

Módulo IVa - Servlets

Prof. Ismael H F Santos

Ementa

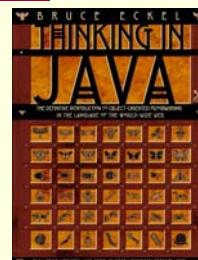
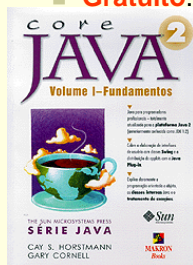
- **Módulo IVa – Servlets**
 - Overview Servlets e JSP
 - Ciclo de Vida
 - HTTP Servlets
 - Gerenciamento de Sessão
 - Contêineres Web – Apache Tomcat
 - Cookies

Bibliografia

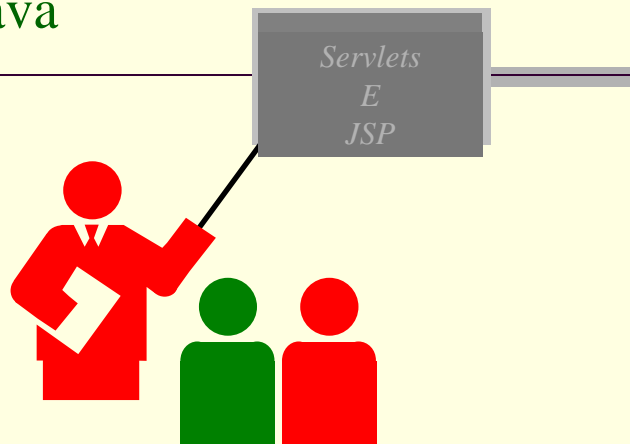
- *Linguagem de Programação JAVA*
 - Ismael H. F. Santos, Apostila UniverCidade, 2002
- *The Java Tutorial: A practical guide for programmers*
 - Tutorial on-line: <http://java.sun.com/docs/books/tutorial>
- *Java in a Nutshell*
 - David Flanagan, O'Reilly & Associates
- *Just Java 2*
 - Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.
- *Java 1.2*
 - Laura Lemay & Rogers Cadenhead, Editora Campos

Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
 - Volume 1 (Fundamentos)
 - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
 - **Gratuito.** <http://www.mindview.net/Books/TIJ/>



POO-Java



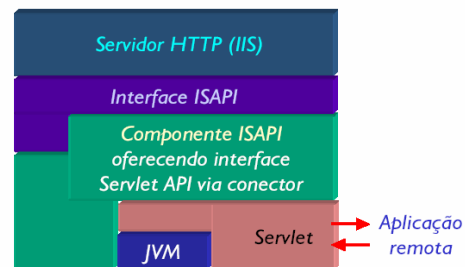
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

5

Servlet API

- API independente de plataforma e praticamente independente de fabricante
- Componentes são escritos em Java e se chamam **servlets**
- Como os componentes SAPI proprietários, rodam dentro do servidor, mas através de uma Máquina Virtual Java
- Disponível como 'plug-in' ou conector para servidores que não o suportam diretamente
 - Desenho ao lado mostra solução antiga de conexão com IIS
- Nativo em servidores Sun, IBM, ...



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

6

Java e Aplicações Web

- **Servlets** e **JavaServer Pages (JSP)** são as soluções Java para estender o servidor HTTP
 - Suportam os **métodos de requisição** padrão HTTP (GET, POST, HEAD, PUT, DELETE, OPTIONS, TRACE)
 - Geram **respostas** compatíveis com HTTP (códigos de status, cabeçalhos RFC 822)
 - Interação com **Cookies**
- Além dessas tarefas básicas, também
 - Suportam **filtros**, que podem ser chamados em cascata para tratamento de dados durante a transferência
 - Suportam **controle de sessão** transparentemente através de cookies ou rescrita de URLs (automática)
- É preciso usar um servidor que suporte as especificações de servlets e JSP

Servlets - Introdução

- **Extensão de servidor escrita em Java**
 - Servlets são "applets" (pequenas aplicações) de servidor
 - Podem ser usados para estender qualquer tipo de aplicação do modelo **requisição-resposta**
 - Todo servlet implementa a interface **javax.servlet.Servlet** (tipicamente estende GenericServlet)
- **Servlets HTTP**
 - Extensões para servidores Web
 - Estendem **javax.servlet.http.HttpServlet**
 - Lidam com características típicas do HTTP como métodos GET, POST, Cookies, etc.

Primeiro Servlet

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

public class SimpleServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out;
        response.setContentType("text/html");
        out = response.getWriter();
        String user = request.getParameter("usuario");
        if (user == null)
            user = "World";

        out.println("<HTML><HEAD><TITLE>");
        out.println("Simple Servlet Output");
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>Simple Servlet Output</H1>");
        out.println("<P>Hello, " + user);
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

9

Primeiro JSP

```
<HTML><HEAD>
<TITLE>Simple Servlet Output</TITLE>
</HEAD><BODY>
<%
    String user =
        request.getParameter("usuario");
    if (user == null)
        user = "World";
%>
<H1>Simple Servlet Output</H1>
<P>Hello, <%= user %>
</BODY></HTML>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

10

Pagina recebida no browser

- **Url da requisição**


```
http://servidor/servlet/SimpleServlet?usuario=Rex
```

```
http://servidor/hello.jsp?usuario=Rex
```

- **Código fonte visto no cliente**

```
<HTML><HEAD>
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, Rex
</BODY></HTML>
```

Usando contexto default
ROOT no TOMCAT



JavaBeans

- *Um **JavaBean** é um componente reutilizável que tem como finalidade representar um modelo de dados*
 - *Define convenções para que atributos de dados sejam tratados como "**propriedades**"*
 - *Permite manipulação de suas propriedades, ativação de eventos, etc. através de um framework que reconheça as convenções utilizadas*
- *Basicamente, um **JavaBean** é uma classe Java qualquer, que tem as seguintes características*
 - *Construtor público default (sem argumentos)*
 - *Atributos de dados private*
 - *Métodos de acesso (accessors) e/ou de alteração (mutators) para cada atributo usando a convenção **getPropriedade()** (ou opcionalmente **isPropriedade()** se boolean) e **setPropriedade()***

Java Beans

- Um **Java Bean** nada mais é do que uma classe Java com algumas características especiais:
 - Possui um **construtor default** (lista de parâmetros vazios)
 - Possui propriedades (métodos de acesso **get** e **set** para os seus atributos.). Forma geral:
 - `public TipoPropriedade getPropriedade();`
 - `public void setPropriedade(TipoPropriedade valor);`
- **Exemplo de bean:**
 - Usuario.java (nas paste web dos exemplos)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

13

Exemplo de JavaBean

```
public class UmJavaBean {
    private String msg;
    private int id;

    public JavaBean () {}

    public String getMensagem () {
        return mensagem;
    }
    public void setMensagem(String msg) {
        this.msg = msg;
    }
    public String getId () {
        return mensagem;
    }
    public void metodo () {
        ...
    }
}
```

«Java Bean» UmJavaBean
mensagem :String «RW»
id :int «R»
metodo():void

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

14

Usando JavaBean em pagina JSP

▪ Página JSP que usa *HelloBean.class*

```
<HTML><HEAD>
<jsp:useBean id="hello" class="beans.HelloBean" />
<jsp:setProperty name="hello" property="mensagem"
    param="usuario" />

<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, <jsp:getProperty name="hello"
    property="mensagem" />

