

Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia da Electrónica das
Telecomunicações e dos Computadores

Infraestruturas Computacionais Distribuídas

WEB - Geração de Páginas dinamicamente - Servlet's

Versão 1.2



Índice

- Ambiente executivo das Servlet's
- Construção de Servlet's
- Exemplo Simples
- Ciclo de Vida das Servlet
- Estado entre Pedidos
- Exemplos

ISEL/DEETC - Secção de Engenharia de Sistemas

2



Enquadramento

Geração de conteúdos dinâmicos no servidor:

- Common Gateway Interface – CGI
- Extensões ao servidor – ISAPI e NSAPI
- Java Servlet's**
- Server Side Includes (SSI)
- Server Side JavaScript (SSJS)
- Active Server Pages (ASP)
- JavaServer Pages (JSP)

ISEL/DEETC - Secção de Engenharia de Sistemas

3



Introdução

- Uma Servlet permite estender a funcionalidade de um servidor web do mesmo modo que um CGI.
- Não é mais que um objecto Java cujos métodos são invocados pelo servidor para tratar pedidos HTTP.
- Applet's Vs Servlet's**
 - Applet – Classe java que é executada do lado do cliente
 - Servlet – Classe java que é executada do lado do servidor

ISEL/DEETC - Secção de Engenharia de Sistemas

4



Introdução (cont.)

- As Servlet's são a forma de em Java criar aplicações Web enabled;
- Cada Servlet é como se fosse um mini servidor Web aumentando as suas capacidades fornecendo funcionalidades adicionais;
- Estas novas funcionalidades podem servir para criar:
 - Websites de E-commerce
 - Front-end para Bases de dados
 - Conversor de imagens
 - Etc...
- Uma Servlet recebe um pedido HTTP e retorna uma resposta HTTP

À semelhança de um
CGI

ISEL/DEETC - Secção de Engenharia de Sistemas

5



Porquê Servlets? (Vantagens)

- As Servlets são:**
 - Persistentes** – Ao contrário dos CGI's o ciclo de vida das servlets estende-se para além de um pedido HTTP.
 - Simplex** – São simples de escrever.
 - Quem realiza as tarefas mais complicadas é o servlet *container*.
 - Java é uma linguagem de alto nível que impede a manipulação directa da memória do sistema, reduzindo a probabilidade de bugs graves.
 - Flexíveis** – Tem acesso a todas as API's Java. Estas são bem definidas e com bastante experiência e suporte pela comunidade Java.
 - Eficientes** – Por cada pedido HTTP são utilizadas tarefas Java e não novos processos.
 - Portáveis** – Executam-se na **Máquina Virtual Java (JVM)**.
 - Seguras** – Executam-se no lado do servidor daí serem seguras para os clientes.

ISEL/DEETC - Secção de Engenharia de Sistemas

6



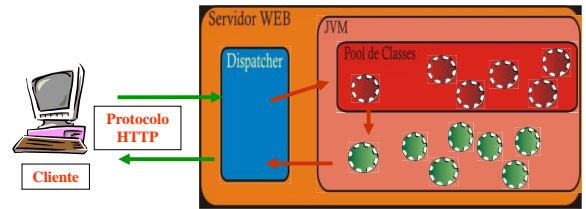
Introdução / Enquadramento

- As **Servlet's** e as **JavaServer Pages (JSP)** são um subconjunto da colecção de API's do lado do servidor, de nome **Java 2 Enterprise Edition (J2EE)**;
- As tecnologias **J2EE** foram desenvolvidas para providenciar uma infra-estrutura escalável e fácil de manter;
- As **Servlet's** e as **JavaServer Pages** juntas formam a camada de apresentação das **web applications J2EE**, enquanto que os **Enterprise JavaBeans (EJB)** são utilizados para desenvolver as camadas de lógica e de dados;
- Para aplicações de complexidade simples/moderada as **Servlet's** e as **JSP's** podem só por si, ou com a ajuda de **JavaBeans** normais, servir todas as necessidades.

Abordagem que vamos estudar

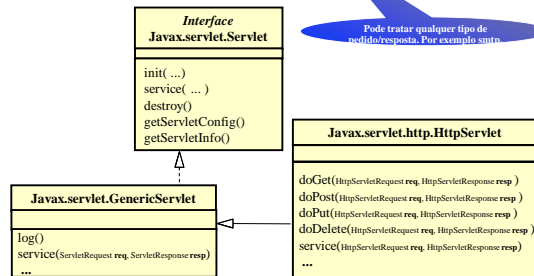


Ambiente executivo das Servlet's

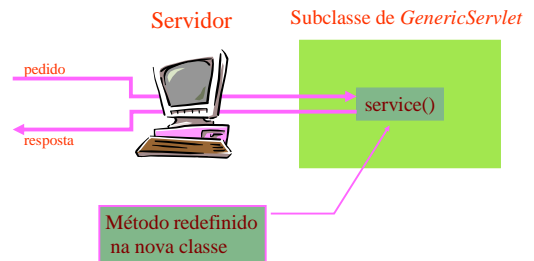


Construção de Servlet's

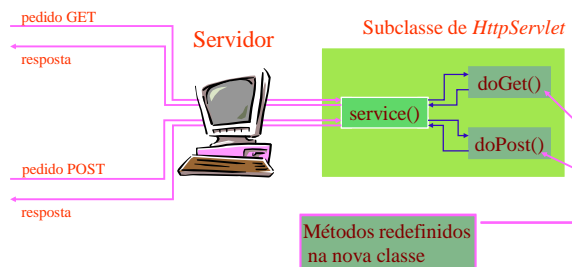
- Uma Servlet é apenas uma classe Java que implementa a interface **javax.servlet.Servlet**.
- Particularizando para o protocolo HTTP temos que uma Servlet será uma subclasse da classe **javax.servlet.http.HttpServlet**.
- Uma Servlet não tem método main(), apenas disponibiliza alguns métodos que são invocados pelo servidor Web.



