



# Servlets

*Helder da Rocha (helder@acm.org)*  
*[www.argonavis.com.br](http://www.argonavis.com.br)*

# Sobre este módulo

- *Neste módulo serão apresentados os fundamentos de servlets*
  - *Como escrever um servlet*
  - *Como compilar um servlet*
  - *Como implantar um servlet no servidor*
  - *Como executar*
- *Também serão exploradas as formas de interação do servlet com a requisição e resposta HTTP*
  - *Como ler parâmetros da entrada*
  - *Como gerar uma página de resposta*
  - *Como extrair dados de um formulário HTML*

# O que são servlets

- *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.*

## Principais classes e interfaces de *javax.servlet*

### ■ Interfaces

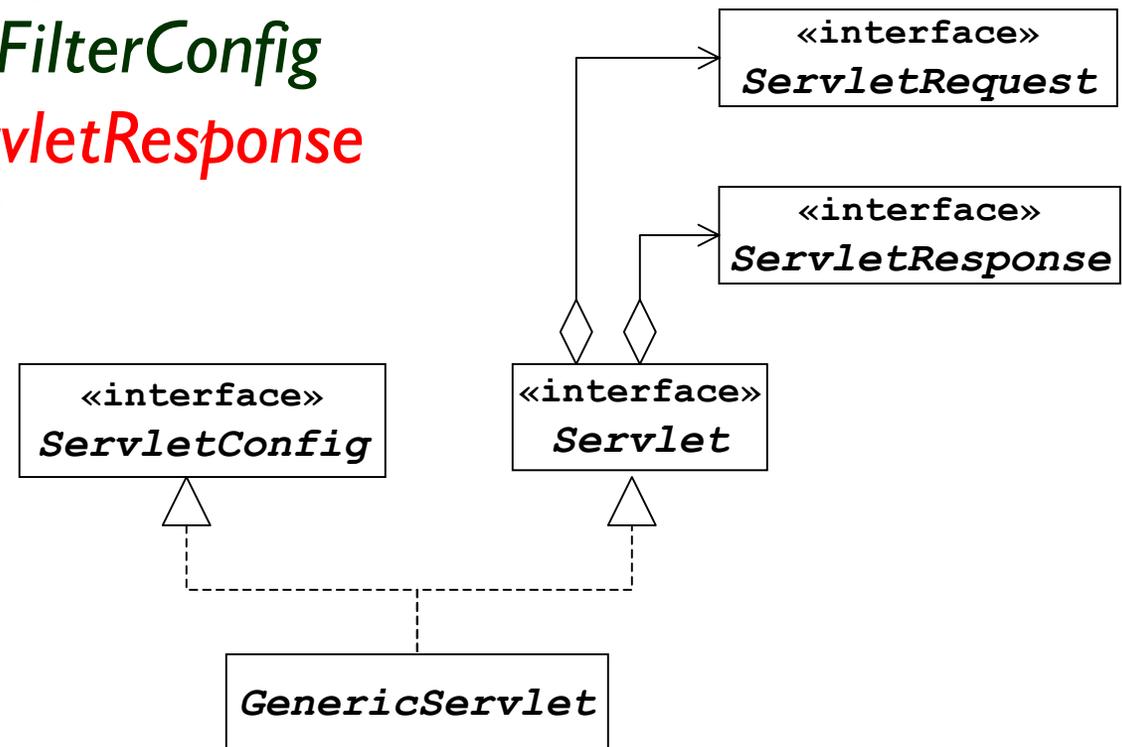
- *Servlet*, *ServletConfig*, *ServletContext*
- *Filter*, *FilterChain*, *FilterConfig*
- *ServletRequest*, *ServletResponse*
- *SingleThreadModel*
- *RequestDispatcher*

