

1

Tópicos Abordados

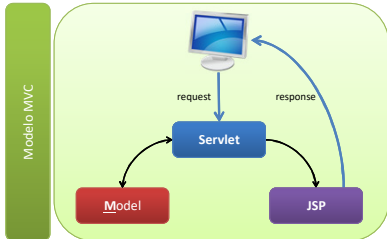
- O papel de cada componente
- Parâmetros e atributos
- Escopos dos atributos
 - Page
 - Request
 - Context
 - Session
- Sessão
 - Funcionamento
 - Cookies
 - URL rewriting
 - Tempo de vida
- Cookies
- Forward e redirect

2

[illegible]

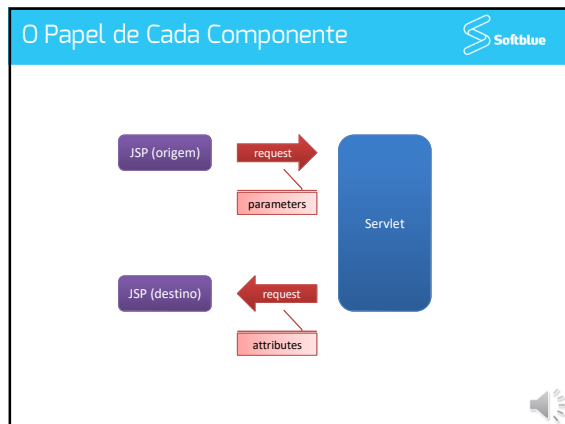
O Papel de Cada Componente

- Se um JSP é um servlet, quando usar um e quando usar outro?



O diagrama ilustra o Modelo MVC (Model-View-Controller) em um ambiente de desenvolvimento web. À esquerda, uma barra vertical verde rotulada "Modelo MVC" indica o contexto. No centro, um retângulo amarelo arredondado contém os componentes principais: um navegador (representado por um monitor) no topo, um retângulo azul rotulado "Servlet" no meio e dois retângulos na base: um vermelho rotulado "Model" e um verde rotulado "JSP". Uma seta azul desce do navegador ao Servlet, rotulada "request". Uma seta curva azul sobe do Servlet ao navegador, rotulada "response". Duas setas pretas apontam do Servlet para o Model e o JSP. Uma seta curva preta também aponta do JSP de volta ao navegador.

3



4

Parameters (Parâmetros)

- São fornecidos pelo HTTP através de requisições GET ou POST

<http://www.softblue.com.br/CadastroServlet?nome=Carlos&empresa=Softblue>

```
public class CadastroServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String nome = request.getParameter("nome");
        String empresa = request.getParameter("empresa");
        CadastroService service = new CadastroService();
        service.cadastrar(nome, empresa);
        ...
    }
}
```

O retorno é sempre uma string

5

Attributes (Atributos)

- São dados colocados numa área comum e podem ser compartilhados

```
public class CadastroServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        ...
        Registro reg = service.cadastrar(nome, empresa);
        request.setAttribute("r", reg);
        request.getRequestDispatcher("sucesso.jsp").forward(request, response);
    }
}
```

O atributo deve possuir um nome e pode ser de qualquer tipo

Redirecionamento para um JSP de destino

6

Attributes (Atributos)

sucesso.jsp

```

<html>
<body>

<%
  Registro reg = (Registro) request.getAttribute("r");
%>

<H1>Cadastro feito com sucesso!</H1>
<H2>O ID do cadastro é: <%= reg.getId() %></H2>

</body>
</html>

```

request é um objeto implícito

7

Escopos dos Atributos

- A visibilidade dos atributos depende do seu escopo

Escopo	Descrição
page	Apenas a página que define o atributo o enxerga. Usado em JSPs
request	Atributos visíveis apenas enquanto durar a requisição
context	Atributos compartilhados com todos os servlets e JSPs da aplicação
session	Atributos visíveis apenas a um cliente específico

8

O Escopo Page

- Usado em JSPs
- Um atributo é definido no JSP e só é visível dentro do JSP

```

<%
  pageContext.setAttribute("nome", "Carlos");
%>

<html>
<body>
<H1>Nome: <%= pageContext.getAttribute("nome") %></H1>
</body>
</html>

```

Define o atributo no escopo page

Lê o atributo

9

O Escopo Request



- Atributos na request duram apenas enquanto durar a requisição

```
Servlet
...
Registro reg = service.cadastrar(nome, empresa);
request.setAttribute("r", reg);
request.getRequestDispatcher("sucesso.jsp").forward(request, response);
...

sucesso.jsp
<%
  Registro reg = (Registro) request.getAttribute("r");
%>
```

10

O Escopo Context



- Os atributos são compartilhados por todos os servlets e JSPs da aplicação

```
Servlet
ServletContext ctx = getServletContext();
ctx.setAttribute("versao", "1.0");
String versao = (String) ctx.getAttribute("versao");

JSP
<%
  application.setAttribute("versao", "1.0");
%>
<%= application.getAttribute("versao") %>
```

11

O Escopo Session



- Os atributos são exclusivos de um cliente da aplicação
- Outros clientes que usam a aplicação não têm acesso aos atributos
- A session define um estado de conversação entre um cliente específico e um servidor
 - O servidor reconhece as múltiplas requisições do cliente