Construção de Servlet's



Construção de Servlet's

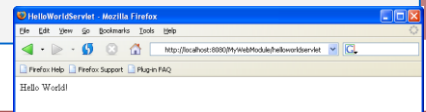


Exemplo – Hello World!!!

```
public class HelloWorldServlet extends javax.servlet.http.HttpServlet {

    //Process the HTTP Get request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloWorldServlet</title></head>");
        out.println("<body>");
        out.println("<p>Hello World! </p>");
        out.println("</body></html>");
    }
}
```





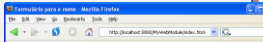
Exemplo 2 – Hello To!!!

```

<html>
<head>
<title>Formulário para o nome</title>
</head>
<body>
<div>
<font color="#000080"><h1>Formulário com o método http GET</h1></font>
<form method="GET" action="helloServlet">
<h3>Insira o seu nome:</h3><input type="text" name="nome"/>
<input type="submit" name="submit" value="Submeter"/>
<input type="reset" name="reset" value="Limpar"/>
</form>
</div>
<br>
<font color="#000080"><h1>Formulário com o método http POST</h1>
<form method="POST" action="helloServlet">
<h3>Insira o seu nome:</h3><input type="text" name="nome"/>
<input type="submit" name="submit" value="Submeter"/>
<input type="reset" name="reset" value="Limpar"/>
</form>
</body>
</html>

```

Página `html` (index.html) com o formulário que recolhe os dados para serem processados pela `servlet`



The screenshot shows a web browser window with the title "Formulário para o nome - Recife's Campus". The address bar shows "http://localhost:8080/FormularioServlet/index.html". The page content includes the heading "Formulário com o método http GET", the text "Insira o seu nome:", and a form with a text input field, a "Submeter" button, and a "Limpar" button.



Exemplo 2 – Hello To!!!

```

public class HelloToServlet extends HttpServlet {
//Process the HTTP Get request
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String rcvNome = request.getParameter("nome");
    if (rcvNome == null) {
        rcvNome = "John Doe"; // default value
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>HelloToServlet</title></head>");
    out.println("<body>");
    out.println("<div>Hello + " + rcvNome + "</div>");
    out.println("<p><i><b><div>\"new java.util.Date()\"</div> A Servlet recebeu um pedido http: " +
        request.getMethod() + ".</p>");
    out.println("</body></html>");
}

//Process the HTTP Post request
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
    doGet(request, response);
}
}

```



Exemplo 2 – Hello To!!!

The top screenshot shows a web browser window with the address bar displaying `http://localhost:3000/MyWebModule/PublishServlet?name=Diogo+%26+Caj%C3%A1`. The page content displays **Hello Diogo & Cajó** and a timestamp: *Mon Nov 21 17:06:05 GMT 2004*. A speech bubble points to the page, stating: "Foi submetido o primeiro formulário. Enviou-se para o servidor um pedido http **GET** com os dados concatenados na URL".

The bottom screenshot shows a code editor window displaying the HTML response received by the browser. The code is as follows:

```
<html>
<head><title>HelloServlet</title></head>
<body>
  <p>Hello Diogo & Cajó!</p>
  <p>Mon Nov 21 17:06:05 GMT 2004</p></body></html>
```

A speech bubble points to the code, stating: "Código fonte do HTML gerado".



Ciclo de Vida das Servlet's

- É instanciada num contentor próprio:
 - É carregada para memória pela primeira vez para servir o 1º pedido que a referência; permanece instanciada para os pedidos que, eventualmente, se sigam; não ser remida se não for usada a partir de um dado tempo sem ter pedidos. Os pedidos são executados através de tarefas (existe concorrência).
 - Partilham entre si a mesma máquina virtual. No entanto a linguagem Java impede que uma Servlet aceda aos dados doutra;
- Os servidores Web são livres de implementar o motor de suporte às Servlets desde que:
- Criem e iniciem as Servlets;
 - As Servlets tratem zero ou mais pedidos feitos por clientes;
 - Destruam a Servlet e efectuem o *garbage collection*.
- A máquina virtual Java pode residir:
- No próprio servidor Web (se o servidor for implementado em Java) (e.g. Tomcat *standalone*);
 - Numa máquina virtual Java a correr num processo, independente, do servidor Web (e.g. Apache e Tomcat interligados com um conector).

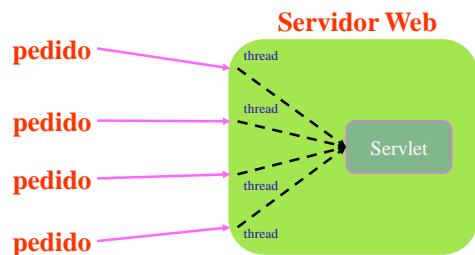


Ciclo de Vida das Servlet's

- As Servlets não têm “construtor”! Ou melhor têm mas não se utiliza para iniciar um objecto em particular.
- Em alternativa a API oferece dois métodos:
 - `init()` – chamado quando é criada uma nova instância da Servlet (instanciada no contentor);
 - `destroy()` – quando a Servlet é destruída (removida do contentor);
- Estes métodos são invocados pelo servlet *container* (contentor das servlets).
- Para cada pedido http recebido é atribuída uma nova tarefa e esta invoca o método:
`service (HttpServletRequest, HttpServletResponse) ;`
 - `HttpServletRequest`
 - Objecto que representa e guarda toda a informação referente ao pedido HTTP efectuado pelo browser.
 - `HttpServletResponse`
 - Objecto que representa e guarda toda a informação referente à resposta HTTP a ser devolvida para o *browser*.