</BODY></HTML>
```

```
package beans;

public class HelloBean implements
    java.io.Serializable {

    private String msg;

    public HelloBean() {
        this.msg = "World";
    }

    public String getMensagem() {
        return msg;
    }

    public void setMensagem(String msg) {
        this.msg = msg;
    }
}
```

Imprime: **Hello, World**

Componentes Web

- São aplicações J2EE
- Rodam em um **Web Container** que oferece serviços como repasse de requisições, segurança, concorrência (threads), gerência do ciclo de vida
- São compostos de **servlets** e **páginas JSP** empacotados em um arquivo **WAR** (tipo de JAR)
- O WAR é essencial para implantar o cliente Web J2EE, mas opcional em servidor standalone
- Os componentes de um WAR ocupam um **contexto** que pode ser acessado através de um cliente HTTP (browser)
 - Fisicamente, o contexto representa uma estrutura de diretórios.
 - Logicamente, representa uma aplicação Web.
 - O contexto tem uma estrutura padrão definida em especificação

Contextos em Aplicações Web

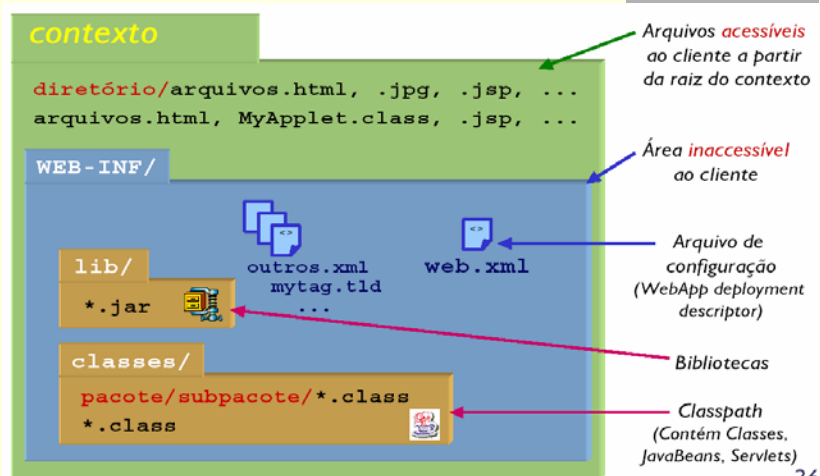
- No **JBoss**, aplicações Web são implantadas copiando contextos compactados em arquivos WAR para a pasta **deploy/**
 - A estrutura da árvore de diretórios deve ser mantida dentro do arquivo WAR
- Todo diretório de contexto tem uma estrutura definida, que consiste de
 - Área de documentos do contexto (**/**), **acessível** externamente
 - Área **inacessível** (**/WEB-INF**), que possui pelo menos um arquivo de configuração padrão (**web.xml**)
 - O **WEB-INF** pode conter ainda **dois** diretórios reconhecidos pelo servidor: (1) um diretório que pertence ao **CLASSPATH** da aplicação (**/WEB-INF/classes**) e (2) outro onde podem ser colocados **JARs** para inclusão no **CLASSPATH** (**/WEB-INF/lib**)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

17

Estrutura de uma Aplicação Web



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

18

Componentes de um contexto

- A raiz define (geralmente) o **nome** do contexto.
 - Na raiz ficam HTMLs, páginas JSP, imagens, applets e outros objetos para download via HTTP

{Contexto}/WEB-INF/web.xml

- Arquivo de configuração da aplicação
- Define parâmetros iniciais, mapeamentos e outras configurações de servlets e JSPs.

{Contexto}/WEB-INF/classes/

- Classpath da aplicação

{Contexto}/WEB-INF/lib/

- Qualquer JAR incluído aqui será carregado como parte do CLASSPATH da aplicação

URL de acesso ao contexto

- A não ser que seja configurado externamente, o **nome do contexto** aparece na URL após o nome/porta do servidor
`http://serv:8080/contexto/subdir/pagina.html`
`http://serv:8080/contexto/servlet/pacote.Servlet`
- Para os documentos no servidor (links em páginas HTML e formulários), a raiz de referência é a **raiz de documentos do servidor**, ou **DOCUMENT_ROOT**: `http://serv:8080/`
- Documentos podem ser achados **relativos ao DOCUMENT_ROOT**
`/contexto/subdir/pagina.html`
`/contexto/servlet/pacote.Servlet`
- Para a configuração do contexto (web.xml), a raiz de referência é a **raiz de documentos do contexto**: `http://serv:8080/contexto/`
- Componentes são identificados **relativos ao contexto**
`/subdir/pagina.html`
`/servlet/pacote.Servlet`

servlet/ é mapeamento virtual definido no servidor para servlets em WEB-INF/classes

Criando um contexto válido

- Para que uma estrutura de diretórios localizada no `webapps/` seja reconhecida como contexto pelo Tomcat, na inicialização, deve haver um arquivo `web.xml` no diretório `WEB-INF` do contexto
 - O arquivo é um arquivo XML e deve obedecer às regras do XML e do DTD definido pela especificação
 - O conteúdo mínimo do arquivo é a *declaração do DTD* e um elemento raiz `<web-app/>`

```
<?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/>
```

- Se houver qualquer erro no `web.xml`, a aplicação não será carregada durante a inicialização

Exemplo configuração (1/3)

```
<web-app>
  <context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
  </context-param>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>example.MyServlet</servlet-class>
    <init-param>
      <param-name>datafile</param-name>
      <param-value>data/data.txt</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>myJSP</servlet-name>
    <jsp-file>/myjsp.jsp</jsp-file>
    <load-on-startup>2</load-on-startup>
  </servlet>
  ...
```

Parâmetro que pode ser lido por todos os componentes

Instância de um servlet

Parâmetro que pode ser lido pelo servlet

Ordem para carga prévia do servlet

Instância de servlet de página JSP

Ordem para pré-compilar JSP

Exemplo configuração (2/3)

```
...
<servlet-mapping>
  <servlet-name>myServlet</servlet-name>
  <url-pattern>/myservlet</url-pattern>
</servlet-mapping>

<session-config>
  <session-timeout>60</session-timeout>
</session-config>

<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

<error-page>
  <error-code>404</error-code>
  <location>/notFound.jsp</location>
</error-page>
...
```

Servlet examples.myServlet foi mapeado à URL /myservlet

Sessão do usuário expira em 30 minutos

Lista de arquivos que serão carregados automaticamente em URLs terminadas em diretório

Redirecionar para esta página em caso de erro 404

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

23

Exemplo configuração (3/3)

```
...
<resource-ref>
  <res-ref-name>jdbc/MeuBanco</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>CONTAINER</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