### ■ Classes abstratas

- *GenericServlet*

### ■ Classes concretas

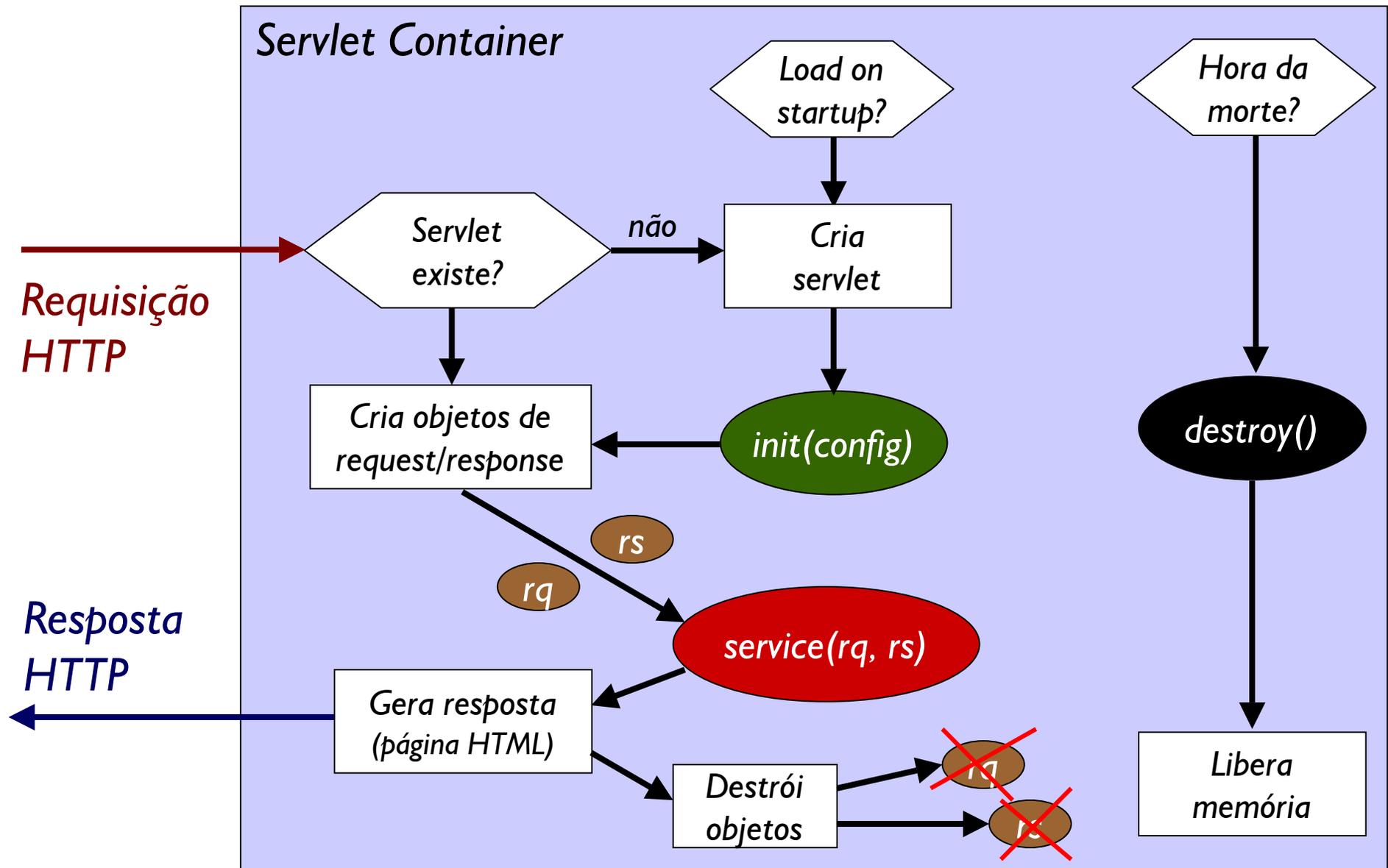
- *ServletException*
- *UnavailableException*
- *ServletInputStream* e *ServletOutputStream*



# 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()**

# Ciclo de vida



# Como escrever um Servlet genérico

- Um servlet genérico deve estender **GenericServlet** e implementar seu método **service()**

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

public class Generico extends GenericServlet {

    public void service (ServletRequest request,
                        ServletResponse response)
                        throws IOException {

        PrintWriter out = response.getWriter();
        out.println("Hello, World!");
        out.close();
    }
}
```

# Inicialização de um servlet

- Tarefa realizada uma vez
- Deve-se sobrepor *init(config)* com instruções que serão realizadas para inicializar um servlet
  - Carregar parâmetros de inicialização, dados de configuração
  - Obter outros recursos
- Falha na inicialização deve provocar *UnavailableException* (subclasse de *ServletException*)

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

- Quando um servlet container decide remover um servlet da memória, ele chama o seu método **destroy()**
  - Use `destroy()` para liberar recursos (como conexões de banco de dados, por exemplo) e fazer outras tarefas de "limpeza".