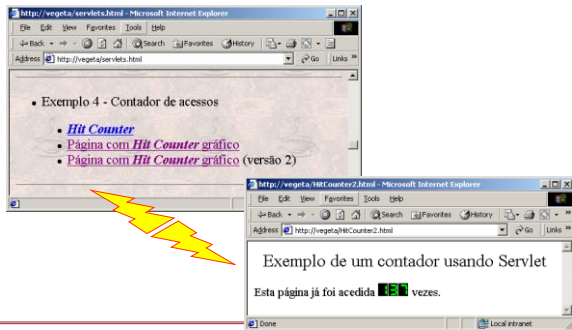
Ciclo de Vida das Servlet's



- Uma Servlet é instanciada no 1º acesso, ou no arranque do servidor, e essa mesma instância será utilizada por todas as threads lançadas para tratar os vários pedidos que forem recebidos pelo servidor web



Ciclo de Vida das Servlet's



INEL-DEITC - Secção de Engenharia de Sistemas

19



Ciclo de Vida das Servlet's

```
import java.io.*;
import java.awt.*;
import java.lang.*;
import javax.servlet.*;
import javax.servlet.http.*;
import Acme.JPM.Encoders.*;

public class HitCounterGh2 extends HttpServlet {
    static String DIR_IMAGE = "/images/odometer/";
    int count = 0;

    synchronized private String getCount() {
        return "" + (++count);
    }
}
```

INEL-DEITC - Secção de Engenharia de Sistemas

20



Ciclo de Vida das Servlet's

Que acontece se o servidor é desligado ?

```
private Image getImageCount(String stringCount) throws ServletException {
    //...
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("image/gif");

    GifEncoder encoder = new GifEncoder( getCount() ),
        response.getOutputStream() );
    encoder.encode();
}
};
```

Vou utilizar os métodos **init** e **destroy** para ler e guardar a contagem

INEL-DEITC - Secção de Engenharia de Sistemas

21



Ciclo de Vida das Servlet's

```
public void init(ServletConfig config) throws ServletException {
    super.init(config);

    try {
        FileReader fileReader = new FileReader("HitCounterGh2.dat");
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        count = Integer.parseInt( bufferedReader.readLine() );
        fileReader.close();
    }
    catch (Exception e) {
        count = 0;
    }
}
```

INEL-DEITC - Secção de Engenharia de Sistemas

22



Ciclo de Vida das Servlet's

```
public void destroy() {
    saveCount();
}

private void saveCount() {
    try {
        FileWriter fileWriter = new FileWriter("HitCounterGh2.dat");
        String stringCount = Integer.toString(count);
        fileWriter.write(stringCount, 0, stringCount.length());
        fileWriter.close();
    }
    catch (Exception e){}
}
```

INEL-DEITC - Secção de Engenharia de Sistemas

23



Web Application

O conceito de **web Application** foi introduzido na especificação das **Servlet's 2.2**.

De acordo com esta especificação, uma **web Application** é uma colecção de **servlet's**, **páginas html**, **classes** e qualquer outro **recurso** que possa ser empacotado para correr em vários **web containers** de diversos fabricantes.

Uma definição mais prática é dizer que uma **web Application** é tudo o que reside na camada **WEB** de uma aplicação.

Uma **aplicação web** pode ser constituída pelos seguintes itens:

- Servlet's
- JavaServer Pages (JSP)
- Classes utilitárias
- Documentos estáticos que incluem html, imagens, etc...
- Bibliotecas (ficheiros .jar)
- Meta informação que descreve a aplicação **web**

INEL-DEITC - Secção de Engenharia de Sistemas

24



Web Application (cont.)

- Na prática uma **Web Application** corresponde a uma directoria com o seu nome e com a respectiva estrutura de directorias que armazenam os seus componentes

Directoria:	Contém:
/WebAppName/	É a directoria raiz da aplicação web (<i>Web application</i>). Todas as páginas HTML e JSP são armazenadas aqui.
/WebAppName/WEB-INF/	É aqui que está localizado o descritor de instalação da aplicação web (<i>web.xml</i>). Note que esta directoria não pertence à parte pública da aplicação. Nenhum ficheiro nesta directoria pode ser accedido directamente por parte do cliente.
/WebAppName/WEB-INF/ classes/	Nesta directoria são colocadas as <i>servlet's</i> (.class) e todas as classes utilitárias.
/WebAppName/WEB-INF/ lib/	Esta directoria deve conter todos os Java Archive Files (.JAR) que a aplicação Web necessita. Por exemplo, onde se deve colocar o .jar que contém o driver JDBC para aceder a bases de dados.

ISEL/DEIETC - Secção de Engenharia de Sistemas

25



O deployment descriptor da web application

- O *deployment descriptor* é um ficheiro em **XML** (*Extensible Markup Language*) com o seguinte nome:

/WebAppName/WEB-INF/web.xml

- A informação que este contém inclui os seguintes elementos:

- Os parâmetros iniciais (*init*) do *ServletContext*
- Configuração das *webresources* (*resources*)
- Definição das *SERVLET's* e *JSP's*
- Mapeamento das *SERVLET's* e *JSP's*
- Mapeamento de *Mime types* (mapeamento [extensão do fich. <-> mime type])
- Lista das páginas iniciais (ex: *index.html*)
- Páginas de erros
- Conteúdo localizado

```
<web-app>
  <display-name>WebAppName</display-name>
  <description>Aplicacao exemplo</description>
  <session-timeout>30</session-timeout>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>HelloWorldServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  ...
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

/WebAppName/WEB-INF/web.xml

Classe que implementa a *servlet* *HelloWorldServlet.class*

Padrão de letras que identificam no URL a *servlet* *hello*



Web ARchive - WAR

- Empacotamento de uma **Web Application** :
 - Agora que já sabemos o que é uma **web application** podemos empacota-la para efectuar o seu **deployment** (instalação) num **web container**.
 - O método standard de empacotar uma **web application** é utilizando um **Web ARchive file** (WAR).
 - Podemos criar um ficheiro WAR utilizando a ferramenta "**Java archiving tool**" jar.