<env-entry>
  <env-entry-name>valor</env-entry-name>
  <env-entry-value>34.45</env-entry-value>
  <env-entry-type>java.lang.Double</env-entry-type>
</env-entry>
</web-app>
```

Recursos externos acessíveis via JNDI (java:comp/env/jdbc/MeuBanco)

Ligação com nome JNDI em outro contexto pode ser feito em arquivo externo (no EAR ou configuração proprietária, ex: jboss-web.xml)

Variáveis compartilhadas pelo ambiente

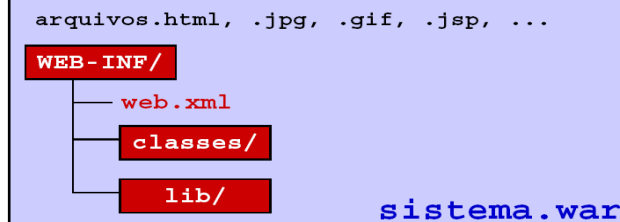
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

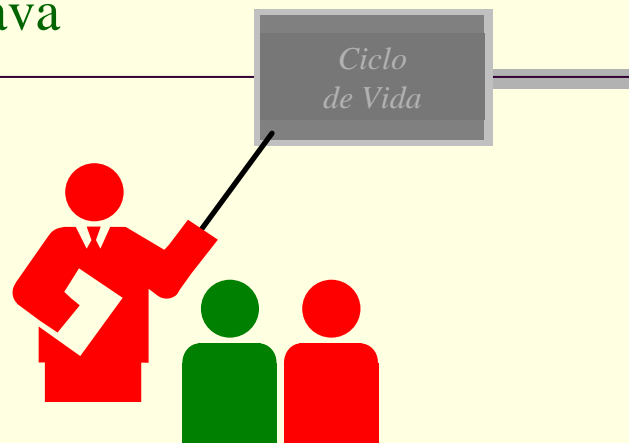
24

Web Archive

- Utilizável no Tomcat e também em servidores J2EE
 - Permite criação de novo contexto automaticamente
 - Coloque JAR contendo estrutura de um contexto no diretório de deployment (*webapps*, no Tomcat)
 - O JAR deve ter a extensão *.WAR*
 - O JAR deve conter *WEB-INF/web.xml* válido
- Exemplo - aplicação: `http://servidor/sistema/`



POO-Java



Servlets - API

Principais classes e interfaces de `javax.servlet`

■ Interfaces

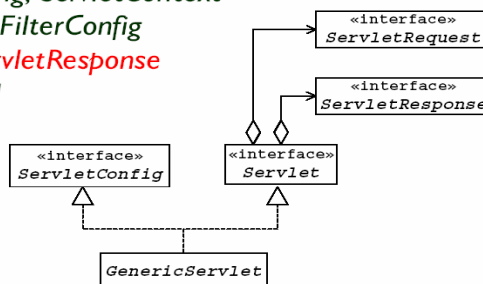
- `Servlet`, `ServletConfig`, `ServletContext`
- `Filter`, `FilterChain`, `FilterConfig`
- `ServletRequest`, `ServletResponse`
- `SingleThreadModel`
- `RequestDispatcher`

■ Classes abstratas

- `GenericServlet`

■ Classes concretas

- `ServletException`
- `UnavailableException`
- `ServletInputStream` e `ServletOutputStream`



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

27

Servlets – Ciclo de Vida

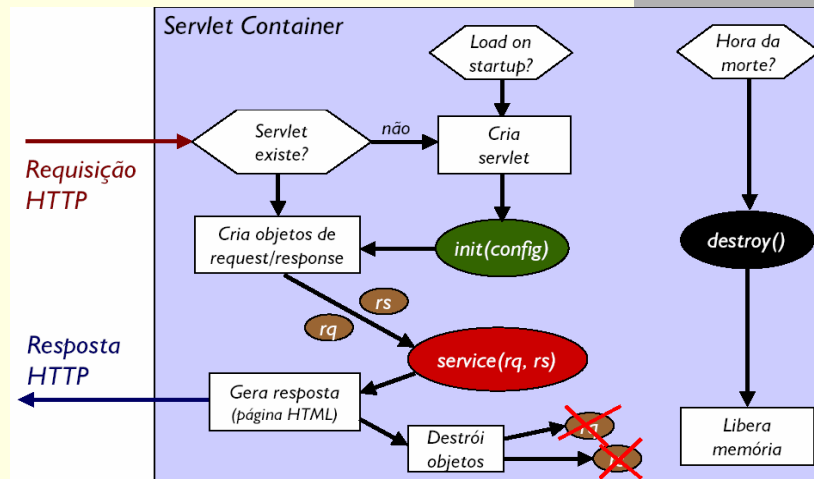
- O ciclo de vida de um servlet é controlado pelo container
- Quando o `servidor` recebe uma requisição, ela é repassada para o container que a delega a um `servlet`. O container
 1. Carrega a classe na memória
 2. Cria uma instância da classe do servlet
 3. Inicializa a instância chamando o método `init()`
- Depois que o servlet foi inicializado, cada requisição é executada em um método `service()`
 - O container cria um objeto de `requisição` (`ServletRequest`) e de `resposta` (`ServletResponse`) e depois chama `service()` passando os objetos como parâmetros
 - Quando a resposta é enviada, os objetos são `destruídos`
- Quando o container decidir remover o servlet da memória, ele o finaliza chamando `destroy()`

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

28

Servlets – Ciclo de Vida



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

29

Servlets – Métodos de Serviço

- São os métodos que implementam operações de resposta executadas quando o cliente envia uma requisição
- Todos os métodos de serviço recebem dois parâmetros: um objeto **ServletRequest** e outro **ServletResponse**
- Tarefas usuais de um método de serviço
 - extrair informações da requisição
 - acessar recursos externos
 - preencher a resposta (no caso de HTTP isto consiste de preencher os cabeçalhos de resposta, obter um stream de resposta e escrever os dados no stream)

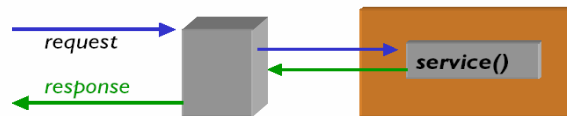
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

30

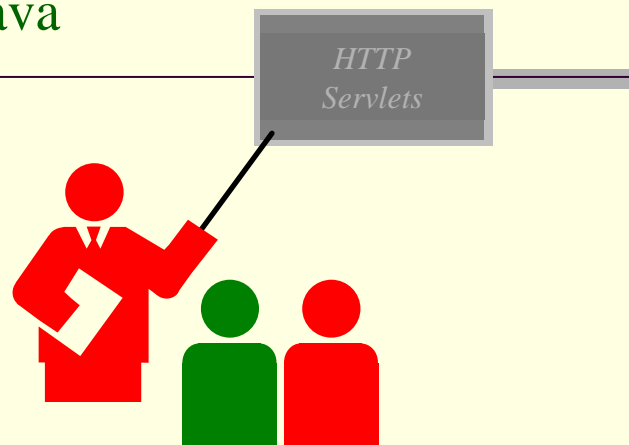
Servlets – Métodos de Serviço (2)

- O método de serviço de um servlet genérico é o método abstrato `service()`
`public void service(ServletRequest, ServletResponse)`
definido em `javax.servlet.Servlet`.
- Sempre que um servidor repassar uma requisição a um servlet, ele chamará o método `service(request, response)`.



- Um servlet genérico deverá sobrepor este método e utilizar os objetos `ServletRequest` e `ServletResponse` recebidos para ler os dados da requisição e compor os dados da resposta, respectivamente

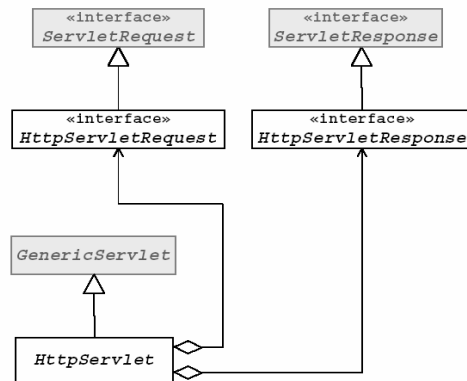
POO-Java



HTTP Servlets - API

Classes e interfaces mais importantes do pacote *javax.servlet.http*

- Interfaces
 - *HttpServletRequest*
 - *HttpServletResponse*
 - *HttpSession*
- Classes abstratas
 - *HttpServlet*
- Classes concretas
 - *Cookie*



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

33

Criando um HTTP servlet

- Para escrever um servlet HTTP, deve-se estender *HttpServlet* e implementar um ou mais de seus métodos de serviço, tipicamente: *doPost()* e/ou *doGet()*

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

public class ServletWeb extends HttpServlet {
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
                      throws IOException {

        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<h1>Hello, World!</h1>");
        out.close();
    }
}
```

April 05

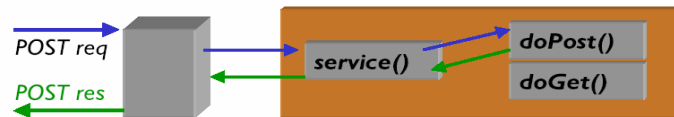
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

34

Métodos de serviço HTTP

- A classe `HttpServlet` redireciona os pedidos encaminhados para `service()` para métodos que refletem os métodos HTTP (GET, POST, etc.):

- `public void doGet(HttpServletRequest, HttpServletResponse)`
- `public void doPost(HttpServletRequest, HttpServletResponse)`
- ... *



- Um servlet HTTP genérico deverá estender `HttpServlet` e implementar **peelo menos um** dos métodos `doGet()` ou `doPost()`

* `doDelete()`, `doTrace()`, `doPut()`, `doOptions()` - Método HEAD é implementado em `doGet()`