```
public void destroy() {  
    banco.close();  
    banco = null;  
}
```

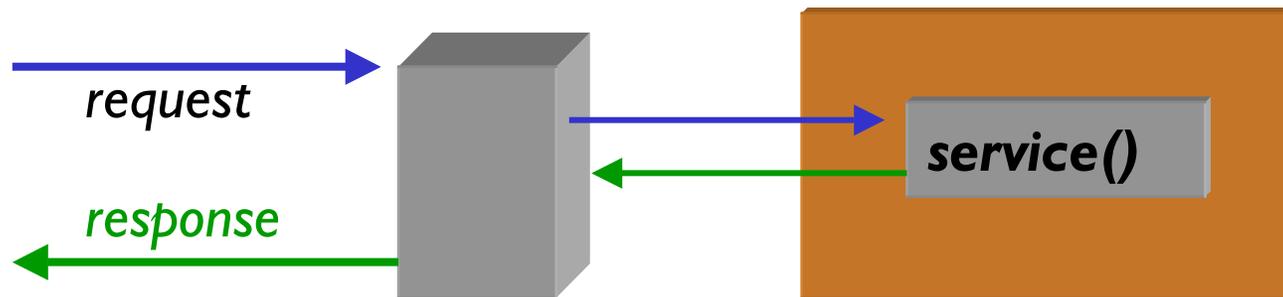
- O servlet geralmente só é destruído quando todos os seus métodos **service()** terminaram (ou depois de um timeout)
  - Se sua aplicação tem métodos `service()` que demoram para terminar, você deve garantir um shutdown limpo.

# 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)

# 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

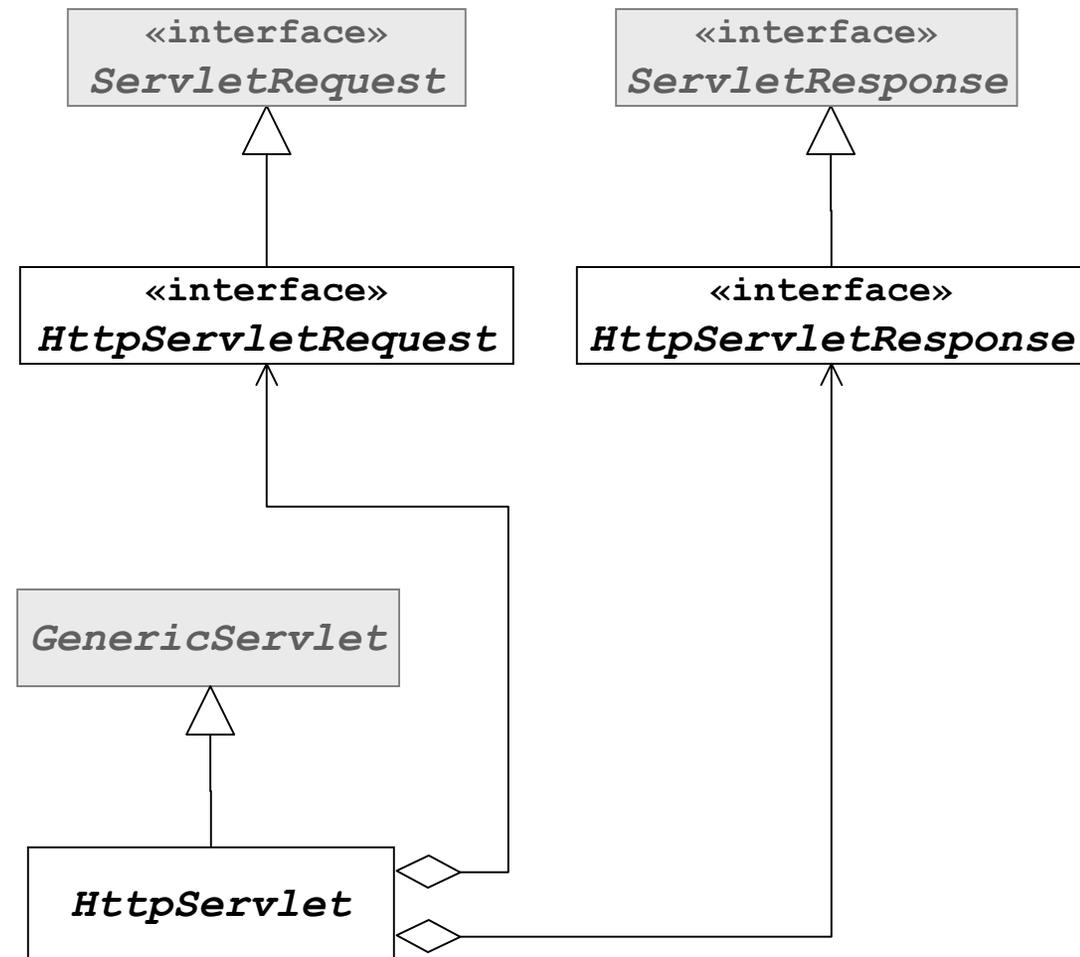
# Servlets genéricos

- *Servlets genéricos servem como componentes para serviços tipo requisição-resposta em geral*
  - *Não se limitam a serviços HTTP*
