

softtech

NETWORK



Curso de Java para Web

Curso de Java para Web

Daniel Destro do Carmo
Softech Network Informática
daniel@danieldestro.com.br

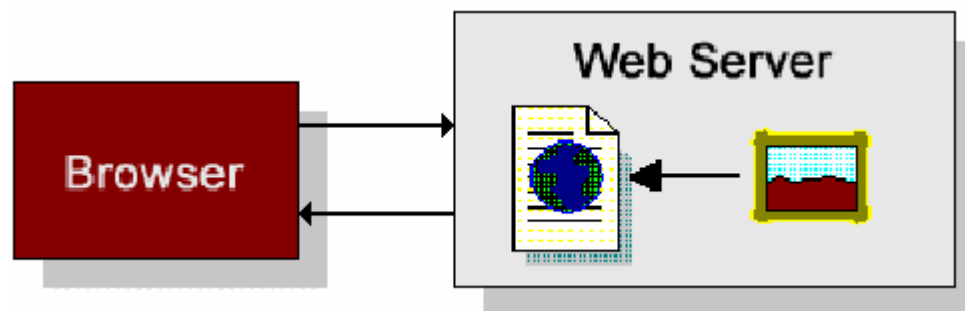
Curso de Java para Web

A Internet e o Protocolo HTTP

As aplicações desenvolvidas para internet são, em sua grande maioria, acessadas via os navegadores web (web browsers), o que quer dizer que elas utilizam o protocolo HTTP (ou HTTPS – HTTP sobre SSL – Secure Socket Layer) para comunicação e o tráfego de dados na rede.

Este protocolo é baseado no modelo de solicitação e resposta. O cliente é quem sempre faz uma solicitação a um servidor, que, por sua vez, processa e gera uma resposta de volta ao cliente.

Neste ponto a conexão entre cliente e servidor é fechada, ou seja, não existe uma sessão permanente entre esses dois pontos de comunicação. O cliente abre uma conexão com o servidor, envia a requisição, recebe a resposta e fecha a conexão.



HTTP = Hyper Text Transfer Protocol

Curso de Java para Web

Requisição e Resposta HTTP

Todo comunicação com o servidor se inicia com uma solicitação (requisição) do cliente (navegador) ao servidor. Após o servidor processar a solicitação, ele devolve uma resposta ao cliente. A solicitação é iniciada quando uma URL é digitada na barra de endereços, um link é clicado ou um form é submetido.

Exemplo de solicitação HTTP:

```
GET      /servlet/MeuServlet      HTTP/1.1
Accept:  text/plain; text/html
Accpet-Language: pt-br
Connection: Keep-Alive
Host: localhost
Referer: http://localhost/paginaTeste.htm
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows XP)
Content-Length: 33
Content-Type: application/x-www-form-urlencoded
Accept-Encodig: gzip, deflate

Nome=Joao&Sobrenome=da%20Silva
```

Tipos de solicitação: POST, GET, HEAD, PUT, DELETE e TRACE.

Curso de Java para Web

Requisição e Resposta HTTP

Exemplo de resposta HTTP:

```
HTTP/1.1      200      OK
Server: Microsoft-IIS/4.0
Date: Mon, 20 Jan 2004 13:00:00 GMT
Content-Type: text/html
Last-Modified: Mon, 21 Jan 2004 13:33:00 GMT
Content-Length: 85

<html>
<head>
  <title>Exemplo de resposta http</title>
</head>
<body></body>
</html>
```

O conteúdo vem na própria resposta da solicitação, geralmente em forma de código HTML, como o exemplo acima.

Curso de Java para Web

Requisição e Resposta HTTP

Os dois principais, e mais usados, tipos de requisições são o GET e POST. Eles possuem algumas diferenças básicas, porém muito importantes:

GET

- Só pode enviar até 255 caracteres de informações
- As informações vão como parte da URL (não indicado para senha)
- O browser ou proxy faz o cache da página pela URL
- Feito quando uma URL é digitada, via um link ou por um form de método GET

POST

- Pode enviar conteúdo ilimitado de informações
- Pode enviar texto e binário (ex: arquivos)
- O browser ou proxy não fazem o cache da página pela URL
- Feito por um form de método POST

Curso de Java para Web

HTML

HTML (Hyper Text Markup Language) é o código interno de uma página na internet. É uma linguagem marcada por tags. É bem simples!

O navegador interpreta estas tags do HTML e processa uma resposta visual ao internauta, que é a página HTML.

```
<html>
<head>
  <title>Exemplo de página html</title>
</head>
<body>
  Este texto vai aparecer no navegador. <br>
  <a href="http://www.uol.com.br">Isto é um link, clique!</a>
  <table>
    <tr><td>Texto na célula de uma tabela</td></tr>
  </table>
  <br>
  
  <br><h1>Texto grande</h1>
  <font size="2" color="red"> ou normal</font>
</body>
</html>
```

Curso de Java para Web

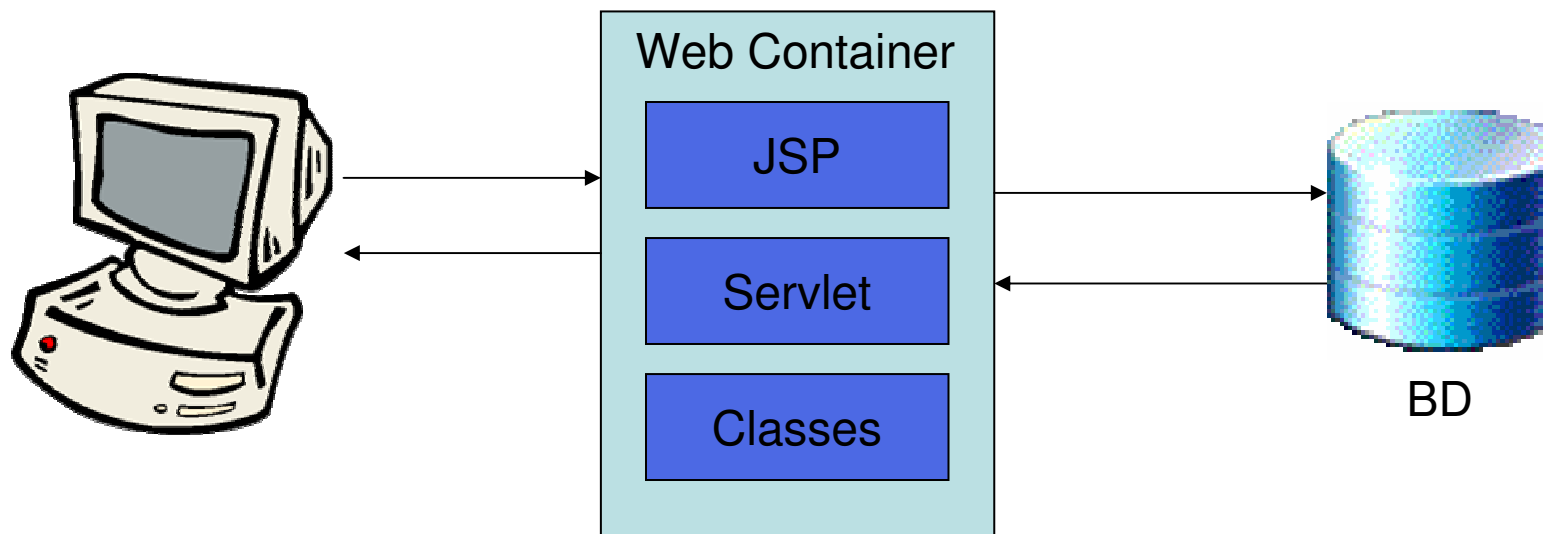
Programação Java para Web

A tecnologia Java nos permite escrever aplicações robustas e seguras para rodar no ambiente internet. Isto é possível através da tecnologia de Java Servlets e JavaServer Pages (JSP).

Servlets são classes Java que atendem às requisições HTTP.

JSP são documentos HTML com código Java embutido. Eles são usados, principalmente, como interface visual com o cliente web.

A arquitetura das aplicações web seguem, geralmente, o modelo de 3 camadas, como abaixo:



Curso de Java para Web

Servidor Java

O Servidor Java é o local onde ficam armazenados os Servlets, JSPs e as classes de negócio da sua aplicação.

O Servidor Java atende às solicitações feitas a ele, invocando os recursos solicitados, como os Servlets, JSPs, HTMLs, imagens e etc, assim como um webserver, porém estendido a funcionalidade dos webserver, servindo as aplicações em Java.

Esses servidores são, geralmente, conhecidos como Servlet Containers, ou Web Containers.

Existem vários Servidores Java, um deles, muito bom e gratuito, é o Apache Tomcat.

Outros servidores mais completos que implementam toda a especificação J2EE podem ser encontrados. Alguns deles: JBoss, IBM Web Sphere, Bea Web Logic, Oracle OC4J, e outros.