12

Acesso Concorrente aos Atributos



- Lembre-se que em um ambiente web existem múltiplos clientes fazendo acessos simultâneos
- Apenas o escopo **request** é thread-safe

```
ServletContext ctx = getServletContext();  
  
synchronized (ctx) {  
    ctx.setAttribute("versao", "1.0");  
}
```



13

Estado de Conversação



- Algumas aplicações exigem que o servidor saiba quem é o cliente com o qual ele está lidando
 - Várias telas de cadastro
 - Carrinho de compras
 - Login de usuário
 - etc.
- O problema é que o protocolo HTTP não armazena estado



14

Sessão



- Na camada web, a forma de fazer isto é usando uma **session** (sessão)
 - Representada pela interface **HttpSession**

```
Servlet  
  
HttpSession session = request.getSession();  
session.setAttribute("usuario", "usu1010");  
String u = (String) session.getAttribute("usuario");  
  
JSP  
  
<%  
    session.setAttribute("usuario", "usu1010");  
%>  
  
<%= session.getAttribute("usuario") %>
```



15

Sessão



- A sessão é única para cada cliente
- Não existe a possibilidade de outro cliente ler ou escrever dados na sessão de outro cliente
- Se você estiver acessando uma aplicação e um atributo for colocado na sua sessão, numa requisição futura a aplicação poderá ler este atributo

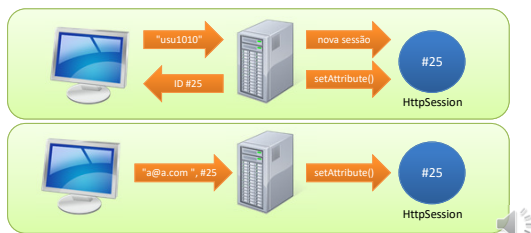


16

Funcionamento da Sessão



- Como o servidor sabe quem é o cliente?
- O servidor gera um ID único por cliente



17

Funcionamento da Sessão



- Para que o processo funcione, é necessário que cliente e servidor troquem a informação de ID da sessão
- Isto pode ser feito de duas formas
 - Cookies
 - URL rewriting



18

Sessão por Cookies



- Cookies são informações enviadas pelo servidor e guardadas no cliente
- O servidor envia um cookie para o cliente e este o armazena
- A cada requisição, o cliente reenvia o cookie ao servidor
- Os cookies vão dentro do cabeçalho do protocolo HTTP



19

Sessão por URL Rewriting



- Usada normalmente quando o cliente não aceita cookies
- A cada URL chamada, o cliente reenvia o ID da sessão na URL

<http://www.softblue.com.br/Comprar?sessionId=2379FD2>

O container adiciona o ID da sessão à URL



20

Tempo de Vida da Sessão



- Pode ser que o cliente tenha seus dados colocados na sessão e pare o processo no meio
- Para evitar o gasto desnecessário de recursos, a sessão é destruída depois de um tempo
 - O tempo pode ser configurado no web.xml
- A sessão também pode ser destruída via programação
 - Método **invalidate()** da interface HttpSession
- Uma boa prática é remover os atributos da sessão que não são mais necessários
 - Método **removeAttribute()** da interface HttpSession



21

Cookies



- Os cookies são pares de chave e valor armazenados no cliente
- Por padrão eles desaparecem assim que o browser é fechado, mas é possível configurá-los para que eles durem mais tempo



22

Trabalhando com Cookies



- O servidor é quem cria o cookie
- Quando o cliente faz uma requisição, ele envia seus cookies ao servidor
- O servidor pode procurar pelo cookie desejado

```
Cookie cookie = new Cookie("nomeCliente", "Carlos");  
cookie.setMaxAge(30 * 60);  
response.addCookie(cookie);
```

```
Cookie[] cookies = request.getCookies();  
for (Cookie cookie : cookies) {  
    if(cookie.getName().equals("nomeCliente")) {  
        String nome = cookie.getValue();  
    }  
}
```



23

Redirecionamento




- Após um servlet terminar sua tarefa, ele normalmente redireciona a requisição para um arquivo JSP, que será mostrado como resultado para o cliente
- Este redirecionamento pode ser feito de duas formas:
 - Forward
 - Redirect



24


Forward



- O forward é um redirecionamento interno
 - O próprio container faz o redirecionamento
 - Olhando a URL, não é possível determinar para onde ocorreu o redirecionamento


```
request.getRequestDispatcher("sucesso.jsp").forward(request, response);
```

Forward para sucesso.jsp



25

Redirect




- O redirect é um redirecionamento para outra URL
 - O container solicita ao browser a requisição de uma nova URL
 - Olhando a URL, é possível ver para onde ocorreu o redirecionamento

```
response.sendRedirect("sucesso.jsp");
```


Redirect para sucesso.jsp

Ao fazer um redirect, todos os dados da request são perdidos




26

Forward ou Redirect?



- O que normalmente ocorre na prática é que o **redirect** é utilizado após alguma operação que salva, atualiza ou exclui dados da aplicação
 - O redirect evita que o cliente pressione F5 no browser e o processo seja executado de novo
- Nas outras situações, normalmente o **forward** é utilizado
 - Se o cliente pressionar F5 no browser o processo é executado de novo, mas isto não traz problemas



27