Inicialização

- A inicialização de um `GenericServlet`, como o `HttpServlet`, deve ser feita com o método `init()`
- Todos os métodos de config estão no servlet, pois `GenericServlet` implementa `ServletConfig`

```
public void init() throws ServletException {
    String dirImagens =
        getInitParameter("imagens");
    if (dirImagens == null) {
        throw new UnavailableException
            ("Configuração incorreta!");
    }
}
```

Requisição HTTP

- Uma requisição HTTP feita pelo browser tipicamente contém vários cabeçalhos RFC822*

```
GET /docs/index.html HTTP/1.0
Connection: Keep-Alive
Host: localhost:8080
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
Accept: image/gif, image/x-bitmap, image/jpg, image/png, */*
Accept-Charset: iso-8859-1, *
Cookies: jsessionid=G3472TS9382903
```

- Os métodos de `HttpServletRequest` permitem extrair informações de qualquer um deles
 - Pode-se também identificar o método e URL

Obtendo dados da requisição

- Alguns métodos de `HttpServletRequest`
 - Enumeration `getHeaderNames()` - obtém nomes dos cabeçalhos
 - String `getHeader("nome")` - obtém primeiro valor do cabeçalho
 - Enumeration `getHeaders("nome")` - todos os valores do cabeçalho
 - String `getParameter(param)` - obtém parâmetro HTTP
 - String[] `getParameterValues(param)` - obtém parâmetros repetidos
 - Enumeration `getParameterNames()` - obtém nomes dos parâmetros
 - Cookie[] `getCookies()` - recebe cookies do cliente
 - `HttpSession getSession()` - retorna a sessão
 - `setAttribute("nome", obj)` - define um atributo obj chamado "nome".
 - Object `getAttribute("nome")` - recupera atributo chamado nome
 - String `getRemoteUser()` - obtém usuário remoto (se autenticado, caso contrário devolve null)

Resposta HTTP

- Uma resposta HTTP é enviada pelo servidor ao browser e contém informações sobre os dados anexados

```
HTTP/1.0 200 OK
Content-type: text/html
Date: Mon, 7 Apr 2003 04:33:59 GMT-03
Server: Apache Tomcat/4.0.4 (HTTP/1.1 Connector)
Connection: close
Set-Cookie: jsessionid=G3472TS9382903

<HTML>
  <h1>Hello World!</h1>
</HTML>
```

- Os métodos de *HttpServletResponse* permitem construir um cabeçalho

Preenchimento da resposta

- Alguns métodos de *HttpServletResponse*
 - *addHeader*(String nome, String valor) - adiciona cabeçalho HTTP
 - *setContentType*(tipo MIME) - define o tipo MIME que será usado para gerar a saída (text/html, image/gif, etc.)
 - *sendRedirect*(String location) - envia informação de redirecionamento para o cliente (Location: url)
 - *Writer* *getWriter*() - obtém um *Writer* para gerar a saída. Ideal para saída de texto.
 - *OutputStream* *getOutputStream*() - obtém um *OutputStream*. Ideal para gerar formatos diferentes de texto (imagens, etc.)
 - *addCookie*(Cookie c) - adiciona um novo cookie
 - *encodeURL*(String url) - envia como anexo da URL a informação de identificador de sessão (sessionId)
 - *reset*() - limpa toda a saída inclusive os cabeçalhos
 - *resetBuffer*() - limpa toda a saída, exceto cabeçalhos

doGet() e doPost()

- Use **doGet()** para receber requisições **GET**
 - Links clicados ou URL digitadas diretamente
 - Alguns formulários que usam **GET**
- Use **doPost()** para receber dados de formulários
- Se quiser usar ambos os métodos, não sobreponha **service()** mas implemente tanto **doGet()** como **doPost()**

```
public class ServletWeb extends HttpServlet {
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response) {
        processar(request, response);
    }
    public void doPost (HttpServletRequest request,
                       HttpServletResponse response) {
        processar(request, response);
    }
    public void processar (HttpServletRequest request,
                          HttpServletResponse response) {
        ...
    }
}
```

Parâmetros da requisição

- Parâmetros são pares **nome=valor** que são enviados pelo cliente concatenados em strings separados por **&**:
`nome=Jo%E3o+Grand%E3o&id=agente007&acesso=3`
- Parâmetros podem ser passados na requisição de duas formas
 - Se o método for **GET**, os parâmetros são passados em uma única linha no query string, que estende a URL após um **"?"**

```
GET /servlet/Teste?id=agente007&acesso=3 HTTP/1.0
```

- Se o método for **POST**, os parâmetros são passados como um **stream** no corpo na mensagem (o cabeçalho **Content-length**, presente em requisições **POST** informa o tamanho

```
POST /servlet/Teste HTTP/1.0
Content-length: 21
Content-type: x-www-form-urlencoded

id=agente007&acesso=3
```

Obtendo parâmetros da requisição

- Caracteres reservados e maiores que ASCII-7bit são codificados em URLs:
 - Ex: ã = %E3
- Formulários HTML codificam o texto ao enviar os dados automaticamente
- Seja o método POST ou GET, os valores dos parâmetros podem ser recuperados pelo método `getParameter()` de `ServletRequest`, que recebe seu nome

```
String parametro = request.getParameter("nome");
```
- Parâmetros de mesmo nome podem ser repetidos. Neste caso `getParameter()` retornará apenas a primeira ocorrência. Para obter todas use `String[] getParameterValues()`

```
String[] params = request.getParameterValues("nome");
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

43

Gerando a resposta

- Para gerar uma resposta, primeiro é necessário obter, do objeto `HttpServletResponse`, um fluxo de saída, que pode ser de caracteres (`Writer`) ou de bytes (`OutputStream`)

```
Writer out = response.getWriter(); // ou
OutputStream out = response.getOutputStream();
```
- Apenas um deve ser usado. Os objetos correspondem ao mesmo stream de dados
- Deve-se também definir o tipo de dados a ser gerado. Isto é importante para que o cabeçalho `Content-type` seja gerado corretamente e o browser saiba exibir as informações

```
response.setContentType("text/html");
```
- Depois, pode-se gerar os dados, imprimindo-os no objeto de saída (`out`) obtido anteriormente

```
out.println("<h1>Hello</h1>");
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

44

Compilação e implementação

- Para **compilar**, use qualquer distribuição da API
 - O `javax.servlet.jar` distribuído pelo Tomcat em `common/lib/`
 - O `javax.j2ee.jar` distribuído no pacote J2EE da Sun (em `lib/`)
 - O `javax.servlet.jar` do JBoss (`server/default/lib/`)
- Inclua o JAR no seu CLASSPATH ao compilar

```
> javac -classpath ../servlet.jar;. MeuServlet.java
```
- Para **implantar**, copie as classes compiladas para um contexto existente no servidor
 - Jakarta-Tomcat (`webapps/ROOT/WEB-INF/classes`)
 - JBoss: (`server/default/deploy/`)

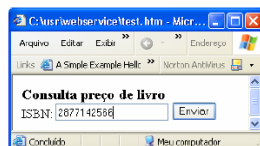
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

45

Execução

- Use
 - `http://localhost:8080/contexto/servlet/nome.do.Servlet`
- Para passar parâmetros
 - Passe-os via URL, acrescentando um `?` seguido dos pares `nome=valor` (separados por `"&"`):
`http://localhost:8080/servlet/Servlet?isbn=123456`
 - Ou escreva um formulário HTML



```
<FORM ACTION="/servlet/Servlet"
METHOD="POST">
<H3>Consulta preço de livro</H3>
<P>ISBN: <INPUT TYPE="text" NAME="isbn">
<INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

46

Instâncias de servlets

- **Uma instância** de um servlet pode ser configurada no web.xml através do elemento `<servlet>`