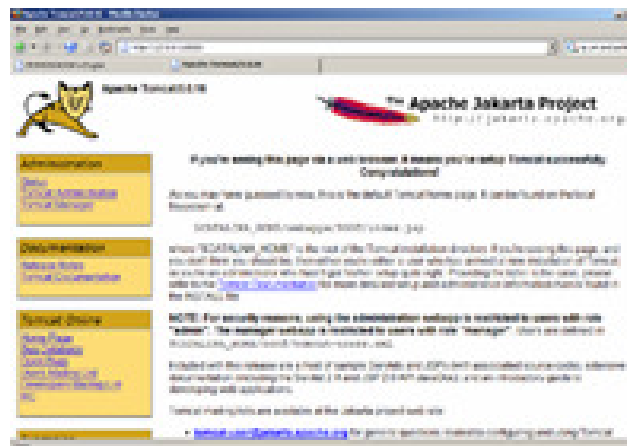


Curso de Java para Web

Instalando o Servidor Apache Tomcat

1. Certifique-se que você tenha o JDK (J2SE) instalado na sua máquina.
2. Faça o download dos binários da última versão (5) do Apache Tomcat no próprio site da Apache. O binário vem empacotado em um arquivo zip.
3. Descompacte o zip no diretório onde deseja ter o Tomcat instalado (ex: "C:\").
4. Edite o arquivo catalina.bat (catalina.sh, no Linux), que se encontra no diretório bin, adicionando a linha abaixo (no Linux omita a palavra SET):
SET JAVA_HOME=C:\Caminho-do-seu-jdk

Pronto! Execute o arquivo *startup.bat*. Seu Tomcat já está instalado e configurado para rodar suas primeiras aplicações web. Acesse: <http://localhost:8080> ou <http://127.0.0.1:8080>



* Você pode configurar a porta padrão (8080) para outra porta, por exemplo a porta 80, editando o arquivo TOMCAT/config/server.xml.

Curso de Java para Web

Primeiro Servlet

Arquivo: web\PrimeiroServlet.java

```
package web;

import java.io.*;
import javax.servlet.*;

public class PrimeiroServlet extends GenericServlet {
    public void service(ServletRequest req, ServletResponse res)
        throws IOException, ServletException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Meu Primeiro Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("Teste do meu primeiro servlet!");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Curso de Java para Web

Compilando o Primeiro Servlet

Como podemos ver no código anterior, o nosso PrimeiroServlet nada mais é do que uma classe Java que estende a classe *javax.servlet.GenericServlet*. *GenericServlet* é uma classe abstrata básica que implementa a interface *javax.servlet.Servlet* e define o método abstrato *service()*, que deve ser implementado por suas subclasses, para definir o código de execução do serviço do servlet.

Para criarmos os Servlets, necessitamos de classes do pacote *javax.servlet* e *javax.servlet.http*. Essas classes pertencem à API Servlet do Java, que não faz parte do J2SE, mas sim do J2EE. O Tomcat já vem com esse pacote, porém podemos instalar o J2EE, se for o caso.

Para compilarmos o código, precisamos incluir no classpath o arquivo *servlet-api.jar* que fica no diretório TOMCAT\common\lib\. Uma vez incluído, agora sim podemos compilar o código.

```
javac -classpath C:\jakarta-tomcat-5.0.16\common\lib\servlet-api.jar web\PrimeiroServlet.java
```

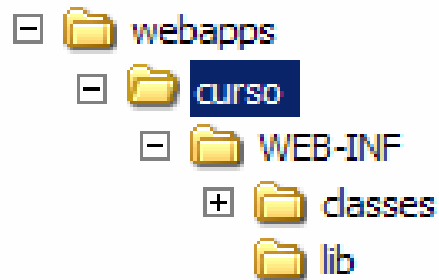
Um arquivo .class será gerado normalmente, como em qualquer classe java compilada.

Curso de Java para Web

Configurando o Primeiro Servlet

Agora que já temos a classe servlet compilada, podemos pôr o servlet pra funcionar no servidor web. A esta tarefa, damos o nome de *deploy*.

O deploy se dá da seguinte maneira: primeiro devemos criar um diretório com o nome da nossa aplicação, por exemplo *curso*. Este diretório deve ficar dentro do diretório TOMCAT\webapps\. Cada diretório dentro de *webapps* é um aplicativo web diferente. Um aplicativo pode conter dezenas de Servlets e outros recursos, como JSP, HTML, etc. Os recursos como JSP, HTML, imagens e outros ficam localizados na pasta raiz do aplicativo ou em sub-pastas. Dentro do diretório *curso* devemos criar outros diretórios, seguindo a estrutura abaixo:



O diretório *WEB-INF* vai conter os arquivos de configuração da sua aplicação e os arquivos de *deploy*. O diretório *classes* deve conter as classes do seu aplicativo. No diretório *lib* vão os jars (libs) da sua aplicação. As suas classes, ao invés de desagrupadas, poderiam estar empacotadas em um jar e ir neste diretório também. Os arquivos como imagens, HTML e outros devem ir no diretório raiz, *curso*.

Tendo colocado o arquivo *PrimeiroServlet.class*, dentro de *curso\classes\web*, agora devemos criar um arquivo de configuração para ele, o chamado *deployment descriptor*, ou apenas *web.xml*. O arquivo *web.xml* é um xml que descreve as configurações de cada aplicação web do Web Container, e contém informações dos Servlets da aplicação e outras configurações. Ele deve ser criado no diretório *WEB-INF*.

Curso de Java para Web

Deployment Descriptor (web.xml)

Os arquivos XML também são marcados por tags. Essas tags podem ser definidas por você mesmo, e não apenas usarem tags pré-definidas, como no HTML. As tags definem a estruturação dos dados. O arquivo web.xml deve ser escrito usando as tags já definidas pela especificação dos Servlets da J2EE.

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

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <description>Aplicação do Curso de Java</description>
  <display-name>Curso de Java</display-name>

  <servlet>
    <servlet-name>PrimeiroServlet</servlet-name>
    <servlet-class>web.PrimeiroServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>PrimeiroServlet</servlet-name>
    <url-pattern>/PrimeiroServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Curso de Java para Web

Executando o Primeiro Servlet

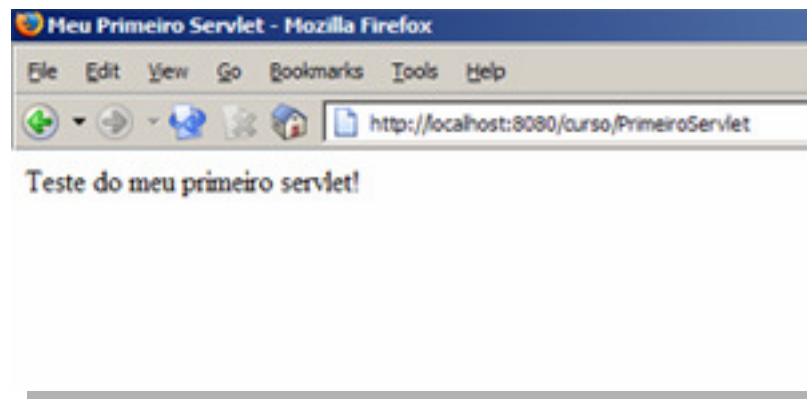
Finalmente podemos executar o nosso Primeiro Servlet, que apenas vai gerar uma simples mensagem na tela do navegador.

Após termos configurado o *web.xml* e disponibilizado os arquivos corretamente, devemos reiniciar o Tomcat.

Agora, então, podemos chamar, ou solicitar, o nosso Primeiro Servlet. Para isso, abra um navegador e digite a seguinte URL:

<http://localhost:8080/curso/PrimeiroServlet>

A seguinte tela deverá aparecer:



Curso de Java para Web

Entendendo os Servlets

Todo o suporte a servlets é provido pelos pacotes *javax.servlet* e *javax.servlet.http*. Eles contêm classes e interfaces que são muito importantes para o entendimento geral de suas funcionalidades. São elas:

Pacote: *javax.servlet* – Este pacote é dos servlets genéricos, independente de protocolo.

- Servlet
- GenericServlet
- ServletRequest
- ServletResponse
- ServletContext
- ServletConfig
- RequestDispatcher
- ServletException
- SingleThreadModel
- ServletOutputStream
- ServletInputStream
- ServletContextListener
- ServletContextAttributeListener
- UnavailableException
- ServletContextEvent

- ServletContextAttributeEvent
- Filter
- FilterConfig
- FilterChain

Pacote: *javax.servlet.http* – Estende a funcionalidade do pacote *javax.servlet* para os servlets do protocolo http.

- HttpServlet
- HttpServletRequest
- HttpServletResponse
- HttpSession
- Cookie
- HttpSessionListener
- HttpSessionAttributeListener
- HttpSessionEvent
- HttpSessionBindingEvent

Curso de Java para Web

javax.servlet.Servlet

A interface *javax.servlet.Servlet* é a fonte de toda a programação Servlet, é a abstração central desta tecnologia. Todo servlet deve implementar esta interface, direta ou indiretamente.

Esta interface possui 5 métodos:

init(ServletConfig conf) – Chamado pelo Servlet Container, para iniciar o servlet.

service(ServletRequest req, ServletResponse res) – Chamado pelo Servlet Container, para o servlet responder às suas solicitações. É onde o servlet vai interagir com as requisições, fazer o processamento e gerar resposta.

destroy() – Chamado pelo Servlet Container, no momento da destruição do servlet, para limpar os recursos (cleanup).

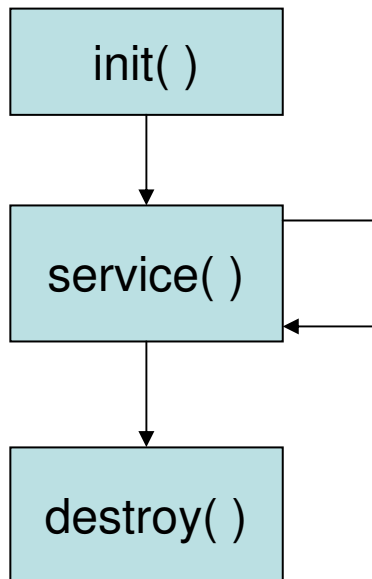
ServletConfig getServletConfig() – Retorna um objeto ServletConfig, que contém as configurações do servlet.

String getServletInfo() – A implementação deste método deve retornar informações sobre o servlet.

Curso de Java para Web

Ciclo de vida dos Servlets

Os servlets possuem um ciclo de vida bem definido, que é gerenciado pelo Servlet Container. Entender este ciclo de vida é muito importante.



O método *init()* é executado apenas uma vez, quando o servlet é carregado pelo Servlet Container. Só então o servlet fica apto a responder às solicitações.

