunicação do servlet com o JSP é realizada através dos métodos do protocolo HTTP get e post implementados no doGet e no doPost do Servlet como no exemplo abaixo.

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

// **TODO** Auto-generated method stub

//Gera uma lista de clientes

List clientes = repositorio.getTop10ByName();

//passo a listagem para a página JSP

request.setAttribute("clientes", clientes);

//chamar a página JSP

RequestDispatcher listagem = request.getRequestDispatcher("clientesListagem.jsp");

listagem.forward(request, response);

}

/\*\*

\* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

}

Exemplo da implementação na página JSP:

<%

List<Cliente> clientes = (List<Cliente>) request.getAttribute("clientes");

**if**(clientes != **null**){

%>

<table>

<tr>

<td>ID</td><td>Nome</td>

</tr>

<%

**for**(Cliente c : clientes){

%>

<tr>

<td><%=c.getId()%></td>

<td><%=c.getNome()%></td>

</tr>

<%

}

%>

</table>

<%

}

%>

Utilizando JPA em um projeto JSP

Para iniciar é preciso mapear o banco de dados criando a classe no pacote domainModel se tratando da arquitetura MVC, e assim fazer as devidas anotations para especificar as configurações do banco de dados na classe que representa sua estrutura.

package domainModel;

import java.util.Date;

import javax.persistence.\*;

@Entity

@Table(name="Clientes")

public class Cliente {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name="Codigo")

private int id;

@Column(name="Nome")

private String nome;

@Temporal(TemporalType.DATE)

@Column

private Date dataNascimento;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getNome() {

return nome;

}

public void setNome(String nome) {

this.nome = nome;

}

public void setDataNascimento(Date dataNascimento) {

this.dataNascimento = dataNascimento;

}

public Date getDataNascimento() {

return dataNascimento;

}

}

Após isso criamos a classe responsável por implementar as principais funções de banco de dados utilizando a classe acima implementada no domainModel. Essa nova classe deve ser implementada no pacote dataAccess.

package dataAccess;

import domainModel.Cliente;

import java.util.List;

import javax.persistence.\*;

public class ClienteRepository {

private EntityManager manager;

private EntityManagerFactory factory;

private EntityTransaction transaction;

public ClienteRepository() {

factory = Persistence.createEntityManagerFactory("Loja");

manager = factory.createEntityManager();

}

public void Save(Cliente obj) throws Exception {

try {

transaction = manager.getTransaction();

transaction.begin();

manager.persist(obj);

transaction.commit();

}

catch(Exception ex){

transaction.rollback();

throw ex;

}

}

public void Delete(Cliente obj) throws Exception {

try {

transaction = manager.getTransaction();

transaction.begin();

manager.remove(obj);

transaction.commit();

}

catch(Exception ex){

transaction.rollback();

throw ex;

}

}

public Cliente Open(int id) throws Exception {

try {

return manager.find(Cliente.class, id);

}

catch(Exception ex){

throw ex;

}

}

public List<Cliente> getTop10ByName() {

return manager.createQuery("select c from Cliente c order by c.nome", Cliente.class)

.setMaxResults(10).getResultList();

}

}

Agora deve-se criar a classe no pacote presentation onde essa nova classe executará as operações para realizar a comunicação das páginas JSP em si com a aplicação. Ela receberá dados da página jsp e de banco de dados, carregando o objeto com informações do banco de dados conforme a solicitação feita na página JSP e após isso retornar os dados para a devida página ou salvando dados no banco.

package presentation;

import domainModel.Cliente;

import dataAccess.ClienteRepository;

import java.io.IOException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.util.List;

/\*\*

\* Servlet implementation class clienteController

\*/

@WebServlet("/Clientes")

public class clienteController extends HttpServlet {

private static final long serialVersionUID = 1L;

// Declaração do repositório

ClienteRepository repositorio;

// Construtor do servlet

public clienteController() {

super();

// Inicialização do repositório

repositorio = new ClienteRepository();

}

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

//Verifica se o parâmetro edit foi passado

String edit = request.getParameter("edit");

//Código que executa quando o parâmetro edit é passado

if(edit != null){

if(!edit.equalsIgnoreCase("new")){

try {

// Carrega o cliente do BD

Cliente cliente = repositorio.Open(Integer.parseInt(edit));

// Passa o cliente para a página JSP

request.setAttribute("cliente", cliente);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

// Chamar página JSP

RequestDispatcher editar = request.getRequestDispatcher("clientesEditar.jsp");

editar.forward(request, response);

return;

}

// Gera uma listagem de clientes

List<Cliente> clientes = repositorio.getTop10ByName();

// Passa a listagem para a página JSP

request.setAttribute("clientes", clientes);

// Chamar a página JSP

RequestDispatcher listagem = request.getRequestDispatcher("clientesListagem.jsp");

listagem.forward(request, response);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

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

try {

// Recebe os parâmetros do formulário

String cod = request.getParameter("cod");

String nome = request.getParameter("nome");

// Carrega o objeto do banco de dados

Cliente cliente = repositorio.Open(Integer.parseInt(cod));

cliente.setNome(nome);

repositorio.Save(cliente);

// Gera uma listagem de clientes

List<Cliente> clientes = repositorio.getTop10ByName();

// Passa a listagem para a página JSP

request.setAttribute("clientes", clientes);

// Chamar a página JSP

RequestDispatcher listagem = request.getRequestDispatcher("clientesListagem.jsp");

listagem.forward(request, response);

}

catch(Exception ex){

}

}

}

Já no WebContent vamos implementar as páginas JSP, onde incluiremos código HTML para visualização no browser. Através de métodos do próprip protocolo HTTP pode –se assim enviar requisições de uma página para outra, então implementando o método doGet e doPost no controller localizado no pacote presentation podemos realizar essa interação entre a página JSP e a aplicação em si.