  - *Podem ser usados para estender, com componentes reutilizáveis, um serviço existente: é preciso implementar um "container" para rodar o servlet*
- *Para serviços Web deve-se usar **Servlets HTTP***
  - *API criada especificamente para lidar com características próprias do HTTP*
  - *Método **service()** dividido em métodos específicos para tratar os diferentes métodos do HTTP*

# API: Servlets HTTP

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

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



# Como escrever um servlet HTTP

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

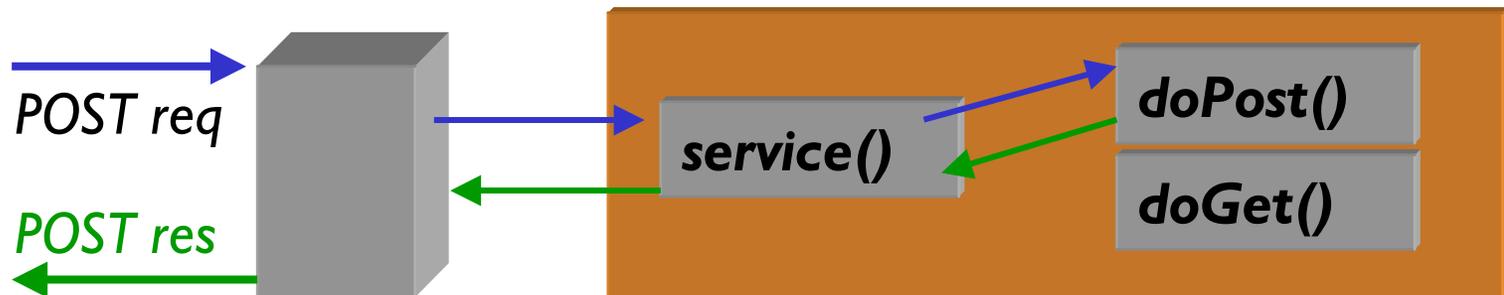
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

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();
    }
}
```