```
public void init( ServletConfig config ) throws ServletException
```

O método *service()* é chamado pelo Servlet Container toda vez que o servlet recebe uma solicitação.

```
public void service( ServletRequest req, ServletResponse res )  
throws ServletException, java.io.IOException
```

O método *destroy()* é chamado pelo container na destruição do servlet, seja quando uma instância for descarregada ou quando o container for desligado.

```
public void destroy()
```

Curso de Java para Web

O método *init()*

O método *init()*, definido na interface *javax.servlet.Servlet* é executado apenas quando o Servlet é carregado, ou seja, no carregamento inicial do container, ou quando o container cria novas instâncias do Servlet.

Cada container pode implementar suporte a múltiplas instâncias de um servlet ao mesmo tempo, ao invés de usar uma única instância para responder às múltiplas solicitações.

O método recebe um parâmetro do tipo *ServletConfig*, que é a classe que contém as configurações do servlet, definidas no *deployment descriptor*, o *web.xml*. Uma referência ao objeto *ServletConfig* é mantida pela implementação do método da classe *GenericServlet*.

Curso de Java para Web

O método *service()*

Este é o método que atende às solicitações feitas aos servlets. O método recebe dois parâmetros importantes, o `ServletRequest` e o `ServletResponse`. Eles representam a requisição feita ao servlet e a resposta gerada ao cliente, respectivamente.

O método *service()* é o ponto em que deveremos codificar a execução do servlet.

A interação com o cliente é feita através dos parâmetros que ele recebe (request e response). A classe `ServletRequest` contém métodos para se extrair informações vindas do cliente na hora da requisição, como parâmetros e informações relativas ao cliente.

Com a classe `ServletResponse`, podemos nos comunicar com o cliente, informando os dados da resposta ao cliente, como o HTML gerado ou outra informação. Essa resposta é enviada por meio de streams.

Curso de Java para Web

Implementando um servlet simples

```
package web;
import javax.servlet.*;

public class ServletSimples implements Servlet {
    private ServletConfig servletConfig;

    public void init( ServletConfig config ) throws ServletException {
        System.out.println("Servlet.init()");
        this.servletConfig = config; //guarda a referência para o ServletConfig
    }

    public void service( ServletRequest req, ServletResponse res )
        throws ServletException, java.io.IOException {
        System.out.println("Servlet.service()");
    }

    public void destroy() {
        System.out.println("Servlet.destroy()");
    }

    public ServletConfig getServletConfig() {
        return this.servletConfig;
    }

    public String getServletInfo() {
        return "Servlet Simples";
    }
}
```

Curso de Java para Web

javax.servlet.ServletConfig

A interface ServletConfig representa as configurações do servlet, feitas no *web.xml*, e é bem simples de se usar. Esta interface possui os métodos:

String getInitParameter(String name) – Retorna o valor do parâmetro indicado por *name* ou *null* se o parâmetro não existe na configuração.

Enumeration getInitParameterNames() – Retorna um Enumeration de String com os nomes de todos os parâmetros configurados.

ServletContext getServletContext() – Retornar o ServletContext da aplicação do servlet.

String getServletName() – Retorna o nome do servlet especificado na configuração.

Esta interface permite apenas que se recuperem os valores, mas não permite que os alterem.

As configurações ficam localizadas no *deployment descriptor*, o arquivo *web.xml*. Estes valores ficam dentro da tag `<init-param>`, que está dentro da configuração do `<servlet>`.

Curso de Java para Web

javax.servlet.ServletConfig

Configurando o arquivo web.xml, com os parâmetros iniciais do servlet:

```
<web-app>
  <servlet>
    <servlet-name>PrimeiroServlet</servlet-name>
    <servlet-class>web.PrimeiroServlet</servlet-class>
    <init-param>
      <param-name>PARAM1</param-name>
      <param-value>VALOR1</param-value>
    </init-param>
    <init-param>
      <param-name>PARAM2</param-name>
      <param-value>VALOR2</param-value>
    </init-param>
  </servlet>
</web-app>
```

No método *init()* podemos, por exemplo, listar os parâmetros configurados para o servlet:

```
public void init( ServletConfig config ) throws ServletException {
    Enumeration enum = config.getInitParameterNames();
    while( enum.hasMoreElements() ) {
        String param = (String) enum.nextElement();
        System.out.println( param + ": " + config.getInitParameter(param) );
    }
}
```

Curso de Java para Web

javax.servlet.ServletContext

O ServletContext é a interface que representa o ambiente da aplicação. Cada aplicação web possui apenas um ServletContext.

Esta interface possui métodos para pegar as configurações da aplicação, informações do servidor, fazer log, acessar recursos da aplicação e mais outras funcionalidades que veremos no decorrer do curso.

O ServletContext também serve para compartilhar informações com toda a aplicação.

```
Enumeration enum = getServletContext().getInitParameterNames();
while( enum.hasMoreElements() ) {
    String param = (String) enum.nextElement();
    System.out.println( param + ": " + config.getInitParameter(param) );
}

InputStream is = getServletContext().getResourceAsStream("/WEB-INF/arq.txt");
//ler dados do arquivo pelo InputStream
is.close();

getServletContext().log( "Mensagem a ser gravada no log da aplicação" );
getServletConfig().getServletContext(); //outra maneira de pegar o contexto
```

Curso de Java para Web

javax.servlet.ServletContext

Os objetos guardados no ServletContext estarão disponíveis para toda a aplicação, ou seja, esses dados ficarão armazenados no escopo da aplicação. Porém, muito cuidado, pois como o ServletContext é compartilhado por toda a aplicação, ele não é um bom lugar para se guardar dados relativos a um cliente específico.

```
package web;
import javax.servlet.*;

public class GuardaValorServlet extends GenericServlet {
    private static int i = 0;

    public void service( ServletRequest req, ServletResponse res )
        throws ServletException, java.io.IOException {
        getServletContext().setAttribute("contador", new Integer(++i));
    }
}
```

```
package web;
import javax.servlet.*;

public class MostraValorServlet extends GenericServlet {
    public void service(ServletRequest req, ServletResponse res )
        throws ServletException, java.io.IOException {
        res.getWriter().println( getServletContext().getAttribute("contador") );
    }
}
```

Curso de Java para Web

javax.servlet.ServletRequest

A interface `ServletRequest` encapsula e manipula o request feito pelo cliente web. Ela oferece métodos para extrair os dados vindos na solicitação.

Esta classe serve os servlets de qualquer protocolo. Porém vamos nos aprofundar somente no protocolo HTTP, específico da web. Portanto, estudaremos os recursos desta classe e os recursos estendidos para o protocolo HTTP na interface `javax.servlet.http.HttpServletRequest`.

Curso de Java para Web

javax.servlet.ServletResponse

A interface `ServletResponse` encapsula e manipula a resposta enviada ao cliente web. Ela oferece métodos para gravar dados na resposta.

Esta classe serve os servlets de qualquer protocolo. Porém vamos nos aprofundar somente no protocolo HTTP, específico da web. Portanto, estudaremos os recursos desta classe e os recursos estendidos para o protocolo HTTP na interface `javax.servlet.http.HttpServletResponse`.

Curso de Java para Web

javax.servlet.http.HttpServlet

HttpServlet é uma classe abstrata que estende a classe *javax.servlet.GenericServlet*, e que implementa o método *service()*, além de adicionar um novo método com a assinatura abaixo:

```
protected void service( HttpServletRequest req, HttpServletResponse ) throws  
ServletException, java.io.IOException;
```

Perceba que, diferente de *javax.servlet.Servlet*, o método *service()* agora recebe dois novos parâmetros: *HttpServletRequest* e *HttpServletResponse*, que são as classes que representam o request e o response do protocolo http, respectivamente. Este método não é abstrato.

A classe *HttpServlet* estende as funcionalidade de *GenericServlet* para o protocolo HTTP. A classe adicionou alguns métodos próprios para atender aos diferentes tipos de solicitação:

Método do HttpServlet	Método HTTP
doPost()	POST
doGet()	GET
doHead()	HEAD
doPut()	PUT
doDelete()	DELETE
doTrace()	TRACE
doOptions()	OPTIONS

Todos os métodos recebem dois argumentos (*HttpServletRequest* e *HttpServletResponse*). O método *service()* do *HttpServlet* analisa o tipo de requisição e direcionada a chamada ao método mais apropriado.

Curso de Java para Web

javax.servlet.http.HttpServlet

Exemplificando:

```
package web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TesteServlet extends HttpServlet {
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("Método doPost( )");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("Método doGet( )");
    }
}
```

Curso de Java para Web

javax.servlet.http.HttpServlet

Para testarmos o nosso TesteServlet e a chamada aos diferentes métodos, vamos criar um HTML que faz os dois métodos de chamada a ele, GET e POST. Crie o arquivo teste.htm na pasta *curso*, e codifique-o assim:

```
<html>
<head>
  <title>Teste</title>
</head>
<body>
<a href="TesteServlet">Chamar TesteServlet via GET</a>
<br><br>
<form name="teste" method="POST" action="TesteServlet">
  <input type="submit" name="ok" value="Chamar via POST">
</form>
</body>
</html>
```

A página exibirá um link e um botão de formulário. Quando clicar no link, o navegador vai fazer uma requisição GET. Quando clicar no botão do form o navegador vai fazer uma requisição POST. Faça os testes e verifique a saída gerada.

Se o atributo *method* da tag *form* for mudado para GET, o form fará uma requisição GET.

Curso de Java para Web

Requisição e Resposta do Servlet

As requisições e as respostas são tudo do que se tratam os servlets, afinal os servlets atendem às requisições, gerando respostas.

Para as aplicações web do protocolo HTTP, existem duas classe:

javax.servlet.http.HttpServletRequest e *javax.servlet.http.HttpServletResponse*, que representam a requisição e resposta, respectivamente. Essas classes estendem as classes *javax.servlet.ServletRequest* e *javax.servlet.ServletResponse*, adicionando funcionalidade extra para os Servlets HTTP.

Os parâmetros podem ser enviados das seguintes maneiras: via query string da URL ou via formulário.

Query String da URL:

<http://localhost:8080/curso/SeuServlet?param1=valor1¶m2=valor2¶m2=valor2.1>

A query string inicia após o nome do recurso, seguido do caracter ?. Cada parâmetro é separado pelo caracter &. Deve ser informado o valor do parâmetro junto ao nome dele, separado pelo caracter =, formando um dado do tipo chave=valor. Um mesmo parâmetro pode conter mais de um valor, repetindo a chave na query string.

Caracteres especiais (ex: espaço, &) devem ser mapeados para o formato URL.

Curso de Java para Web

Requisição e Resposta do Servlet

Envio de parâmetros via formulário HTML:

```
<html>
<head>
  <title>Teste Form</title>
</head>
<body>
<form name="teste" method="POST" action="SeuServlet">
  <b>Nome:</b> <input type="text" name="nome"> <br>
  <b>Senha:</b> <input type="password" name="senha"> <br>
  <b>Sexo:</b> <input type="radio" name="sexo" value="M"> Masc.
  <input type="radio" name="sexo" value="F"> Fem. <br>
  <b>Hobbies:</b><br>
  <input type="checkbox" name="hobbie" value="Cinema"> Cinema <br>
  <input type="checkbox" name="hobbie" value="Futebol"> Futebol <br>
  <input type="checkbox" name="hobbie" value="Música"> Musica <br>
  <input type="submit" name="enviar" value="Enviar">
  &nbsp;
  <input type="reset" name="limpar" value="Limpar">
</form>
</body>
</html>
```

Curso de Java para Web

javax.servlet.http.HttpServletRequest

A interface `HttpServletRequest` representa e manipula a requisição do cliente, oferecendo os métodos da superclasse `ServletRequest`, além de adicionar suporte a cookies e sessão.

Esta interface define os seguintes métodos para recuperar o valor dos parâmetros passados:

String `getParameter(String name)` – Retorna o valor do parâmetro informado ou *null* se o valor não foi informado ou não existe.

String[] `getParameterValues(String name)` – Retorna um array de Strings caso o parâmetro tenha múltiplos valores.

Enumeration `getParameterNames()` – Retorna um enumeration com os nomes de todos os parâmetros enviados.

String `getHeader(String name)` – Retorna o valor do cabeçalho (header) enviado.

Enumeration `getHeaders(String name)` – Retorna um Enumeration com os valores do cabeçalho.

Enumeration `getHeaderNames()` – Retorna um Enumeration com os nomes de todos os cabeçalhos.

Estes métodos são comuns com a classe `ServletRequest`, portanto, podem ser usados para qualquer outro protocolo além do HTTP.

Curso de Java para Web

javax.servlet.http.HttpServletRequest

Recuperando os parâmetros enviados no request:

```
package web;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TesteRequestServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, java.io.IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("NOME:" + req.getParameter("nome") );
        Enumeration e = req.getParameterNames();
        while( e.hasMoreElements() ) {
            String param = (String) e.nextElement();
            out.println( param + ": " + req.getParameter(param) + "<BR>" );
        }
    }
}
```

Faça o deploy do servlet e em seguida chame a seguinte URL:

<http://localhost:8080/curso/TesteRequestServlet?nome=João&sobrenome=Silva>

Agora faça o teste passando os parâmetros via o formulário HTML exemplificado anteriormente.

Curso de Java para Web

javax.servlet.http.HttpServletRequest

Recuperando os cabeçalhos enviados no request:

```
package web;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TesteHeaderServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, java.io.IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("User-Agent:" + req.getHeader("User-Agent") );

        Enumeration e = req.getHeaderNames();
        while( e.hasMoreElements() ) {
            String header = (String) e.nextElement();
            out.println( header + ": " + req.getHeader( header ) + "<br>" );
        }
    }
}
```

Curso de Java para Web

javax.servlet.http.HttpServletResponse

A interface HttpServletResponse representa e manipula a resposta ao cliente, oferecendo os métodos da superclasse ServletResponse, além de adicionar métodos adicionais.

Esta interface define os seguintes métodos:

setContentType(String type) – Indica o tipo de resposta a ser enviado. Este método, quando usado, deve ser usado antes de enviar qualquer resposta ao cliente.

java.io.PrintWriter getWriter() – Retorna um objeto PrintWriter, que será usado para enviar resposta ao cliente, geralmente na forma de texto, html ou xml.

java.io.OutputStream getOutputStream() – Retorna um objeto OutputStream, que será usado para enviar resposta binária ao cliente. Pode gerar uma imagem ou outro documento e enviar os bytes dele.

setHeader(String name, String value) – Define um novo par nome/valor para um cabeçalho.

addHeader(String name, String value) – Adiciona um par nome/valor para um cabeçalho.

boolean containsHeader(String name) – Indica se o cabeçalho já existe.

setStatus(int code) – Define o código da resposta.

setStatus(int code, String msg) – Define o código e a mensagem da resposta.

sendRedirect(String url) – Define a URL na qual a página será redirecionada.

Curso de Java para Web

javax.servlet.http.HttpServletResponse

Enviando uma resposta HTML ao cliente:

```
package web;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TesteResponseServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html><head><title>Meu Primeiro Servlet</title></head>");
        out.println("<body>Teste do HttpServletResponse!</body>");
        out.flush(); //força o envio dos dados do buffer
        out.println("</html>");
    }
}
```

Curso de Java para Web

javax.servlet.http.HttpServletResponse

Enviando o status da resposta ao cliente:

```
package web;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TesteResponseServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res )
        throws IOException, ServletException {

        boolean acessoOK = false;
        //verifica o acesso do usuário
        if( acessoOK ) {
            res.setStatus( 200 ); //Status de OK
        } else {
            res.setStatus( 403, "Usuário não possui permissão de acesso" );
        }
    }
}
```

Curso de Java para Web

javax.servlet.http.HttpServletResponse

Redirecionando a página do cliente:

```
package web;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TesteResponseServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res )
        throws IOException, ServletException {

        boolean acessoOK = false;
        //verifica o acesso do usuário
        if( "noticia".equals( req.getParameter("area") ) ) {
            res.sendRedirect( "http://www.cnn.com" );
        } else {
            PrintWriter out = res.getWriter();
            out.println("Nenhuma área informada!");
        }
    }
}
```

Curso de Java para Web

javax.servlet.http.HttpServletResponse

Servlet que envia um arquivo jar:

```
package web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class JarServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        res.setContentType("application/jar");
        OutputStream out = res.getOutputStream();
        String jar = "/arquivo.jar";
        InputStream is = getServletContext().getResourceAsStream( jar );
        byte b = -1;
        while( (b = (byte)is.read()) > -1 ) {
            out.write( b );
        }
        out.flush(); //força o envio dos dados do buffer
    }
}
```

Curso de Java para Web

Despachando a solicitação

Às vezes podemos decidir remeter a solicitação para outro recurso, após o processamento do nosso servlet. Imagine um servlet que faça um processamento e gera uma resposta, mas em caso de erro desejamos repassar a solicitação para outro recurso. Para isso devemos utilizar o RequestDispatcher. O request e response é repassado ao próximo recurso.

Exemplo:

```
package web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ProcessoServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res )
        throws IOException, ServletException {
        try {
            //faz o processamento e gera uma saída
        } catch( Exception e ) {
            //caso o processamento lance uma exceção, remete pra página de erro
            RequestDispatcher rd =
                getServletContext().getRequestDispatcher("/erro.htm");
            rd.forward( req, res );
        }
    }
}
```

Curso de Java para Web

Incluindo um recurso externo

A mesma classe RequestDispatcher permite que, além de despacharmos para outro recurso, podemos incluir outro recurso no processamento atual.

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;

public class IncludeServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res)
        throws java.io.IOException, ServletException {
        res.getWriter().println("Recurso 1");
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/InclusoServlet");
        rd.include( req, res );
    }
}
```

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;

public class InclusoServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res)
        throws java.io.IOException, ServletException {
        res.getWriter().println("Recurso 2");
    }
}
```

Curso de Java para Web

Repassando novos valores no request

Quando fazemos o *forward()* ou *include()* de outro recurso, os dados do request e response são repassados ao recurso solicitado, porém também podemos tirar vantagem dos recursos do request para repassar novas informações aos próximos recursos. Para isso a classe `HttpServletRequest` oferece os métodos:

`void setAttribute(String name, Object value)` – Adiciona o objeto passado na sessão, usando o nome especificado.

`Object getAttribute(String name)` – Retorna o objeto armazenado na sessão referenciado pelo nome especificado.

Exemplo:

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;
public class RepassaInfoServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res )
        throws java.io.IOException, ServletException {
        Integer i = new Integer( req.getParameter("valor") );
        i = new Integer( i.intValue() * 123 ); //faz um calculo qualquer
        req.setAttribute( "valor", i );
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/InclusoServlet");
        rd.forward( req, res );
    }
}
```

Curso de Java para Web

Sessão de Usuário

Os servlets suportam sessão de usuário. A sessão de usuário serve como um repositório dos dados pertinentes à sessão do usuário específico, mantendo, desta forma, o estado da sessão do usuário. A interface que representa a sessão de usuário é a *javax.servlet.http.HttpSession*.

A classe possui os seguintes métodos:

void setAttribute(String name, Object value) – Adiciona o objeto passado na sessão, usando o nome especificado.

Object getAttribute(String name) – Retorna o objeto armazenado na sessão referenciado pelo nome especificado.

Para pegar o objeto HttpSession, devemos utilizar o método *getSession()* da classe *HttpServletRequest*.

A sessão de usuário é usada, por exemplo, num sistema de e-commerce para armazenar os itens do carrinho de compra do internauta. Ou então para armazenar os dados do usuário de um sistema, após ele fazer login no sistema.

Curso de Java para Web

Sessão de Usuário e Login

Vamos analisar o exemplo de um servlet que faz o login do usuário no sistema e mantém as informações do usuário na sessão.

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {
    public void doPost( HttpServletRequest req, HttpServletResponse res)
        throws java.io.IOException, ServletException {

        boolean validacaoOK = true;
        String usuario = req.getParameter("usuario");
        String senha = req.getParameter("senha");
        if( usuario==null || senha==null ) validacaoOK = false;
        //faz a validação do usuário
        if( validacaoOK ) {
            HttpSession sessao = req.getSession( true );
            sessao.setAttribute( "usuario", usuario );
            res.getWriter().println("Login realizado com sucesso!");
        } else {
            //usuario nao foi validado. Remete para página de erro de login
            RequestDispatcher rd = getServletContext().getRequestDispatcher("/errLogin.htm");
            rd.forward( req, res );
        }
    }
}
```

Curso de Java para Web

Sessão de Usuário e Login

Agora vamos usar um servlet que só pode ser acessado se o usuário fez o login.

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;

public class AcessoRestritoServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res)
        throws java.io.IOException, ServletException {
        HttpSession sessao = req.getSession( true );
        String usuario = (String) sessao.getAttribute("usuario");
        java.io.PrintWriter out = res.getWriter();
        if( usuario != null ) {
            out.println("Usuário possui acesso liberado no login.");
        } else {
            //usuario nao tem o nome na sessão. Não está logado
            out.println("<html><body><form method='post' action='LoginServlet'>");
            out.println("Usuario: <input type='text' name='usuario'> <br>");
            out.println("Senha: <input type='password' name='senha'> <br>");
            out.println("<input type='submit' value='OK'></form>");
            out.println("</body></html>");
        }
    }
}
```

Curso de Java para Web

Compartilhando informações

Como vimos nas classes `HttpServletRequest`, `HttpSession` e `ServletContext`, possuímos uma maneira em comum de se compartilhar informações e dados para cada um desses componentes.

Cada um possui seu escopo definido. O `HttpServletRequest` possui escopo de request, ou seja, só é válido enquanto o request for válido. Ao final da resposta ele perde a validade. Cada usuário possui um request por solicitação. Este é um bom lugar para armazenar dados que serão usados no próprio request.

O `HttpSession` possui escopo de sessão, ou seja, é válido enquanto a sessão do usuário estiver ativa. O servidor identifica o usuário e deixa uma sessão válida, até que ela seja invalidada programaticamente ou que o tempo da sessão expire. Cada usuário tem a sua própria sessão. Este é um bom lugar para guardar informações pertinente ao usuário e sua sessão.

O `ServletContext` possui escopo de aplicação, que é válido por toda a aplicação, do início do serviço do servidor ao final dele. Os dados armazenados aí, estarão acessíveis para qualquer servlet ou jsp da aplicação. Este é um bom lugar para guardar dados e recursos compartilhados.

Todas essas interfaces possuem os métodos:

`void setAttribute(String name, Object value)` – Adiciona o objeto passado no escopo, usando o nome especificado.

`Object getAttribute(String name)` – Retorna o objeto armazenado no escopo, referenciado pelo nome especificado.

Curso de Java para Web

Servlets Monoprocessados

Os Servlet Containers podem ter mais de uma instância de um mesmo servlet ativa, porém ele usa o paralelismo para responder a solicitação de muitos usuários, ou seja, o container pode utilizar o mesmo servlet para responder a solicitações em paralelo, pois os servlets são *multi-threaded*.

Há uma maneira de tornar os servlet mono processados, porém isto tem um impacto fortíssimo na aplicação, pois cada solicitação só poderá ser atendida depois que a solicitação anterior já foi atendida, ou seja, ao final da execução do servlet, o que pode ocasionar uma grande espera de resposta.

A interface que torna um servlet mono processado é a interface `SingleThreadModel`.

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;

public class MonoProcessadoServlet extends HttpServlet implements SingleThreadModel {
    public void service( HttpServletRequest req, HttpServletResponse res)
        throws java.io.IOException, ServletException {
        try { Thread.sleep( 10000 ); } //espera 10 segundos
        catch( Exception e ) {}
        res.getWriter().println("Servlet Processado!");
    }
}
```

Se chamarmos este servlet em duas janela separadas, veremos que a segunda chamada levará um tempo a ser executada, até que a primeira tenha terminado.

Curso de Java para Web

Eventos de Contexto

É possível sabermos quando algum evento ocorreu com o `ServletContext`, ou seja, com o contexto da aplicação. O web container nos informa sobre esses eventos por meios de Listeners. Esses eventos podem ser: criação ou destruição do contexto (ciclo de vida do contexto) e adição, alteração ou remoção de algum atributo no contexto.

Para isso, dispomos das interfaces `ServletContextListener`, `ServletContextEvent`, `ServletContextAttributeListener` e `ServletContextAttributeEvent`, todas do pacote `javax.servlet`.

As classes que implementem `ServletContextListener` servem para responder aos eventos do ciclo de vida do contexto. As classes devem implementar os seguintes métodos:

`public void contextInitialized(ServletContextEvent e)` – Este método é invocado quando o contexto é criado pelo container.

`public void contextDestroyed(ServletContextEvent e)` – Este método é invocado quando o contexto é destruído pelo container.

A interface `ServletContextEvent` tem o método `getServletContext()` que retorna o contexto da aplicação.

Curso de Java para Web

Eventos de Contexto

Exemplo:

```
public class MeuContextListener implements javax.servlet.ServletContextListener {
    public void contextInitialized( javax.servlet.ServletContextEvent e ) {
        System.out.println( "Meu contexto foi iniciado..." );
    }

    public void contextDestroyed( javax.servlet.ServletContextEvent e ) {
        System.out.println( "Meu contexto foi destruído..." );
    }
}
```

Agora precisamos declarar o listener no web.xml, para que o container o reconheça e o execute. No web.xml:

```
<web-app>
  <listener>
    <listener-class>MeuContextListener</listener-class>
  </listener>
</web-app>
```

Podem existir vários elementos <listener> dentro do web.xml, para registrar todos os listeners da aplicação. Os elementos <listener> devem vir antes do element <servlet>.

Curso de Java para Web

Eventos dos Atributos do Contexto

Podemos ainda sermos notificados quando qualquer atributo seja adicionado, alterado ou removido do contexto. Para isso utilizaremos a interface `ServletContextAttributeListener`.

As classes que implementem `ServletContextAttributeListener` devem implementar os seguintes métodos:

`public void attributeAdded(ServletContextAttributeEvent e)` – Este método é invocado quando um atributo é adicionado no contexto.

`public void attributeReplaced(ServletContextAttributeEvent e)` – Este método é invocado quando um atributo do contexto é substituído (ou alterado).

`public void attributeRemoved(ServletContextAttributeEvent e)` – Este método é invocado quando um atributo é removido do contexto.

A interface `ServletContextAttributeEvent` possui o método `getServletContext()` que retorna o contexto da aplicação, além dos métodos `getName()` e `getValue()` que retornam o nome e o valor do atributo, respectivamente.

Curso de Java para Web

Eventos dos Atributos do Contexto

Exemplo:

```
public class MeuContextAttributeListener
    implements javax.servlet.ServletContextAttributeListener {
    public void attributeAdded( javax.servlet.ServletContextAttributeEvent e ) {
        System.out.println( "Novo atributo: " + e.getName() + "=" + e.getValue() );
    }

    public void attributeReplaced( javax.servlet.ServletContextAttributeEvent e ) {
        System.out.println( "Atributo alterado: " + e.getName() + "=" + e.getValue() );
    }

    public void attributeRemoved( javax.servlet.ServletContextAttributeEvent e ) {
        System.out.println( "Atributo removido: " + e.getName() + "=" + e.getValue() );
    }
}
```

Para declarar o listener no web.xml, seguimos os mesmos passos explicados anteriormente.