```
$jar cvf myTestWebApp.war .
added manifest
adding: WEB-INF/ (in = 0) (out= 0) (stored 0%)
adding: WEB-INF/classes/ (in = 0) (out= 0) (stored 0%)
adding: WEB-INF/classes/helloworldservlet/ (in = 0) (out= 0) (stored 0%)
adding: WEB-INF/classes/helloworldservlet/HelloWorldServlet.class (in = 1290) (out= 635) (deflated 50%)
adding: WEB-INF/web.xml (in = 530) (out= 266) (deflated 49%)
```

Na directoria raiz da Web Application

Como resultado é criado um ficheiro *myTestWebApp.war*, que contém toda a *web application* comprimida.

ISEL/DEIETC - Secção de Engenharia de Sistemas

27



Apache Jakarta Tomcat

- O Tomcat é o *servlet container* utilizado na implementação oficial de referência das tecnologias Java *Servlet* and *JavaServer Pages*

Servlet/JSP Spec	Tomcat version
2.5/2.1	6.0.16
2.4/2.0	5.5.26
2.3/1.2	4.1.37
2.2/1.1	3.3.2

Servlets: <http://java.sun.com/products/servlet/>

Especificação: <http://java.sun.com/products/servlet/reference/api/index.html>

Tomcat: <http://tomcat.apache.org/>

ISEL/DEIETC - Secção de Engenharia de Sistemas

28



Efectuar o Deployment de uma Web App.

- Existem várias formas de efectuar o **deploy** de uma **web Application** e varia consoante o *Servlet container* que está a ser utilizado:

- Copiar a directoria da *web application* para a directoria (necessita de um restart ao tomcat):

%Tomcat_Home%/webapps/

- ★ Utilizar o Web ARchive (war)

- ★ Via interface *web* do *manager* do tomcat

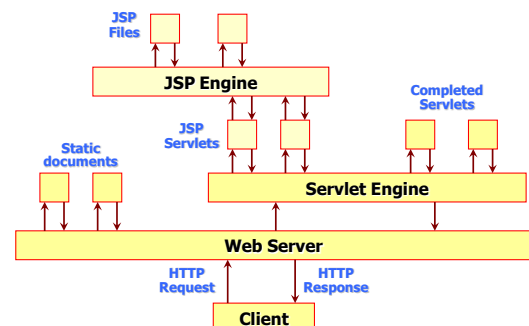


ISEL/DEIETC - Secção de Engenharia de Sistemas

29



Arquitectura interna de um Servlet Container

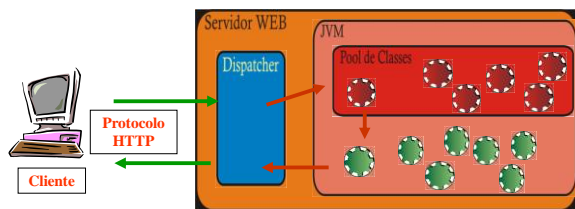


ISEL/DEIETC - Secção de Engenharia de Sistemas

30

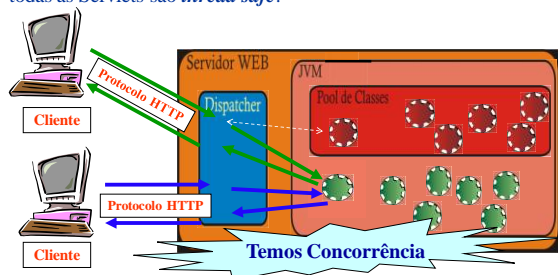


Ambiente executivo das Servlet's



Ambiente Multi-Thread

- Por cada pedido **HTTP** que o **Tomcat** recebe cria uma nova **tarefa** e invoca o método **service()** da Servlet correspondente. Existe apenas uma instância da Servlet. O tomcat assume que todas as Servlets são **thread safe**.



Ambiente Multi-Thread

- Para resolver o problema da concorrência temos as seguintes hipóteses:

- Tratar explicitamente a concorrência utilizando os mecanismos de sincronização que temos à disposição.

- Criar uma Servlet que implemente o interface **SingleThreadModel**. É apenas uma marker para informar o Servlet container que a Servlet não é **ThreadSafe** e que este terá de garantir que não existem duas tarefas a invocar, simultaneamente, o método **service** da mesma instância. Existem duas hipóteses possíveis:

- Ou mantém uma instância que atende os pedidos à vez;
- Ou cria uma pool de instâncias para cada uma atender uma tarefa.

O Tomcat por omissão assume esta hipótese.

O Tomcat cria uma pool de instâncias para tratar a concorrência. Atenção no uso de variáveis da instância em particular.



Troca de Informação browser/servidor/servlet

- Além da informação que o utilizador "gerou" as Servlets têm à sua disposição informação sobre:

- O servidor (web);
- O cliente (browser);
- O pedido;

- Os métodos da `javax.servlet.http.HttpServlet` e suas ascendentes permitem obter essas informações.



Troca de Informação browser/servidor/servlet

- Informação sobre o servidor Web:

- Nome do servidor;
- Porto do servidor;
- Versão do servidor;
- Directoria raiz do servidor;



Troca de Informação browser/servidor/servlet

- Informação sobre o cliente (*browser*):

- Nome do *host* cliente;
- Endereço do cliente;
- Tipo de autenticação;
- Nome do utilizador;



Troca de Informação browser/servidor/servlet

- Informação sobre o pedido:
 - Protocolo usado (versão HTTP);
 - Método usado;
 - Tipo do conteúdo;
 - Dimensão do conteúdo;
 - Tipos de MIMEs aceites pelo *browser*;
 - Informação sobre software do *browser*;
 - Link usado pelo cliente;

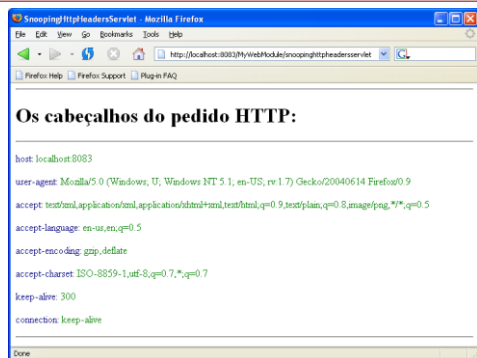


Exemplo – Snooping HTTP Headers