```
<servlet>
  <servlet-name>myServlet</servlet-name>
  <servlet-class>exemplo.pacote.MyServlet</servlet-class>
  <!-- elementos de configuração opcionais aqui -->
</servlet>
```

- `<servlet-name>` e `<servlet-class>` são obrigatórios
- É uma boa prática escolher nomes de servlets seguindo as **convenções Java**
 - Use caixa mista, colocando em maiúsculas cada palavra, mas comece com letra minúscula. Ex: **banco**, **pontoDeServico**
- Pode-se criar **múltiplas instâncias** da mesma classe definindo blocos servlet com `<servlet-name>` diferentes
 - Não terão muita utilidade a não ser que tenham também configuração diferente e mapeamentos diferentes

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

47

Servlet alias – mapeamento no web.xml

- É uma boa prática definir **alias** para os servlets
 - Nomes grandes são difíceis de digitar e lembrar
 - Expõem detalhes sobre a implementação das aplicações
- Para definir um mapeamento de servlet é necessário usar `<servlet>` e `<servlet-mapping>`
- `<servlet-mapping>` associa o nome do servlet a um padrão de URL **relativo ao contexto**. A URL pode ser
 - Um caminho **relativo ao contexto** iniciando por /
 - Uma extensão de arquivo, expresso da forma ***.extensao**

```
<servlet>
  <servlet-name>umServlet</servlet-name>
  <servlet-class>pacote.subp.MeuServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>umServlet</servlet-name>
  <url-pattern>/programa</url-pattern>
</servlet-mapping>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

48

Sintaxe de mapeamentos

- **Mapeamento exato**
 - Não aceita `/nome/` ou `/nome/x` na requisição
 - `<url-pattern>/nome</url-pattern>`
 - `<url-pattern>/nome/subnome</url-pattern>`
- **Mapeamento para servlet default**
 - Servlet é chamado se nenhum dos outros mapeamentos existentes combinar com a requisição
 - `<url-pattern>/</url-pattern>`
- **Mapeamento de caminho**
 - Aceita texto adicional (path info) após nome do servlet na requisição
 - `<url-pattern>/nome/*</url-pattern>`
 - `<url-pattern>/nome/subnome/*</url-pattern>`
- **Mapeamento de extensão**
 - Arquivos com a extensão serão redirecionados ao servlet
 - `<url-pattern>*.ext</url-pattern>`

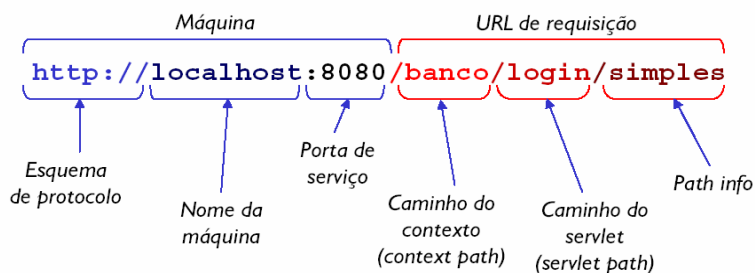
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

49

Anatomia de uma URL

- Diferentes partes de uma URL usada na requisição podem ser extraídas usando métodos de `HttpServletRequest`
 - `getContextPath()`: `/banco`, na URL abaixo
 - `getServletPath()`: `/login`, na URL abaixo
 - `getPathInfo()`: `/simples`, na URL abaixo



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

50

ServletConfig

- A interface **ServletConfig** serve para que um servlet possa ter acesso a informações de configuração definidas no web.xml
- Todo servlet implementa **ServletConfig** e, portanto, tem acesso aos seus métodos
- Principais métodos de interesse
 - String **getInitParameter(String nome)**: lê um parâmetro de inicialização `<init-param>` do web.xml
 - Enumeration **getInitParameterNames()**: obtém os nomes de todos os parâmetros de inicialização disponíveis
- Os métodos de **ServletConfig** devem ser chamados no método **init()**, do servlet

Parâmetros de inicialização

- Parâmetros de inicialização podem ser definidos para cada instância de um servlet usando o elemento **<init-param>** dentro de **<servlet>**
 - Devem aparecer depois de **<servlet-name>** e **<servlet-class>** (lembre-se que a ordem foi definida no DTD)
 - Requer dois sub-elementos que definem o nome do atributo e o seu valor

```
<servlet>
  <servlet-name>umServlet</servlet-name>
  <servlet-class>pacote.subp.MeuServlet</servlet-class>
  <init-param>
    <param-name>dir-imagens</param-name>
    <param-value>c:/imagens</param-value>
  </init-param>
  <init-param> ... </init-param>
</servlet>
```

Lendo Parâmetros de inicialização

- Parâmetros de inicialização podem ser lidos no método `init()` e guardados em variáveis de instância para posterior uso dos métodos de serviço

```
private java.io.File dirImagens = null;

public void init() throws ServletException {
    String dirImagensStr =
        getInitParameter("dir-imagens");
    if (dirImagens == null) {
        throw new UnavailableException
            ("Configuração incorreta!");
    }
    dirImagens = new File(dirImagensStr);
    if (!dirImagens.exists()) {
        throw new UnavailableException
            ("Diretorio de imagens nao existe!");
    }
}
```

ServletContext

- A interface `ServletContext` encapsula informações sobre o contexto ou aplicação
- Cada servlet possui um método `getServletContext()` que devolve o contexto atual
 - A partir de uma referência ao contexto atual pode-se interagir com o contexto e compartilhar informações entre servlets
- Principais métodos de interesse de `ServletContext`
 - `String getInitParameter(String)`: lê parâmetros de inicialização do contexto (não confunda com o método similar de `ServletConfig!`)
 - `Enumeration getInitParameterNames()`: lê lista de parâmetros
 - `InputStream getResourceAsStream()`: lê recurso localizado dentro do contexto como um `InputStream`
 - `setAttribute(String nome, Object)`: grava um atributo no contexto
 - `Object getAttribute(String nome)`: lê um atributo do contexto
 - `log(String mensagem)`: escreve mensagem no log do contexto

Inicialização de contexto

- No `web.xml`, `<context-param>` vem antes de qualquer definição de `servlet`

```
<context-param>
  <param-name>tempdir</param-name>
  <param-value>/tmp</param-value>
</context-param>
```

- No `servlet`, é preciso primeiro obter uma instância de `ServletContext` antes de ler o parâmetro

```
ServletContext ctx = getServletContext();
String tempDir = ctx.getInitParameter("tempdir");
if (tempDir == null) {
    throw new UnavailableException("Configuração errada");
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

55

Carregamento de arquivos no contexto

- O método `getResourceAsStream()` permite que se localize e se carregue qualquer arquivo no contexto sem que seja necessário saber seu caminho completo
 - Isto é importante pois contextos podem ser usados em diferentes servidores e armazenados em arquivos WAR
- Exemplo

```
ServletContext ctx = getServletContext();
String arquivo = "/WEB-INF/usuarios.xml";
InputStream stream = ctx.getResourceAsStream(arquivo);
InputStreamReader reader =
    new InputStreamReader(stream);
BufferedReader in = new BufferedReader(reader);
String linha = "";
while ( (linha = in.readLine()) != null) {
    // Faz alguma coisa com linha de texto lida
}
```

April 05

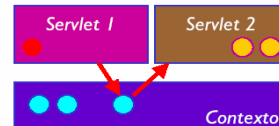
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

56

Gravação de atributos no contexto

- Servlets podem compartilhar objetos pelo contexto usando

- `setAttribute("nome", objeto);`
- Object `getAttribute("nome");`



- Exemplo de uso

Servlet 1

```
String[] vetor = {"um", "dois", "tres"};
ServletContext ctx = getServletContext();
ctx.setAttribute("dados", vetor);
```

Servlet 2