```
<web-app>
  <listener>
    <listener-class>MeuContextAttributeListener</listener-class>
  </listener>
</web-app>
```

Curso de Java para Web

Eventos de Sessão

Assim como nos eventos de contexto, o container pode nos notificar sobre os eventos de sessão, tanto na sua criação, destruição e com seus atributos.

Para isso, dispomos das interfaces `HttpSessionListener`, `HttpSessionEvent`, `HttpSessionAttributeListener` e `HttpSessionBindingEvent`, todas do pacote `javax.servlet.http`.

As classes que implementem `HttpSessionListener` servem para responder aos eventos do ciclo de vida da sessão. As classes devem implementar os seguintes métodos:

`public void sessionCreated(HttpSessionEvent e)` – Este método é invocado quando a sessão é criada pelo container.

`public void sessionDestroyed(HttpSessionEvent e)` – Este método é invocado quando a sessão é destruída pelo container.

A interface `HttpSessionEvent` possui o método `getSession()` que retorna a sessão do usuário corrente.

Curso de Java para Web

Eventos de Sessão

Exemplo:

```
public class MeuSessionListener
    implements javax.servlet.http.HttpSessionListener {
    public void sessionCreated( javax.servlet.http.HttpSessionEvent e ) {
        System.out.println( "Sessão criada..." );
    }

    public void sessionDestroyed( javax.servlet.http.HttpSessionEvent e ) {
        System.out.println( "Sessão destruída..." );
    }
}
```

Agora precisamos declarar o listener no web.xml, para que o container o reconheça e o execute. No web.xml:

```
<web-app>
  <listener>
    <listener-class>MeuSessionListener</listener-class>
  </listener>
</web-app>
```

Curso de Java para Web

Eventos dos Atributos de Sessão

Podemos ainda sermos notificados quando qualquer atributo seja adicionado, alterado ou removido da sessão de usuário. Para isso utilizaremos a interface `HttpSessionAttributeListener`.

As classes que implementem `HttpSessionAttributeListener` devem implementar os seguintes métodos:

`public void attributeAdded(HttpSessionBindingEvent e)` – Este método é invocado quando um atributo é adicionado na sessão.

`public void attributeReplaced(HttpSessionBindingEvent e)` – Este método é invocado quando um atributo da sessão é substituído (ou alterado).

`public void attributeRemoved(HttpSessionBindingEvent e)` – Este método é invocado quando um atributo é removido da sessão.

A interface `HttpSessionBindingEvent` possui o método `getSession()` que retorna a sessão do usuário, além dos métodos `getName()` e `getValue()` que retornam o nome e o valor do atributo, respectivamente.

Curso de Java para Web

Eventos dos Atributos da Sessão

Exemplo:

```
public class MeuSessionAttributeListener
    implements javax.servlet.http.HttpSessionAttributeListener {
    public void attributeAdded( javax.servlet.http.HttpSessionBindingEvent e ) {
        System.out.println( "Novo atributo: " + e.getName() + "=" + e.getValue() );
    }

    public void attributeReplaced( javax.servlet.http.HttpSessionBindingEvent e ) {
        System.out.println( "Atributo alterado: " + e.getName() + "=" + e.getValue() );
    }

    public void attributeRemoved( javax.servlet.http.HttpSessionBindingEvent e ) {
        System.out.println( "Atributo removido: " + e.getName() + "=" + e.getValue() );
    }
}
```

Para declarar o listener no web.xml, seguimos os mesmos passos explicados anteriormente.

```
<web-app>
  <listener>
    <listener-class>MeuSessionAttributeListener</listener-class>
  </listener>
</web-app>
```

Curso de Java para Web

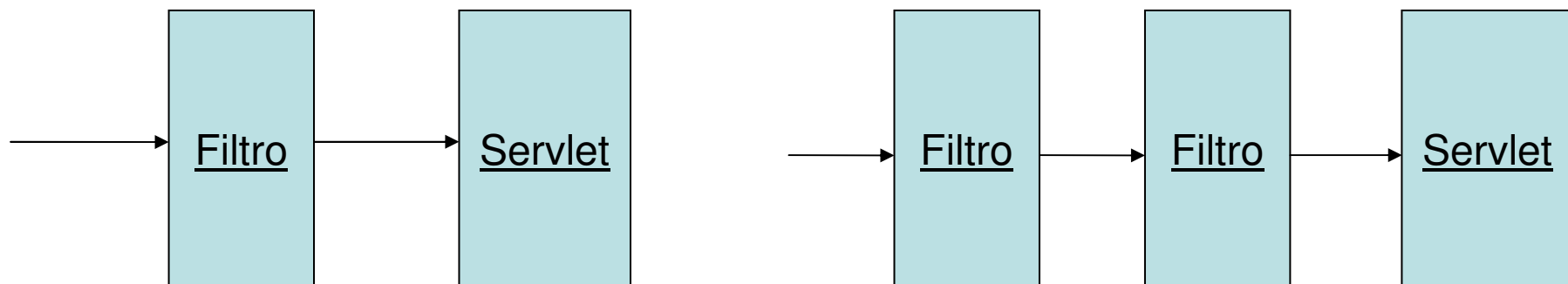
Filtros

Os filtros servem para interceptar as requisições feitas ao container, antes dela atingir o recurso solicitado.

É possível trabalhar com o request (`HttpServletRequest`) e o response (`HttpServletResponse`), alterando os seus estados.

Múltiplos filtros podem ser usados. Eles, neste caso, serão encadeados.

Os filtros devem implementar a interface ***java.servlet.Filter***. A interface ***FilterConfig***, contém as configurações do filtro, declaradas no ***web.xml***. A interface ***FilterChain*** é usada para repassar o comando para o próximo filtro, ou para o recurso solicitado, caso seja o último filtro da cadeia.



Curso de Java para Web

Filtros

As classes quem implementam a interface Filter, devem implementar os seguintes métodos:

public void init(FilterConfig filterConfig) - Este método é invocado pelo container quando o filtro for criado. O objeto FilterConfig passado como argumento contém as configurações do filtro.

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) - Este método é invocado pelo container quando uma solicitação é feita a um recurso, para o qual o filtro está mapeado.

public void destroy() - Este método é invocado pelo container quando o filtro for destruído.

A interface FilterConfig fornece os seguintes métodos:

```
public String getFilterName()  
public String getInitParameter( String parameterName )  
public java.util.Enumeration getInitParameterNames()  
public ServletContext getServletContext()
```

A interface FilterChain fornece o seguinte método:

```
public void doFilter(ServletRequest req, ServletResponse res )
```

A implementação das interfaces FilterConfig e FilterChain são oferecidas pelo próprio container.

Curso de Java para Web

Filtros

Exemplo de um filtro simples:

```
import javax.servlet.*;

public class MeuFiltro implements Filter {
    private FilterConfig filterConfig;

    public void init( FilterConfig filterConfig ) {
        this.filterConfig = filterConfig;
        System.out.println( "MeuFiltro iniciado..." );
    }

    public void doFilter( ServletRequest req, ServletResponse res,
                        FilterChain chain ) {
        System.out.println( "MeuFiltro invocado! - param1=" +
                            filterConfig.getInitParameter( "param1" ) );
        chain.doFilter( req, res );
    }

    public void destroy() {
        System.out.println( "MeuFiltro destruído" );
    }
}
```

Curso de Java para Web

Filtros

Agora devemos configurar o nosso filtro, no arquivo web.xml:

```
<web-app>
  <filter>
    <filter-name>MeuFiltro</filter-name>
    <filter-class>MeuFiltro</filter-class>
    <init-param>
      <param-name>param1</param-name>
      <param-value>valor1</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>MeuFiltro</filter-name>
    <url-pattern>/*</ulr-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>MeuFiltro</filter-name>
    <servlet-name>MeuServlet</servlet-name>
  </filter-mapping>

  <servlet>
    <servlet-name>MeuServlet</servlet-name>
    <servlet-class>web.MeuServlet</servlet-class>
  </servlet>
</web-app>
```

Curso de Java para Web

Cookies

A tecnologia Servlet também oferece um ótimo suporte a cookies.
A classe Cookie representa um cookie que é armazenado no browser do cliente.

Para obtermos os Cookies do cliente, utilizamos um método do request:

```
Cookie[] cookies = request.getCookies();
```

Para criarmos e adicionarmos novos Cookies no cliente, apenas criamos um objeto do tipo Cookie, armazenando o nome do Cookie e seu valor, e em seguida, adicionamos o Cookie no response.