```
public class SnoopingHttpHeadersServlet extends HttpServlet {
    //Process the HTTP Get request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>SnoopingHttpHeadersServlet</title></head>");
        out.println("<body>");
        out.println("<hr><h1> Os cabeçalhos do pedido HTTP: </h1><hr>");
        Enumeration enumHeader = request.getHeaderNames();
        while (enumHeader.hasMoreElements()) {
            String name = (String) enumHeader.nextElement();
            String value = request.getHeader(name);
            out.println("<p><font color='\"#000080\"'>\" + name +
                "</font>: <font color='\"#008000\"'>\" + value + "</font></p>");
        }
        out.println("<hr>");
        out.println("</body></html>");
    }
}
```



Exemplo – Snooping HTTP Headers



Estado entre Pedidos

- O protocolo HTTP é um protocolo sem estado entre pedidos;
- Assim sendo como é que se consegue guardar informação de estado?

Nota: Não se pode usar o endereço IP pois o endereço pode ser o de um *proxy* que serve múltiplos clientes.



Estado entre Pedidos

- Uma das maiores vantagens de utilizar as Servlets é a facilidade de transformar, transparentemente, o protocolo HTTP (*stateless*), numa sequência de actividades possibilitando uma *web application* parecer uma aplicação normal.
- Utilizando Servlets existem vários métodos para manter o estado entre pedidos:
 - Autenticação de utilizadores;**
 - Campos escondidos;**
 - Rescrita dos URLs;**
 - Utilização de cookies;**
 - Usando a API (Session Tracking).**



Estado entre Pedidos

- Autenticação de utilizadores:**
 - O administrador do servidor Web restringe uma área do servidor;
 - No caso de o utilizador ser válido, pode-se obter o nome do utilizador através do método `getRemoteUser()`;





Estado entre Pedidos

Autenticação de utilizadores:

- É simples de implementar;
- Funciona de um modo independente do local onde o cliente se encontra;
- É necessária uma conta para cada utilizador que acede ao recurso.

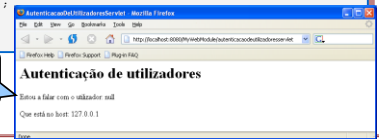


Estado entre Pedidos

```
public class AutenticacaoDeUtilizadoresServlet extends HttpServlet {
    //Process the HTTP Get request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>AutenticacaoDeUtilizadoresServlet</title></head>");
        out.println("<body>");
        out.println("<h1>Autenticação de utilizadores</h1>");
        out.println("<p>Estou a falar com o utilizador: " + request.getRemoteUser() + "</p>");
        out.println("<p>Que está no host: " + request.getRemoteHost() + "</p>");
        out.println("</body></html>");
    }
}
```

O Browser é responsável por enviar a informação referente ao utilizador. As definições de segurança normalmente não o permitem.



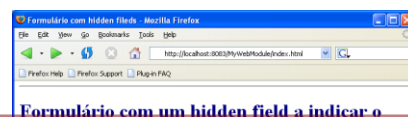
Estado entre Pedidos

Campos escondidos:

- Neste caso a informação de estado é mantida como atributos de um formulário mas os atributos estão escondidos;
- Do ponto de vista de quem processa os dados não existe distinção entre processar um campo não escondido e um escondido.



Estado entre Pedidos



```
<html><head><title>Formulário com hidden fields</title></head>
<body><div>
<font color="#000080"><h1>Formulário com um hidden field a indicar o estado no fluxo de
píasecute/ginas</h1></font>
<form method="GET" action="camposescondidoservlet">
<h1>Insira o seu username:</h1><input type="text" name="username"/>
<h1>Insira a sua password:</h1><input type="password" name="password"/>
<input type="hidden" name="estado" value="login"/>

<input type="submit" value="Submeter"/>
<input type="reset" value="Limpar"/>
</form>
</body></html>
```

Hidden field com a informação que o utilizador agora está na página de login.



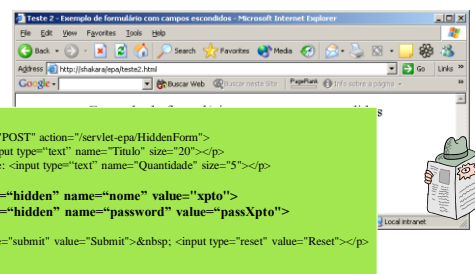
Estado entre Pedidos

```
public class CamposEscondidosServlet extends HttpServlet {
    //Process the HTTP Get request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String username = request.getParameter("username");
        if (username == null) { username = ""; }
        String password = request.getParameter("password");
        if (password == null) { password = ""; }
        String estado = request.getParameter("estado");
        if (estado == null) { estado = ""; }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>CamposEscondidosServlet</title></head>");
        out.println("<body>");
        out.println("<h1>Exemplo de campos escondidos</h1>");
        out.println("<h2>Recebi os seguintes atributos na mensagem http:</h2>");
        out.println("<p>Username: " + username + "</p>");
        out.println("<p>Password: " + password + "</p>");
        out.println("<p>Estado: " + estado + "</p>");
        out.println("</body></html>");
    }
}
```



Estado entre Pedidos



```
<form method="POST" action="/servlet-epa/HiddenForm">
<p>Titulo: <input type="text" name="Titulo" size="20"></p>
<p>Quantidade: <input type="text" name="Quantidade" size="5"></p>

<input type="hidden" name="nome" value="xpto">
<input type="hidden" name="password" value="passXpto">