```
ServletContext ctx = getServletContext();
String[] dados = (String[])ctx.getAttribute("dados");
```

- Outros métodos

- `removeAttribute(String nome)` - remove um atributo
- Enumeration `getAttributeNames()` - lê nomes de atributos

Escopo e Threads

- Geralmente, só há **uma instância** de um servlet rodando para vários clientes
 - Atributos de instância são compartilhados!
- Se não desejar compartilhar dados entre clientes, use sempre objetos **thread-safe**
 - Atributos guardados no **request**
 - Variáveis **locais**
- Quaisquer outros atributos, como atributos de sessão, atributos de instância e de contexto são compartilhados entre requisições
 - Caso deseje compartilhá-los, use **synchronized** nos blocos de código onde seus valores são alterados.

Repasse de requisição

- Objetos *RequestDispatcher* servem para repassar requisições para outra página ou servlet. Seus dois principais métodos são
 - `include(request, response)`
 - `forward(request, response)`
- Esses métodos não podem definir cabeçalhos
 - `forward()` repassa a requisição para um recurso
 - `include()` inclui a saída e processamento de um recurso no servlet
- Para obter um *RequestDispatcher* use o *ServletRequest*

```
RequestDispatcher dispatcher =
    request.getRequestDispatcher("url");
```
- Para repassar a requisição para outra máquina use

```
dispatcher.forward(request, response);
```
- No repasse de requisição, o controle não volta para o browser.
 - Todos os parâmetros e atributos da requisição são preservados

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

59

Redirecionamento x Repasse

- Pode-se enviar um cabeçalho de redirecionamento para o browser usando

```
response.sendRedirect("url");
```
- Isto é o mesmo que fazer

```
response.setHeader("Location", "url");
```
- *Location* é um cabeçalho HTTP que instrui o browser para redirecionar para outro lugar
- Sempre que o controle volta ao browser, a primeira requisição terminou e outra foi iniciada
 - Os objetos *HttpServletResponse* e *HttpServletRequest* e todos seus atributos e parâmetros foram destruídos
- Com repasse de requisições, usando *RequestDispatcher*, o controle não volta ao browser mas continua em outro servlet (com `forward()`) ou no mesmo servlet (com `include()`)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

60

POO-Java

Gerenciamento
Sessão



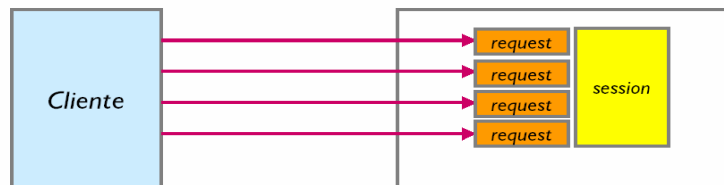
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

61

Sessões

- Como o HTTP não mantém **estado de sessão**, são as aplicações Web que precisam cuidar de mantê-lo quando necessário
- Sessões representam um cliente
 - A sessão é única para cada cliente e persiste através de várias requisições



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

62

Sessões

- Sessões são representados por objetos `HttpSession` e são obtidas a partir de uma requisição
- Dois métodos podem ser usados

```
HttpSession session = request.getSession(false);
```

 - Se a sessão não existir, retorna null, caso contrário retorna sessão.

```
HttpSession session = request.getSession();
```

 - Retorna a sessão ou cria uma nova. Mesmo que `getSession(true)`
- Para saber se uma sessão é nova, use o método `isNew()`

```
if (session.isNew()) {  
    myObject = new BusinessObject();  
} else {  
    myObject = (BusinessObject) session.getAttribute("obj");  
}
```
- `getSession()` deve ser chamado antes de `getOutputStream()`*
 - Sessões podem ser implementadas com cookies, e cookies são definidos no cabeçalho HTTP (que é montado antes do texto)

*ou qualquer método que obtenha o stream de saída, como `getWriter()`

Compartilhamento objetos na sessão

- Dois métodos

```
setAttribute("nome", objeto);  
Object getAttribute("nome");
```

permitem o compartilhamento de objetos na sessão. Ex:

Requisição 1

```
String[] vetor = {"um", "dois", "tres"};  
HttpSession session = request.getSession();  
session.setAttribute("dados", vetor);
```


Requisição 2

```
HttpSession session = request.getSession();  
String[] dados = (String[])session.getAttribute("dados");
```
- Como a sessão pode persistir além do tempo de uma requisição, é possível que a persistência de alguns objetos não sejam desejáveis
 - Use `removeAttribute("nome")` para remover objetos da sessão

Sessão

- A sessão é implementada com cookies se o cliente suportá-los
 - Caso o cliente não suporte cookies, o servidor precisa usar outro meio de manter a sessão

- Solução: sempre que uma página contiver uma URL para outra página da aplicação, a URL deve estar dentro do método `encodeURL()` de `HttpServletResponse`

```
out.print("<a href=' " +  
        response.encodeURL("caixa.jsp") + "'>");
```

- Se cliente suportar cookies, URL passa inalterada (o identificador da sessão será guardado em um cookie)
- Se cliente não suportar cookies, o identificador será passado como parâmetro da requisição

ex: `http://localhost:8080/servlet/Teste;jsessionid=A424JX08S99`

Escopo de objetos em servlets

- Servlets podem compartilhar informações de várias maneiras
 - Usando meios persistentes (bancos de dados, arquivos, etc)
 - Usando objetos na memória por escopo (requisição, sessão, contexto)
 - Usando variáveis estáticas ou de instância

- Servlets oferecem três níveis diferentes de persistência na memória (ordem decrescente de duração)

- **Contexto da aplicação:** vale enquanto aplicação estiver na memória (`javax.servlet.ServletContext`)

- **Sessão:** dura uma sessão do cliente (`javax.servlet.http.HttpSession`)

- **Requisição:** dura uma requisição (`javax.servlet.ServletRequest`)

- Para gravar dados em um objeto de persistência na memória

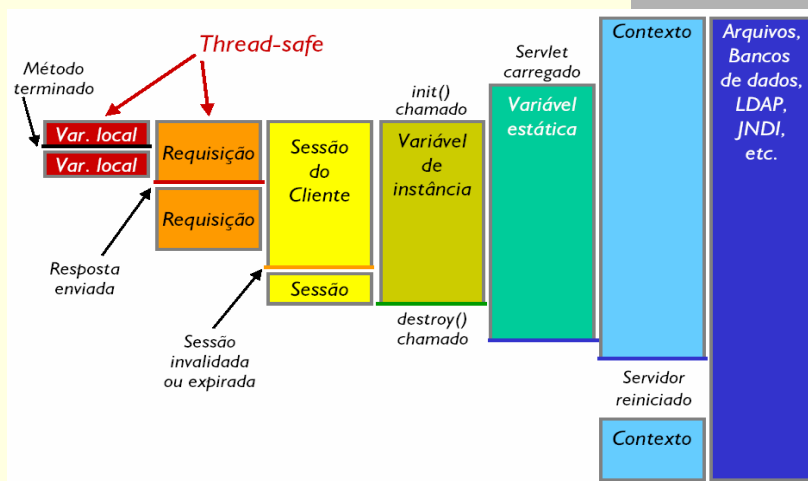
```
objeto.setAttribute("nome", dados);
```

- Para recuperar ou remover os dados

```
Object dados = objeto.getAttribute("nome");
```

```
objeto.removeAttribute("nome");
```

Escopo de objetos em servlets: resumo



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

67

Recursos compartilhados

- Há vários cenários de acesso concorrente
 - Componentes compartilhando sessão ou contexto
 - Threads acessando variáveis compartilhadas
- Servlets são automaticamente multithreaded
 - O container cria **um thread na instância para cada requisição**
 - É preciso **sincronizar blocos críticos** para evitar problemas decorrentes do acesso paralelo
- Exemplo: protegendo definição de atributo de contexto:

```
synchronized(this) {  
    context.setAttribute("nome", objeto);  
}
```
- Para situações onde multithreading é inaceitável, servlet deve implementar a interface **SingleThreadModel** (só um thread estará presente no método `service()` ao mesmo tempo)
 - Evite isto a todo custo: muito ineficiente!

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

68

Acesso a Bancos de Dados

- *Servlets são aplicações Java e, como qualquer outra aplicação Java, podem usar JDBC e integrar-se com um banco de dados relacional*

- *Pode-se usar `java.sql.DriverManager` e obter a conexão da forma tradicional*