```
Cookie cookie = new Cookie("ID_USUARIO", "12345");  
response.addCookie(cookie);
```

Curso de Java para Web

JavaServer Pages (JSP)

Os JSPs são páginas HTML que contêm código Java embutido, estendendo a funcionalidade dos HTMLs, tornando-os dinâmicos.

A sintaxe é a mesma do Java, a diferença é que vamos inserir código Java no corpo do HTML. Para isso, devemos codificar o código Java dentro do bloco marcado por `<% e %>`. Tudo dentro deste bloco é Java, tudo fora dele é texto ou HTML.

A intenção de se usar JSP é a de tirar o código HTML de dentro dos Servlets, que demanda muito trabalho para codificar e manter. Como JSPs podemos desenhar um HTML e então adicionar Java dentro dele.

Arquivo: teste.jsp

```
<html>
<head>
  <title>Teste JSP</title>
</head>
<body>
<%
  for( int i=1; i<5; i++ ) {
%>
  <font size="<%=i%>">Texto tamanho <%=i%></font>
<%
  }
%>
</body>
</html>
```

Curso de Java para Web

JavaServer Pages (JSP)

O código do JSP anterior seria o equivalente abaixo em Servlet. Qual é mais prático???

```
public class MeuServlet extends HttpServlet {
    public void service( HttpServletRequest req, HttpServletResponse res )
        throws ServletException, java.io.IOException {
        PrintWriter pw = res.getWriter( );
        pw.println("<html>");
        pw.println("<head>");
        pw.println(" <title>Teste JSP</title>");
        pw.println("</head>");
        pw.println("<body>");
        for( int i=1; i<5; i++ ) {
            out.println("<font size=" + i + ">Texto tamanho " + i + "</font>");
        }
        pw.println("</body>");
        pw.println("</html>");
    }
}
```

Curso de Java para Web

JavaServer Pages (JSP)

Scriptlets

Todos os comandos Java dentro do bloco marcado por `<%` e `%>`, chamamos de scriptlet.

Se desejarmos imprimir no JSP o valor de uma variável, podemos usar o scriptlet da seguinte forma:

```
<%  
    String nome = "João da Silva";  
    out.println( nome );  
%>
```

`out` é um objeto implícito do JSP, que representa a saída (PrintWriter) do JSP.

Expressões

Atua como uma facilidade de imprimir uma resposta. O código abaixo é o equivalente à segunda linha do exemplo acima.

```
<%=nome%>
```

Repare o caracter = antes da expressão, e que ela não termina com o caracter ponto-e-vírgula.

```
<html>  
<body>  
<%  
    int valor = 1000 * 999;  
    out.println( valor );  
%>  
<br>  
<%=valor%></body>  
</html>
```

Curso de Java para Web

JavaServer Pages (JSP)

Quando o servlet container for executar o JSP pela primeira vez, ele o converte para um classe Servlet primeiro, por isso pode levar algum tempo até que o jsp seja processado pela primeira vez. Portanto, podemos dizer que o JSP vai virar um Servlet mais tarde. Porém isto não muda nada na nossa maneira de programar, pois podemos solicitar o JSP pelo nome, normalmente. Nem a configuração no *web.xml* é necessária.

Tudo o que for codificado dentro do seu JSP estará dentro do método *service()* do Servlet gerado. Porém há uma maneira de se declarar novos métodos e atributos para o JSP. Usamos a **Declaração**.

```
<%!  
    void novoMetodo() {  
        //faz algo  
    }  
  
    private int valor = 10;  
%>  
<html>  
<head><title>Teste JSP</title></head>  
<body>  
<%  
    novoMetodo(); //chamado ao método declarado no JSP  
    out.println( valor );  
%>  
</body>  
</html>
```

Curso de Java para Web

JavaServer Pages (JSP)

Diretivas

As diretivas informam as informações gerais sobre a página JSP para a Engine JSP. Existem três tipos de diretivas: *page*, *include* e *taglib*.

Exemplo:

```
<%@ page language="java" import="java.util.Date" %>
<html>
<head><title>Teste JSP</title></head>
<body>
<%
    Date agora = new Date();
%>
Data atual: <%=agora%>
<%@ include file="outra-pagina.htm" %>
</body>
</html>
```

A diretiva *page* informa uma lista de atributos sobre a página, como os imports a serem feitos, tamanho de buffer da página, auto flush, se é thread safe, content type, se é página de erro, qual a página de erro e etc. A diretiva *include* inclui outro recurso (dinâmico ou não) no nosso JSP. A diretiva *taglib* trabalha com Tag Libs, que veremos mais tarde.

Curso de Java para Web

JavaServer Pages (JSP)

Utilizando páginas de erro em jsp.

Arquivo: teste.jsp

```
<%@ page language="java" import="java.util.Date" %>
<%@ page errorPage="erro.jsp" %>
<html>
<head><title>Teste da Página de Erro do JSP</title></head>
<body>
<%
    if( true ) throw new Exception("Este é um erro bobo!");
%>
</body>
</html>
```

Arquivo: erro.jsp

```
<%@ page isErrorPage="true" %>
<html>
<head><title>Página de erro do JSP</title></head>
<body>
Erro: <font color='red'><%=exception.getMessage() %></font>
</body>
</html>
```

Curso de Java para Web

JavaServer Pages (JSP)

Objetos implícitos

Os JSPs possuem alguns objetos implícitos na página. São eles:

- **application** - Objeto ServletContext
- **request** - Objeto HttpServletRequest
- **response** - Objeto HttpServletResponse
- **session** - Objeto HttpSession
- **pageContext** - Objeto PageContext, que representa a própria página
- **page** - Referência à própria página JSP (this)
- **out** - Objeto PrintWriter que envia resposta ao cliente
- **config** - Objeto ServletConfig
- **exception** - Objeto Throwable, disponível somente nos JSPs que são páginas de erro

```
<html>
<head><title>Objetos Implícitos do JSP</title></head>
<body>
<%
  String val = request.getParameter("param1");
  out.println("Texto enviado pelo out");

  session.setAttribute( "attrib1", "Valor do Attrib1" );
%>
Valor do param1: <%=val%>
Attrib1 da sessão: <%=session.getAttribute("attrib1")%>
</body>
</html>
```

Curso de Java para Web

JavaServer Pages (JSP)

Actions

As actions são comandos já pré-programados. Elas são declaradas no formato de tags.

As actions são:

- ✓ include
- ✓ forward
- ✓ useBean
- ✓ setProperty
- ✓ getProperty
- ✓ plugin

A sintaxe de utilização é a seguinte: `<jsp:nomeDaAction atributos />`

Exemplo:

```
<jsp:include page="pagina.htm" />
```

```
<jsp:forward page="MeuServlet" />
```

```
<jsp:forward page="pagina.jsp">  
  <jsp:param name="nome" value="Maria" />  
  <jsp:param name="sobrenome" value="Madalena" />  
</jsp:forward>
```

Curso de Java para Web

JavaServer Pages (JSP)

A action ***include*** é usada para incluir um recurso no JSP. A action ***forward*** é usada para repassar o processamento para outro recurso. Elas funcionam da mesma maneira que os métodos `include` e `forward` da classe `RequestDispatcher`. O atributo *page* recebe o nome do recurso (Servlet, JSP, etc), que pode ser fornecido dinamicamente, em tempo de execução. Exemplo:

```
<jsp:include page="pagina.htm" />
```

```
<jsp:forward page="recurso.jsp" />
```

Curso de Java para Web

JavaServer Pages (JSP)

A action **useBean** é usada para se criar uma referência a um bean existente ou a um novo bean criado.