<?xml version="1.0" encoding="ISO-8859-1" ?>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ page import="domainModel.Cliente, java.util.List" %>

<!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=ISO-8859-1" />

<title>Listagem de Clientes</title>

</head>

<body>

<%

List<Cliente> clientes = (List<Cliente>) request.getAttribute("clientes");

if(clientes != null) {

%>

<table>

<tr><td>ID</td><td>Nome</td></tr>

<%

for(Cliente c : clientes) {

%>

<tr>

<td><%=c.getId() %></td>

<td><%=c.getNome() %></td>

<td><a href="/Loja/Clientes?edit=<%=c.getId() %>">Editar</a></td>

<td><a href="/Loja/Clientes?del=<%=c.getId() %>">Apagar</a></td>

</tr>

<%

}

%>

</table>

<%

}

%>

<a href="/Loja/Clientes?edit=new">Criar Novo Cliente</a>

</body>

</html>

Para o funcionamento do JPA não podemos esquecer de acrescentar as devidas configurações no persistence.xml. Abaixo encontra-se um exemplo do persistence utilizando banco MySql.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<persistence version=*"2.0"*

xmlns=*"http://java.sun.com/xml/ns/persistence"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd"*>

<persistence-unit name=*"Loja"*>

<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

<class>domainModel.Cliente</class>

<exclude-unlisted-classes>false</exclude-unlisted-classes>

<properties>

<property name=*"javax.persistence.jdbc.driver"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"javax.persistence.jdbc.url"* value=*"jdbc:mysql://localhost:3306/loja"*/>

<property name=*"javax.persistence.jdbc.user"* value=*"root"*/>

<property name=*"javax.persistence.jdbc.password"* value=*""*/>

</properties>

</persistence-unit>

</persistence>

Referências

Definições JSP Disponível em: http://www.criarweb.com/artigos/227.php. Acesso em 12/05/2011.

Arquitetura REST. Disponível em: http://projetos.inf.ufsc.br/arquivos\_projetos/projeto\_673/TCCRicardoGhisi.pdf. Acesso em 13/05/2011.

Arquitetura REST . Disponível em: http://pt.wikipedia.org/wiki/REST. Acesso em 13/05/2011

Arquitetura MVC. Disponível em: http://pt.wikipedia.org/wiki/MVC. Acesso em 15/05/2011

Fundamentos MVC. Disponível em: http://www.linhadecodigo.com.br/artigo/2367/Abordando-a-arquitetura-MVC-e-Design Patterns-Observer-Composite-Strategy.aspx. Acesso em 15/05/2011.

Fundamentos MVC. Disponível em: http://www.revistaphp.com.br/artigo.php?id=50. Acesso em 15/05/2011.

<http://www.devmedia.com.br/post-3573-Fundamentos-de-Servlets.html>

<http://www.arquivodecodigos.net/dicas/java-servlets-entendendo-a-classe-httpservlet-2338.html>

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

<http://www.htmlstaff.org/ver.php?id=1585>

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

<http://www.dei.isep.ipp.pt/~i020525/trabalhos/2ano-1sem-lpg2-tp1/jsp.pdf>  
<http://maniish.wordpress.com/2009/11/10/server-tags/>

<http://www.htmlstaff.org/ver.php?id=653>

<http://javafree.uol.com.br/viewtopic.jbb?t=1407>

<http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/jsf/jsf.htm>

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

<http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/jsp/strutsmvc.htm>

<http://docs.google.com/viewer?a=v&q=cache:1f5Q1rCm6RwJ:www.argonavis.com.br/cursos/java/j550/j550_15.pdf+como+implementar+arquitetura+mvc+jsp+utilizando+conceito+de+resst&hl=pt-BR&gl=br&pid=bl&srcid=ADGEEShwniEcI0Kii7CBcB0pLaLGdLTVJ6TJsN-yns7Ek3_iawTzfoHegts4LWD9P_dIbu7sNPMPz_qYY-BpA_QjhRs-UJI5eBnfPdZJzgsssLjWyvAsQuBsvKDMENAc_SxkZkVM-Xo7&sig=AHIEtbTVna2Hg2Tk1ZrpFQu2IlA29Gqvhw&pli=1>

<http://www.linhadecodigo.com.br/artigo/2059/Web-Services-REST.aspx>

<http://alexandregama.wordpress.com/tag/arquitetura/>