```
Class.forName("nome.do.Driver");
Connection con =
    DriverManager.getConnection("url", "nm", "ps");
```

- *Pode-se obter as conexões de um pool de conexões através de `javax.sql.DataSource` via JNDI (use esta solução em servidores J2EE!)*

```
DataSource ds = (DataSource)ctx.lookup("jdbc/Banco");
Connection con = ds.getConnection();
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

69

Acesso ao pool de conexões

- *Fábricas de objetos são acessíveis via `<resource-ref>`. A mais comum é a fábrica de conexões de banco de dados*

```
<resource-ref>
  <description>Cloudscape database</description>
  <res-ref-name>jdbc/BankDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>SERVLET</res-auth>
</resource-ref>
```

- *`<res-auth>` informa quem é responsável pela autenticação*

- *Através da `DataSource`, obtém-se uma conexão.*

```
InitialContext initCtx = new InitialContext();
DataSource ds = (DataSource)
    initCtx.lookup("java:comp/env/jdbc/BankDB");
Connection con1 = ds.getConnection();// res-auth: CONTAINER
Connection con2 =
    source.getConnection("user", "pass");// res-auth: SERVLET
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

70

POO-Java

Configurando
o Tomcat



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

71

Contêineres WEB

- Os scriptlets contidos nas páginas JSP são processados pelo **Contêiner JSP**;
 - Ao browser, chega apenas a página HTML resultante do processamento do arquivo .jsp. Ou seja scriptlets (<% ... %>) nunca chegam ao browser.
- Para executar servlets e arquivos JSP, é preciso implantá-los em um **Contêiner Web**.
- Um Contêiner Web pode estar executando como parte de um servidor HTTP ou pode ele próprio ser o servidor.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

72

Contêineres WEB

- O Contêiner Web é responsável por:
 - transformar arquivos JSP em arquivos fonte em Java (mais especificamente, em fontes servlets)
 - compilar esses arquivos fonte, criando arquivos `.class`
 - repassar esses arquivos `.class` à JVM para para execução.
- Existem diversos Contêineres WEB atualmente...

Contêineres WEB (cont.)

- Apache Tomcat
 - <http://jakarta.apache.org/tomcat/>
- Sun JSWDK
 - <http://java.sun.com/products/servlet/download.html>
- Allaire JRun
 - <http://www.allaire.com/products/jrun/>
- New Atlanta ServletExec
 - <http://newatlanta.com/>
- Gefion Software LiteWebServer
 - <http://www.gefionsoftware.com/LiteWebServer/>
- Sun Java Web Server™
 - <http://www.sun.com/software/jwebserver/try/>

Contêineres WEB (cont.)

- **Bluestone**
 - <http://www.bluestone.com>
- **Borland Enterprise Server**
 - <http://www.inprise.com>
- **iPlanet Application Server**
 - <http://www.iplanet.com>
- **Orbix E2A (formally iPortal)**
 - <http://www.iona.com>
- **Jetty**
 - <http://www.mortbay.com>
- **JRun**
 - <http://www.allaire.com>

April 05

Prof. Ismael H. F. Santos - ismael@tegraf.puc-rio.br

75

Contêineres WEB (cont.)

- **Orion Application Server**
 - <http://www.orionserver.com>
- **Resin**
 - <http://www.caucho.com>
- **SilverStream**
 - <http://www.silverstream.com>
- **Weblogic Application Server**
 - <http://www.bea.com>
- **WebSphere**
 - <http://www-4.ibm.com/sfotware/webserver/appserv>

April 05

Prof. Ismael H. F. Santos - ismael@tegraf.puc-rio.br

76

Tomcat

- É um servidor Web (open source) do projeto Apache.
- Atualmente, na versão 5.0.x
- O dois principais módulos do Tomcat são
 - **Catalina**, o contêiner WEB do Tomcat.
 - **Jasper**, o compilador de páginas JSP
 - **Conectores**. O conector padrão é o HTTP.
- Por default, o Tomcat ocupará a porta 8080 da máquina onde estiver executando.
- Pasta de instalação default (CATALINA_HOME)
 - C:\Program Files\Apache Software Foundation\Tomcat 5.0

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

77

Estrutura de pastas do Tomcat 5.0

- **work**: onde são gerados os .java das servlets a partir de JSPs
- **bin**: executáveis do Tomcat
- **commons**: classes utilizadas pelo Tomcat que também estão disponíveis para as aplicações implantadas.
- **webapps**: pastas de contextos (para conter as aplicações Web)
- **shared**: contém classes que visíveis para todas as aplicações (e.g., driver JDBC)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

78

Estrutura de pastas do Tomcat 5.0

- **server**: classes que formam o servidor Tomcat e o seu contêiner WEB.
- **conf**: arquivos de configuração (server.xml e outros)
- **logs**: arquivos de log para as aplicações. Refira-se a esses arquivos para depurar suas aplicações.
- **temp**: diretório temporário utilizado internamente pelo Tomcat

Operação com o Tomcat

- Derrubando e levantando: para levantar ou derrubar o Tomcat, utilize a ferramenta **Tomcat Monitor**.
 - Alternativamente, pode-se fazer isso com os scripts localizados na pasta bin: *startup.bat* e *shutdown.bat*.
- Abra um navegador WEB e acesse a página cujo endereço é <http://localhost:8080>
- O resultado é a página principal do Tomcat
 - Com links para documentação e para administração do servidor
 - Tomcat Manager (gerenciamento das aplicações WEB)
 - Tomcat Administration (configuração do servidor WEB)
 - Exemplos de JSPs e de servlets

Contextos

- Um contexto é um diretório que deve ser criado pelo programador para que o Tomcat reconheça os arquivos e recursos de uma aplicação Web.
- Esses diretórios armazenam os recursos (arquivos) de uma aplicação WEB.
- No Tomcat, há três maneiras de criar o contexto de uma aplicação WEB:
 1. Transferir os arquivos da aplicação (JSP, servlets, imagens, etc.) para **contextos** predefinidos pelo servidor.
 2. Configurar o servidor para que reconheça um novo contexto onde os arquivos da aplicação residem (**server.xml**)
 3. Implantar a aplicação como um WebArchive (**WAR**)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

81

Implantação de aplicações no Tomcat

- O contexto raiz chama-se **ROOT**.
 - Arquivos copiados para **<CATALINA_HOME>\webapps\ROOT** podem ser acessados via **http://servidor:8080/**
 - Servlets em **<CATALINA_HOME>\webapps\ROOT\WEB-INF\classes** podem ser acessados via **http://servidor:8080/servlet/**

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

82

Implantação de aplicações no Tomcat

- Para informar o Tomcat de um contexto externo à raiz, o arquivo **server.xml** deve ser editado.
 - Esse arquivo é armazenado em **<CATALINA_HOME>\conf**
 - Exemplo: contexto *expljsp*.
- Para aplicações complexas, crie um ou mais arquivos **WAR (Web ARchive)**
 - Use a ferramenta jar para empacotar os arquivos da aplicação.
 - Renomeie o arquivo com a extensão war.
 - Copie esse arquivo para a pasta raiz do contêiner.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

83

Construção de uma servlet

- **Passos para construir uma servlet**
 1. Crie uma estrutura de diretórios para a sua aplicação.
 - Pastas **WEB-INF** e **classes**
 - Nota: o conteúdo de WEB-INF não é visível a partir do browser.
 2. Escreva o código fonte da servlet.
 - É necessário importar os pacotes **javax.servlet** e **javax.servlet.http**.
 3. Compile o código fonte.
 - O arquivo **servlet-api.jar** deve estar no classpath
 - Localizado em **<CATALINA_HOME>\commonlib\servlet-api.jar**

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

84

Construção de uma servlet

- **Passos para construir uma servlet (cont.)**
 4. Crie um descritor de implantação (deployment descriptor).
 - **Passo opcional.**
 - **Um descritor de implantação é um arquivo XML (web.xml)**
 5. Execute o Tomcat.
 6. Chame a servlet a partir de um navegador web.

Jakarta Tomcat - Configuração

- **Registrando Servlets**
 - **O arquivo web.xml**
 - **Localizado em geral no diretório:**
 - %CATALINA_HOME%\webapps**<Aplicacao>**WEB-INF
 - **Registrando um servlet:**