<p><input type="submit" value="Submit"> &nbsp; <input type="reset" value="Reset"></p>
</form>
```




Estado entre Pedidos

Rescrita dos URLs:

- Com esta técnica modifica-se ou rescreve-se o URL acedido:

- @extra path information;
- @parâmetros adicionais;
- @URL customizados;

http://shakara/servlet-epa/URLRewriting
 http://shakara/servlet-epa/URLRewriting/476
 http://shakara/servlet-epa/URLRewriting?IdSessao=476
 http://shakara/servlet-epa/URLRewriting;\$IdSessao\$476



Estado entre Pedidos

Utilização de cookies:

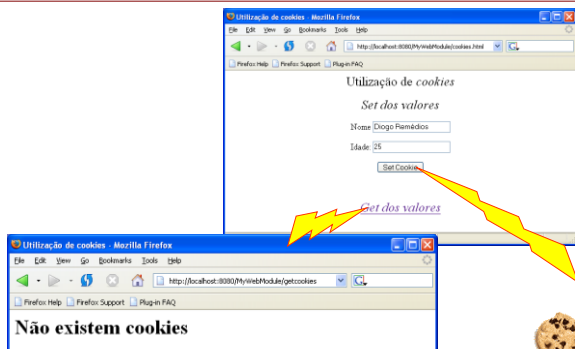
- Um *cookie* não é mais do que um conjunto de dados que é enviado do servidor Web para o *browser* e que mais tarde é reenviado para o servidor pelo *browser*;
- Um *cookie* pode identificar, inequivocamente, um cliente;

Utilização de cookies (restrições):

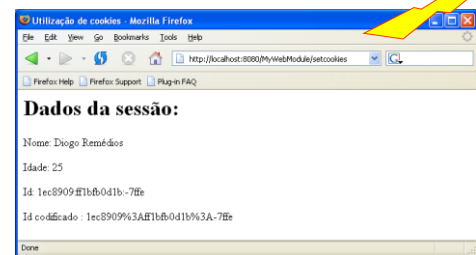
- Um *browser* pode não aceitar *cookies*;
- Os *browsers* podem limitar-se a aceitar apenas 20 *cookies* por servidor, num total de 300 por utilizador;
- Os *browsers* podem limitar a dimensão dos *cookies* a 4096 bytes (4 Kb).



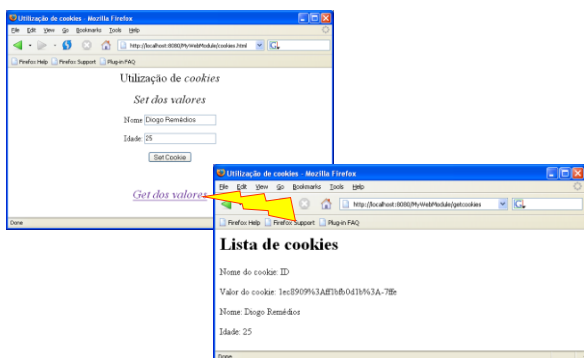
Exemplo de Utilização de cookies



Exemplo de Utilização de cookies



Exemplo de Utilização de cookies



Exemplo de Utilização de cookies

```
public class SetCookies extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String nome = request.getParameter("Nome");
        String idade = request.getParameter("Idade");
        UserData userData = new UserData(nome, idade);
        PrintWriter output = response.getWriter();
        response.setContentType("text/html");
        Cookie cookieId = new Cookie("ID", userData.getEncodedId());
        response.addCookie(cookieId);
        output.println("<HTML><HEAD><TITLE>Utilização de cookies</TITLE></HEAD>");
        output.println("<BODY><H1>Dados da sessão:</H1>");
        output.println("<P>Nome: " + nome + "</P>");
        output.println("<P>Idade: " + idade + "</P>");
        output.println("<P>Id: " + userData.getId() + "</P>");
        output.println("<P>Id codificado: " + userData.getEncodedId() + "</P>");
        output.println("</BODY></HTML>");
    }
}
```





Exemplo de Utilização de cookies

```
public class GetCookies extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter output = response.getWriter();
        response.setContentType("text/html");
        output.println("<HTML>");
        output.println("<HEAD><TITLE>Utilização de cookies</TITLE></HEAD>");
        output.println("<BODY>");

        Cookies[] cookies = request.getCookies();
        if (cookies==null)
            output.println("<H1>Não existem cookies</H1>");
        else {
            output.println("<H1>Lista de cookies</H1>");
            for(int indexCookie=0; indexCookie<cookies.length; ++indexCookie) {
                output.println("<P>Nome do cookie: " + cookies[indexCookie].getName());
                output.println("<P>Valor do cookie: " + cookies[indexCookie].getValue());
                UserData userData = new UserData( cookies[indexCookie].getValue() );
                output.println("<P>Nome: " + userData.getUserName() + "</P>");
                output.println("<P>Idade: " + userData.getIdade() + "</P>");
            }
        }
        output.println("</BODY>");
        output.println("</HTML>");
    }
}
```



55



Estado entre Pedidos - Session Tracking

Usando a API (Session Tracking):

- Qualquer servidor que suporte Servlets deve suportar a API "Session Tracking";
- A implementação mínima para a versão JSDK 2.0 é suportada com base em cookies.

É criada uma instância da classe *javax.servlet.http.HttpSession* que funciona como um *container* que fica residente no servidor. Guarda todo o tipo de dados que forem necessários:

- Informação do perfil do utilizador;
- Preferências do utilizador;
- Seleccções efectuadas, por exemplo, produtos num carrinho de compras;
- Etc...



INEL-DEITC - Secção de Engenharia de Sistemas

56



Estado entre Pedidos - Session Tracking

Métodos mais relevantes na *javax.servlet.http.HttpSession*

- java.lang.String getId()*; Retorna o identificador único atribuído à sessão.
- long getCreationTime()*; Retorna a data de criação da sessão.
- java.lang.Object getAttribute(java.lang.String name)*; Retorna o atributo guardado com a chave name.
- java.util.Enumeration getAttributeNames()*; Retorna um enumerado com todos os nomes dos atributos guardados numa dada sessão.
- void setAttribute(java.lang.String name, java.lang.Object value)*; Insere na sessão um objecto (value) associado à chave (name).
- void removeAttribute(java.lang.String name)*; Remove o atributo com a chave indicada.
- boolean isNew()*; Indica se a sessão foi criada como resultado.
- void invalidate()*; Invalida a sessão libertando as referências para os objectos que esta contém.



INEL-DEITC - Secção de Engenharia de Sistemas

57



Exemplo - Acesso a Correio Electrónico

Preteende-se aceder a uma conta de correio electrónico usando um conjunto de páginas Web.



INEL-DEITC - Secção de Engenharia de Sistemas

58



Exemplo - Acesso a Correio Electrónico

```
POST /servlet-epa/WebEmail HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, */*
Referer: http://shakara/epa/Exemplo10.html
Accept-Language: pt
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
Host: shakara
Content-Length: 36
Connection: Keep-Alive
Cache-Control: no-cache

Servidor=sol&Nome=cajo&Password=cajo
```



Efectua pedido



INEL-DEITC - Secção de Engenharia de Sistemas

59



Exemplo - Acesso a Correio Electrónico

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Tue, 10 Sep 2002 10:35:38 GMT
Set-cookie: NSES40Session=2%253A3d7dcafa%253Aab3fde886430d3f6;path=/; expires=Tue, 10-Sep-2002 11:05:38 GMT
Content-type: text/html
Content-length: 829

<HTML>
...
</HTML>
```