```
<jsp:useBean id="nomeBean" class="String" scope="session" />
```

No código acima ele tenta buscar o objeto String, guardado na sessão com o nome de atributo nomeBean. Se existir ele referencia este objeto achado, senão uma nova String é criada. Então o objeto é guardado no escopo definido. Isto serve para qualquer outro tipo de classe também. Exemplo:

```
<jsp:useBean id="carro" class="Carro" scope="session" />
<%
    if( carro.getModelo().equals( "Golf" ) { }
%>
```

O código acima é similar a:

```
Carro carro = null;
if( session.getAttribute("carro") == null ) carro = new Carro();
else carro = (Carro) session.getAttribute("carro");
Session.setAttribute( "carro" carro );
if( carro.getModelo().equals( "Golf" ) {}
```

Curso de Java para Web

JavaServer Pages (JSP)

A action *setProperty* serve para alterar o valor das propriedades de um bean ou objeto existente.

Exemplo:

```
<jsp:useBean id="carro" class="Carro" scope="request" />
<jsp:setProperty name="carro" property="modelo" value="Omega" />
```

No exemplo acima, o valor "Omega" 'atribuído ao atributo modelo do objeto Carro. Para que isso ocorra, o método *setModelo(String modelo)* deve existir. O código é semelhante a:

```
carro.setModelo( "Omega" );
```

Podemos, ao invés de usarmos uma valor definido em código, atribuir o valor vindo do request:

```
<jsp:setProperty name="carro" property="modelo" param="modelo" />
```

O código é equivalente a:

```
carro.setModelo( request.getParameter("modelo") );
```

Curso de Java para Web

JavaServer Pages (JSP)

A action *getProperty* serve para recuperar e imprimir o valor das propriedades de um bean ou objeto existente. Exemplo:

```
<jsp:useBean id="carro" class="Carro" scope="request" />  
<jsp:getProperty name="carro" property="modelo" />
```

No exemplo acima, o valor do atributo modelo do objeto carro é imprimido no output do JSP. Para que isso ocorra, o método *getModelo()* deve existir. O código é semelhante a:

```
out.println( carro.getModelo() );
```

Curso de Java para Web

TagLibs

As TagLibs, ou Tag Libraries, são um conjunto de tags personalizadas que adicionam funcionalidade ao seu JSP.

Essas tags podem ser feitas por terceiros (ex: tags do Struts) ou criadas por você mesmo, segundo sua necessidade.

As TagLibs visam substituir os scriptlets do seu JSP, a fim de deixar seu JSP mais limpo e simples, além de facilitar a vida dos web designers.

As actions, vistas anteriormente, são, nada mais, nada menos, que uma TagLib padrão dos JSPs.

Essas TagLibs são um conjunto de classes que definem as ações e comportamento de cada tag, além de um arquivo TLD (Tag Lib Descriptor) que descreve a funcionalidade da TagLib.

As classes e interfaces da TagLibs fazem parte do pacote *javax.servlet.jsp.tagext*.

A interface Tag é a base de todas as tags. Também contamos com as interfaces IterationTag, BodyTag e BodyContent.

Curso de Java para Web

TagLibs

Quando formos utilizar TagLibs fornecidas por terceiros, geralmente teremos um arquivo JAR com as classes que implementam as tags e um ou mais arquivos TLD, que descrevem essas tags.

Configurando e acessando as TagLibs:

- Copie o JAR fornecido para dentro do diretório /WEB-INF/lib/ da sua aplicação web;
- Configure (declare) a TagLib no seu web.xml;
- Configure a TagLib no seu JSP;
- Faça uso da TagLib no seu JSP.

Curso de Java para Web

TagLibs

Depois de copiar o JAR com as classes de tags fornecidas, devemos configurar o web.xml.

Vamos supor que o arquivo TLD fornecido tenha o nome “mytaglib.tld”. Copie-o para dentro do diretório WEB-INF da sua aplicação.

Então, no web.xml, devemos adicionar a seguinte configuração, dentro da tag <web-app>:

```
<taglib>
  <taglib-uri>/myTagLib</taglib-uri>
  <taglib-location>/WEB-INF/mytaglib.tld</taglib-location>
</taglib>
```

Curso de Java para Web

TagLibs

Agora que temos tudo configurado no ambiente, devemos preparar o nosso JSP para usar e executar as tags.

```
<%@ page language="java" %>
<%@ taglib uri="/myTagLib" prefix="teste" %>
<html>
<body>
  <teste:minhaTag />
</body>
</html>
```

Neste simples exemplo, consideramos que exista uma tag chamada MinhaTag, que faz parte da TagLib utilizada.

* Para maiores detalhes de como executar a tag desejada, devemos consultar a documentação das tags fornecidas.

Curso de Java para Web

Criando TagLibs

Assim como podemos utilizar tags de terceiros, podemos também, criar nossas próprias tags.

Para isso, primeiro, devemos criar uma classe que será a nossa tag, e que fará parte da nossa TagLib (ou biblioteca de tags).

Exemplo:

```
package teste.web.tag;
import javax.servlet.*;
import javax.servlet.jsp.tagext.*;

public class MinhaTag extends TagSupport {
    public int doEndTag() throws JspException {
        JspWriter out = pageContext.getOut();
        try {
            out.println("MinhaTag diz - Olá Mundo!");
        } catch (Exception e) {}
        return EVAL_PAGE;
    }
}
```

Este exemplo de tag, imprime uma simples mensagem (texto) no código gerado do JSP.

Curso de Java para Web

Criando TagLibs

Depois de criada e compilada, precisamos criar o TLD que descreve nossa TagLib. Para isso, criaremos um arquivo chamado minhataglib.tld, com a seguinte estrutura:

```
<?xml version="1.0" encoding="ISSO-8859-1" ?>
<!DOCTYPE taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <tag>
    <name>minhaTag</name>
    <tagclass>teste.web.tag.MinhaTag</tagclass>
  </tag>
</taglib>
```

Após criarmos o TLD e o copiarmos para a pasta WEB-INF, devemos, opcionalmente, configurar o web.xml, como no exemplo anterior, e então usar a tag, como no JSP abaixo:

```
<%@ taglib uri="/WEB-INF/minhataglib.tld" prefix="my" %>
<my:minhaTag />
```

Curso de Java para Web

Criando TagLibs

Algumas classes de suporte já são fornecidas, portanto, não precisamos escrever nossas tags a partir das interfaces. Podemos usar as seguintes classes:

TagSupport – implementa a interface IterationTag;

BodyTagSupport – implementa a interface BodyTag.

Ambas as classes possuem os seguintes métodos, sempre executados em sequência:

public void setPageContext(PageContext p) – Guarda uma referência para o objeto PageContext.

public void setParent(Tag parent) – Guarda referência para a tag pai, se existir.

public int doStartTag() – Executado no início do processamento da tag. Este método pode retornar os valores das seguintes constantes:

SKIP_BODY – ignora o processamento do corpo da tag

EVAL_BODY_INCLUDE – processa o corpo da tag

EVAL_BODY_BUFFERED – processa o corpo da tag com um objeto BodyContent

(apenas para BodyTagSupport)

Curso de Java para Web

Criando TagLibs

public int doAfterBody() – Executado uma ou mais vezes. Este método pode retornar os valores das seguintes constantes:

SKIP_BODY – ignora o processamento repetido do corpo da tag

EVAL_BODY_AGAIN – processa o corpo da tag novamente

public int doEndTag() – Chamado depois do método doStartTag(). Este método pode retornar os valores das seguintes constantes:

SKIP_PAGE – ignora o processamento do resto da página

EVAL_PAGE – processa o restante da página normalmente

public void release() – Último método a ser executado. Para limpeza de recursos alocados, se houver.

Curso de Java para Web

Criando TagLibs

A classe `BodyTagSupport` ainda tem os métodos:

`public void setBodyContent(BodyContent bodyContent)` – Guarda referência para o conteúdo do corpo da mensagem.

`public void doInitBody()` – Executado antes de processar o corpo da tag. Este método pode retornar os valores das seguintes constantes:

`EVAL_BODY_BUFFERED` – processa o corpo da tag com um objeto `BodyContent`

`EVAL_BODY_TAG` – processa o corpo da tag (depreciado)

Curso de Java para Web

Criando TagLibs

As suas tags ainda podem suportar parâmetros, que são informados na tag, na hora de usá-la no JSP.

Para cada parâmetro desejado, é necessário tem um atributo e seus respectivos métodos get e set, além de configurá-los no TLD.

Para exemplificar um uso completo de tag personalizada, vamos criar uma tag que faz a repetição do texto do corpo de sua tag.

Com este exemplo, seremos capazes de mostrar todas as funcionalidades das tags.

Curso de Java para Web

Criando TagLibs

```
package teste.web.tag;
import javax.servlet.*;
import javax.servlet.jsp.tagext.*;

public class RepeteTag extends BodyContentSupport {
    private int repeticoes = 0;
    private int repeticoesFeitas = 0; //controle interno

    public int getRepeticoes() { return repeticoes; }

    public void setRepeticoes(int repeticoes) {
        this.repeticoes = repeticoes;
    }

    public int doAfterBody() {
        repeticoesFeitas++;
        if( repeticoesFeitas > repeticoes ) {
            return SKIP_BODY;
        }
        JspWriter out = bodyContent.getEnclosingWriter();
        out.println( bodyContent.getString() );
        return EVAL_BODY_AGAIN;
    }
}
```

Curso de Java para Web

Criando TagLibs

Criando o TLD declarativo da nossa TagLib:

```
<?xml version="1.0" encoding="ISSO-8859-1" ?>
<!DOCTYPE taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <tag>
    <name>repeteco</name>
    <tagclass>teste.web.tag.RepeteTag</tagclass>
    <attribute>
      <name>repeticoes</name>
      <required>>true</required>
    </attribute>
  </tag>
</taglib>
```

No JSP:

```
<%@ taglib uri="/WEB-INF/minhataglib.tld" prefix="my" %>
<my:repeteco repeticoes="5">
Isto vai se repetir<br>
</my:repeteco>
```

Curso de Java para Web

Material de apoio

Java para a Web com Servlets, JSP e EJB

Kurniawan, Budi

Editora Ciência Moderna

Tutorial de HTML online

<http://www.icmc.usp.br/ensino/material/html/>

<http://www.w3schools.com/html/>