```
<web-app>
  <servlet>
    <servlet-name>ServletSes</servlet-name>
    <servlet-class>interfPesquisa.ServletSes</servlet-class>
  </servlet>
</web-app>
```

Jakarta Tomcat - Configuração

- **Parametros de inicialização**
 - Muitas vezes existem valores que são constantes durante a execução do servlet, mas que podem mudar durante a vida da aplicação:
 - Exemplo:
 - **Localização de recursos, Mensagens padrão**
- **Principais métodos**
 - Classes Servlet e ServletConfig
 - **public java.util Enumeration getInitParameterNames()**
 - **public java.lang.String getInitParameter(java.lang.String name)**

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

87

Parâmetros de Inicialização

- **Configurando no Tomcat**
 - **Arquivo web.xml**

```
<servlet-class>interfPesquisa.ServletSes</servlet-class>
+ <init-param>
+ <init-param>
+ <init-param>
+ <init-param>
+ <init-param>
+ <init-param>
+ <init-param>
- <init-param>
  <param-name>arqXSLCSV</param-name>
  <param-value>respostaTOCSV.xml</param-value>
</init-param>
- <init-param>
  <param-name>arqXSLPrint</param-name>
  <param-value>respostaTOprint.xml</param-value>
</init-param>
- <init-param>
  <param-name>arqXSLAutorizacao</param-name>
  <param-value>respostaTOAutorizacao.xml</param-value>
</init-param>
- <init-param>
  <param-name>dtdPesquisa</param-name>
  <param-value>http://139.82.24.86/ses/dtd/Pesquisa.dtd</param-value>
</init-param>
</servlet>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

88

Jakarta Tomcat - Configuração

init(ServletConfig config)

```
public void init( ServletConfig config ) throws ServletException
{
    super.init( config );
    System.out.println( "Inicializando ServletSes..." );
    SingletonProperties properties = SingletonProperties.getInstance();
    Enumeration parameters = getInitParameterNames();

    while ( parameters.hasMoreElements() )
    {
        String paramName = ( String ) parameters.nextElement();
        String paramValue = getInitParameter( paramName );
        System.out.println( "InitParameter - Nome: " + paramName + " Valor " +
            paramValue );
        properties.setProperty( paramName , paramValue );
    }

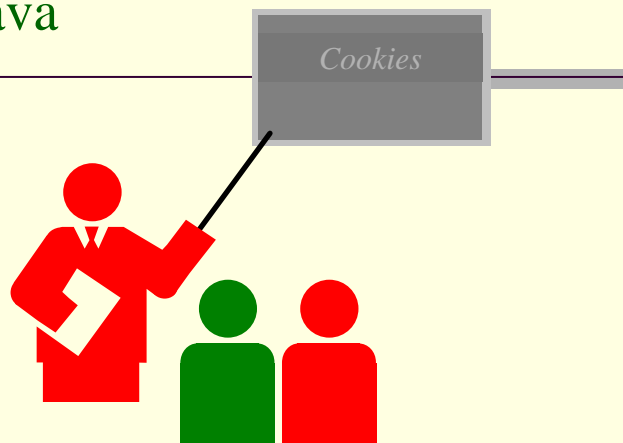
    System.out.println( properties );
    facade = new FacadeInterf();
    ARQ_DESIGN = properties.getProperty( "arqDesign" );
    DIR_ARQS = properties.getProperty( "dirHTML" );
}
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

89

POO-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

90

Controle de sessão

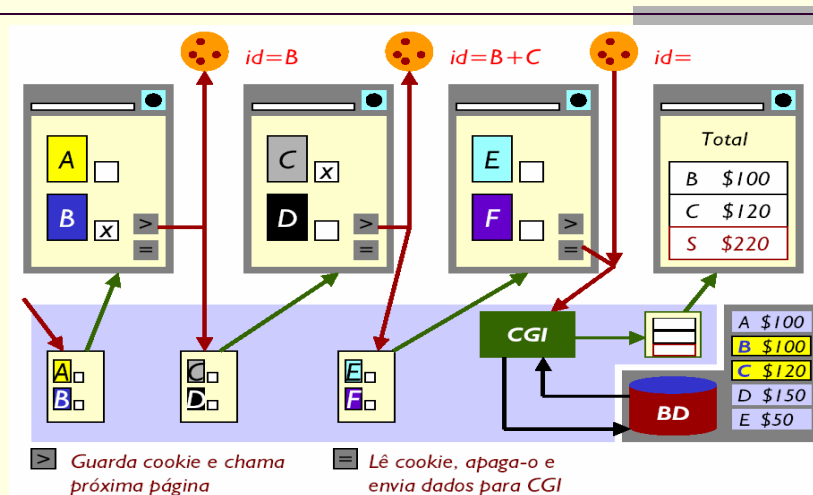
- *HTTP não preserva o estado de uma sessão. É preciso usar mecanismos artificiais com CGI (ou qualquer outra tecnologia Web)*
 - **Seqüência de páginas/aplicações:** desvantagens: seqüência não pode ser quebrada; mesmo que página só contenha HTML simples, precisará ser gerada por aplicação
 - **Inclusão de dados na URL:** desvantagens: pouca flexibilidade e exposição de informações
 - **Cookies** (informação armazenada no cliente): desvantagens: espaço e quantidade de dados reduzidos; browser precisa suportar a tecnologia

Cookies

- *Padrão Internet (RFC) para persistência de informações entre requisições HTTP*
- *Um cookie é uma pequena quantidade de informação que o servidor armazena no cliente*
 - Par **nome=valor**. Exemplos: `usuario=paulo`, `num=123`
 - Escopo no servidor: **domínio** e **caminho** da página
 - Pode ser **seguro**
 - Escopo no cliente: browser (sessão)
 - Duração: uma sessão ou tempo determinado (cookies persistentes)
- *Cookies são criados através de cabeçalhos HTTP*

```
Content-type: text/html
Content-length: 34432
Set-Cookie: usuario=ax343
Set-Cookie: lastlogin=12%2610%2699
```

Exemplo com cookies: Loja Virtual



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

93

Cookies

- Um cookie corresponde a uma informação que o servidor requisita que seja armazenada no cliente.
 - Seguem sempre o formato: **nome=valor**
 - Exemplos: usuario=bezerra, id=1234
- Cookies são criados através de cabeçalhos HTTP, pelo uso da diretiva **Set-Cookie**.
- Exemplo:
Content-type: text/html
Content-length: 43894
Set-Cookie: usuario=bezerra
Set-Cookie: ultimologin=13%09%2004

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

94

Cookies (cont.)

- Usos típicos de cookies:

- Identificar um usuário durante uma sessão de e-commerce.
- Evitar utilização de nome de usuário e senha
- Customização de sites
- Propaganda direcionada

- Enviando um cookie para o browser em JSP:

```
<%  
Cookie c = new Cookie("usuario", "bezerra");  
c.setMaxAge(2592000); // Seconds  
response.addCookie(c);  
%>
```

Cookies (cont.)

- Enviando um cookie para o browser em uma servlet:

```
...  
Cookie c = new Cookie("name", "value");  
c.setMaxAge(...);  
response.addCookie(c);  
...
```


Cookies (cont.)

- Lendo cookies do browser em uma servlet:

```
...  
Cookie[] cookies =  
    response.getCookies();  
for(int i=0; i<cookies.length; i++) {  
    Cookie c = cookies[i];  
    if (c.getName().equals("someName")) {  
        doSomethingWith(c);  
        break;  
    }  
}  
...  
}
```