Obtém resposta

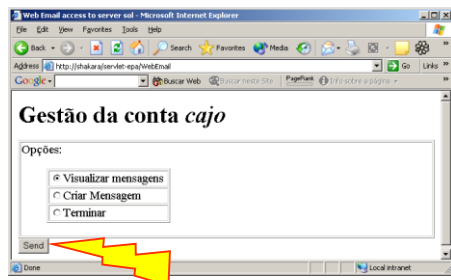


INEL-DEITC - Secção de Engenharia de Sistemas

60



Exemplo - Acesso a Correio Electrónico



ISEL/DEIETC - Secção de Engenharia de Sistemas

61



Exemplo - Acesso a Correio Electrónico

```
POST /servlet-epa/WebEmail HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, application/vnd.ms-powerpoint, */*
Referer: http://shakara/servlet-epa/WebEmail
Accept-Language: pt
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
Host: shakara
Content-Length: 10
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: NSES40Session=2%253A3d7dcafa%253Aab3fde886430d3f6

Opcoes=Vis
```



Efectua pedido



ISEL/DEIETC - Secção de Engenharia de Sistemas

62



Exemplo - Acesso a Correio Electrónico

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Tue, 10 Sep 2002 10:47:01 GMT
Set-cookie: NSES40Session=2%253A3d7dcafa%253Aab3fde886430d3f6;path=/;
expires=Tue, 10-Sep-2002 11:17:02 GMT
Content-type: text/html
Content-length: 149

<HTML>
<HEAD><TITLE>Web Email access</TITLE></HEAD>
<BODY>
<H1>User Name: cajo</H1>
<H1>Password: cajo</H1>
</BODY>
</HTML>
```



Obtém resposta



ISEL/DEIETC - Secção de Engenharia de Sistemas

63



Exemplo - Acesso a Correio Electrónico



ISEL/DEIETC - Secção de Engenharia de Sistemas

64



Exemplo - Acesso a Correio Electrónico

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

/**
 * Acesso a correio electrónico através de um browser.
 */
public class WebEmail extends HttpServlet {
    PrintWriter output;
    HtmlGenerator page;
    HttpSession session;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        output = response.getWriter();
        page = new HtmlGenerator(output);
        session = request.getSession(true);

        response.setContentType("text/html");
        Boolean validUser = (Boolean)session.getAttribute("WebEmail.validUser");
        if ( (validUser==null) || (validUser.booleanValue()==false) )
            CheckLogin(request, response);
        else
            ProcessEmail(request, response);
    }
}
```

ISEL/DEIETC - Secção de Engenharia de Sistemas

65



Exemplo - Acesso a Correio Electrónico

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    output = response.getWriter();
    page = new HtmlGenerator(output);
    session = request.getSession(true);

    response.setContentType("text/html");
    Boolean validUser = (Boolean)session.getAttribute("WebEmail.validUser");
    if ( (validUser==null) || (validUser.booleanValue()==false) )
        CheckLogin(request, response);
    else
        ProcessEmail(request, response);
}
}
```

ISEL/DEIETC - Secção de Engenharia de Sistemas

66



Exemplo - Acesso a Correio Electrónico

```
private void CheckLogin(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String user      = request.getParameter("Nome");
    String password  = request.getParameter("Password");
    String server    = request.getParameter("Servidor");

    Pop3Client popClient = new Pop3Client(server, user, password);
    if ( popClient.isUserOk() ) {
        session.setAttribute("WebEmail.validUser", new Boolean(true) );
        session.setAttribute("WebEmail.user", user);
        session.setAttribute("WebEmail.password", password);
    }
}
```



Exemplo - Acesso a Correio Electrónico

```
...
page.beginPage("Web Email access to server " + server);
page.html("Gestão da conta <i>" + user + "</i>");
page.beginForm("POST", "/servlet-epa/WebEmail");
buildForm(); page.endForm(); page.endPage();
}
else {
    session.setAttribute("WebEmail.validUser", new Boolean(false) );
    response.setHeader("Refresh", "3; URL=/epa/Exemplo10.html");
    page.beginPage("Web Access to server Krillin");
    page.html("Bad user name or password."); page.endPage();
}
}
```



Exemplo - Acesso a Correio Electrónico

```
private void ProcessEmail(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    String user      = (String)session.getAttribute("WebEmail.user");
    String password  = (String)session.getAttribute("WebEmail.password");

    page.beginPage("Web Email access");

    page.html("User Name: " + user);
    page.html("Password: " + password);

    page.endPage();
}
};
```



Exemplo - Loja online - WebShop

The screenshot shows the WebShop online interface. It features a header with the shop name and navigation links. Below the header, there is a section titled 'Lista de Produtos:' which lists various electronic items like LCD TVs, USB drives, and digital cameras, each with a price and an 'Adicionar ao carrinho' button. To the right, there is a 'WebShop - Compras online' section showing the shopping cart status: 'O seu carrinho ainda está vazio'.



Exemplo - WebShop

The screenshot shows the WebShop online interface. It features a header with the shop name and navigation links. Below the header, there is a section titled 'Lista de Produtos:' which lists various electronic items like LCD TVs, USB drives, and digital cameras, each with a price and an 'Adicionar ao carrinho' button. To the right, there is a 'WebShop - Compras online' section showing the shopping cart status: 'O seu carrinho ainda está vazio'.



Exemplo - WebShop

- Pretende-se aceder a uma loja online usando um conjunto de páginas Web.
- Neste exemplo é guardado na sessão um objecto que é o carrinho de compras. Este contém um vector com os vários itens escolhidos.
- A leitura dos produtos existentes é feita no init();

```
...
public void init() { // ler do ficheiro WebShop.txt e preencher o itemVector
    try {
        persistentDlc = new String(getServletContext().getRealPath("/") );
        BufferedReader bufferedReader = new BufferedReader(new FileReader(persistentDlc + "WebShop.txt"));
        String s;
        while((s = bufferedReader.readLine()) != null){
            //ler a contagem e o nome da musica
            String[] res = s.split("_sep_");
            Item product = new Item(res[0], res[1], res[2]);
            itemVector.add(product); //adicionar o novo produto
        }
        bufferedReader.close();
    }
    catch (Exception e) { log("Excepcao a ler a os produtos do ficheiro WebShop.txt..."); }
}
...

```

ItemVector é global para que todas as threads que executem o service possam saber quais os produtos existentes na loja.



Exemplo - WebShop

- Caso seja o primeiro acesso criar o carrinho de compras e guardar na sessão.

```
...
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                throws ServletException, IOException {

    HttpSession mySession = request.getSession(true);
    PrintWriter out = response.getWriter();
    response.setContentType("text/html");

    if(mySession.getAttribute("WebShop_ShoppingCart") == null){
        // there is no shopping cart, let's create one
        ShoppingCart sk = new ShoppingCart();
        mySession.setAttribute("WebShop_ShoppingCart", sk);
        output.println("<p> Criou o Shopping kart na sessão: "+mySession.getId() + "</p>");
    }
}
```



Exemplo - WebShop

- Sempre que é necessário o acesso ao carrinho das compras é utilizada a sessão.

```
... // exemplo de adicionar um produto ao carrinho
if (request.getParameter("estado").compareTo("Adicionar") == 0)
    if (request.getParameter("productId") != null) {

        // adicionar produto ao carrinho
        String prodId = request.getParameter("productId");
        ShoppingKart mysk = (ShoppingKart) mySession.getAttribute("Webshop.ShoppingKart");
        Iterator existingItemIter = itemVector.iterator();
        while (existingItemIter.hasNext()) {
            Item existingItem = (Item) existingItemIter.next();
            if (existingItem.getId().compareTo(prodId) == 0) //produto a adicionar

                mysk.add(existingItem);

            output.println("<h3>produto </h3>");
            output.println("<div>Adicione </div>");
            + existingItem.getName() + "<br><font>=</font>";
            + "400' font-style: italic'><font size='4'>Adicione </font>";
            + existingItem.getPrice() + " €" + "<br>";
            + "<div>Foi adicionado ao seu carrinho de compras </div>";
            + "<p>a href='</p>";
        }
    }
} // Element Added to the shopping kart
} ...
```



Conclusões

- As Servlets são uma alternativa no desenvolvimento de páginas dinâmicas geradas no contexto de um servidor Web;
- Existem outras alternativas:
 - Programas CGI;
 - Extensões à API do servidor;
 - Active Server Pages – ASP;
 - JavaServer Pages – JSP;
 - JavaScript suportado pelo servidor.
- Não necessitam que o *browser* disponha de uma máquina virtual Java;
- São portáveis uma vez que estão escritas em Java e obedecem a uma API conhecida;



Conclusões

- Podem usar todo o potencial da linguagem Java;
- São eficientes. Uma vez carregada em memória a Servlet fica disponível para satisfazer mais pedidos;
- São seguras:
 - Strong type check da linguagem Java;
 - Utilização de excepções;
 - Utilização de Security Managers.
- O código desenvolvido é simples. A própria API disponibiliza objectos específicos para certas funcionalidades;



Conclusões

- As Servlets podem ser vistas como uma extensão ao servidor;
- Como estão integradas no servidor permitem uma maior interacção entre a Servlet e o servidor Web;
- O código é interpretado, o que, face a outras alternativas pode fazer com que a Servlet tenha um desempenho pior caso a implementação não recorra à tecnologia JIT;
- A utilização de Servlets pode não ser uma boa opção na geração de formulários se a quantidade de informação gerada for muito grande.



Lista de Exemplos Demonstrativos

- 📁 **Índice** [Exemplos de servlets disponíveis na PHOENIX, na sua área de web]
 - 📁 **Hello World:** [HelloWorldServlet]
 - 📄 Simples; [HelloWorldServlet.java]
 - 📄 Com formulário para recolher o nome; [HelloToServlet.java]
 - 📁 **Concorrência num contador de acessos:** [ConcorrenciaServlet]
 - 📄 Servlet Problemática; [ServletErrada.java]
 - 📄 Solução 1; [ServletBoa1.java]
 - 📄 Solução 2; [ServletBoa2.java]
 - 📁 **Listar cabeçalhos http:** [SnoopingHttpHeadersServlet]
 - 📄 Lista todos os cabeçalhos http; [SnoopingHttpHeadersServlet.java]
 - 📄 Lista alguns cabeçalhos http; [SnoopingHeaderCompCGIServlet.java]



Lista de Exemplos Demonstrativos

- **Manutenção de informação de estado:** [ManutencaoDeEstado]
 - ⊗ Autenticação de Utilizadores; [AutenticacaoDeUtilizadoresServlet.java]
 - ⊗ Campos escondidos; [CamposEscondidosServlet.java]
 - ⊗ Rescrita do URL; [index.html]
 - ⊗ Cookies; [setcookies.java e getcookies.java]

- **WebShop :** [WebShopServlet]
 - ⊗ Exemplo guardando informação na sessão; [WebShop.java]
[ShoopingKart.java]
[Item.java]



Referências de Estudo

- Servlet API e J2SE 1.4.2 API
 - <http://www.deetc.isel.ipl.pt/engenhariadesi/cadeiras/scd/documentacao.htm>

- Cookies Netscape
 - http://www.netscape.com/newsref/std/cookie_spec.html

- API's de Servidores WEB
 - <http://www.apache.org/docs/API.html>

- Classes utilitárias
 - <http://www.acme.com/java/software/>



Glossário de Termos

- WWW – World-Wide Web;
- HTML – HyperText Markup Language;
- HTTP – HyperText Transfer Protocol;
- URL – Uniform Resource Locator;
- URI – Uniform Resource Identifier;
- URN – Uniform Resource Name;
- RFC – Request For Comments;
- CGI – Common Gateway Interface;
- API – Application Programming Interface;
- ASP – Active Server Pages;
- JSP – JavaServer Pages;



Bibliografia

- RFC's:
 - 1945 – HTTP/1.0;
 - 2068 – HTTP/1.1;
 - 2109 – HTTP *State Management Mechanism*;

- HTML Sourcebook, 3th Edition;
 - Graham, Ian S. – Wiley Computer Publishing

- Java Servlet Programming;
 - Hunter, J. & Crawford, W – O' Reilly

- Professional JSP 2nd Edition
 - Karl Avedal, Robert Burdick, ..., Steve Wilkinson
 - Publisher: Wrox Press Ltd, ISBN: 1861004958