# 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 **pelo menos um** dos métodos `doGet()` ou `doPost()`

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

- A inicialização de um `GenericServlet`, como o `HttpServlet`, pode (e deve) ser feita com a versão de `init()` sem argumentos (e não `init(config)`)
- 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!");  
    }  
}
```

# A 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

\* Especificação de cabeçalho para e-mail

# Obtenção de dados de requisições

- 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)

# A 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 de uma 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

# Como implementar 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
```

# Como ler 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");
```

# Como gerar uma 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>");
```

# Criação de servlets simples

- São necessárias quatro etapas para construir e usar um servlet
  - **Codificar** o servlet, usando a Servlet API
  - **Compilar** o servlet, usando o JAR que contém as classes da API (distribuído pelo software do Web Container)
  - **Implantar** o servlet no servidor (Web Container)
  - **Executar** o servlet, chamando-o pelo browser
- Code - Compile - Deploy - Run

# Compilação e Implantação

- Para **compilar**, use qualquer distribuição da API
  - O **servlet.jar** distribuído pelo Tomcat em **common/lib/**
  - O **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/**)

- *Pode-se usar o Ant para fazer a compilação e deployment de uma aplicação Web*
  - *Defina um `<classpath>` adicional nas tarefas `<javac>` que inclua o caminho do `servlet.jar`*
  - *Use `<copy>` para copiar os arquivos para os contextos corretos*
  - *Use `<property environment="env" />` e as propriedades `env.TOMCAT_HOME` ou `env.CATALINA_HOME` para ler as variáveis de ambiente do sistema e tornar seu build file independente da localização do servidor*
- *O Ant também pode ser usado para gerar o JAR que encapsula uma aplicação Web (WAR) usando a tarefa `<war>`*

- Se você instalou os servlets em um contexto raiz, execute-os através da URL
  - `http://localhost:8080/servlet/nome.do.Servlet`
- Se você instalou os servlets em outro contexto use
  - `http://localhost:8080/contexto/servlet/nome.do.Servlet`
- Para passar parâmetros
  - Escreva um formulário HTML, ou
  - Passe-os via URL, acrescentando um ? seguido dos pares `nome=valor`:  
`http://localhost:8080/servlet/Servlet?id=3&nome=Ze`

- 1. Crie um servlet (*j550.cap02.ParameterList*) que imprima, em uma tabela, todos os nomes de parâmetros enviados e seus valores
  - O servlet deve suportar tanto GET como POST
  - Use `request.getParameterNames()` e `getParameterValues(String)`
  - Use o formulário de exemplo `AllForms.html` fornecido para testá-lo
- 2. Crie um servlet (*j550.cap02.HeaderList*) que imprima, em uma tabela, todos os nomes de cabeçalhos HTTP da requisição e seus valores
  - O servlet deve suportar tanto GET como POST
  - Use `request.getHeaderNames()` e `getHeaders(String)`

## Exercícios (2)

- 3. Escreva um servlet simples (*j550.cap02.HoraServlet*) que devolva uma página contendo o dia, mês, ano e hora
  - Escreva e compile o servlet
  - Copie-o para *\$TOMCAT\_HOME/webapps/ROOT/WEB-INF/classes*
  - Rode-o no browser: *http://localhost:8080/servlet/j550.HoraServlet*
- 4. Escreva um servlet (*j550.cap02.FatorialServlet*) que gere uma tabela HTML com a lista de fatoriais entre 0 e 10
- 5. Altere o exercício 2 para que o servlet verifique a existência de um parâmetro "maximo". Se ele não for null, converta-o em int e use-o como limite para gerar a tabela de fatoriais.
  - Passe parâmetro com *http://../j550.cap02.HoraServlet?maximo=5*
- 6. Escreva um servlet que imprima formatado em uma tabela HTML o conteúdo do arquivo *produtos.txt* (*j550.cap02.ProdutosServlet*)

# Formulários HTML

- *Todo formulário em HTML é construído usando elementos dentro de um bloco **<FORM>***
- *O bloco **<FORM>** define a URL que receberá o formulário e pode definir também o método usado*

```
<FORM ACTION="URL para onde serão enviado os dados"
  METHOD="método HTTP (pode ser GET ou POST)"
  ENCTYPE="formato de codificação"
  TARGET="nome da janela que mostrará a resposta" >
  ... corpo do formulário
      (permite qualquer coisa permitida em <BODY>)
  ...
</FORM>
```

# Formulários e links

- *Formulários são similares a links.*
- *Um par formulário-botão tem o mesmo efeito que um link criado com <A HREF>*
  - *O link está no formulário e o evento no botão*
- *O bloco*

```
<FORM ACTION="/dados/tutorial.html">  
  <INPUT TYPE="submit" VALUE="Tutorial">  
</FORM>
```
- *gera a mesma requisição que*

```
<A HREF="/dados/tutorial.html">Tutorial</A>
```
- *que é*

```
GET /dados/tutorial.html HTTP/1.0
```

# Envio de dados com Formulários

- Vários elementos *HTML* servem para entrada de dados e são usados dentro de formulários. Todos os elementos de entrada de dados têm um **nome** e enviam um **valor**
- Exemplo de formulário para entrada de dados



```
<FORM ACTION="/cgi-bin/catalogo.pl"
      METHOD="POST">
  <H3>Consulta preço de livro</H3>
  <P>ISBN: <INPUT TYPE="text" NAME="isbn">
  <INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

Cabeçalho HTTP

Linha em branco

Mensagem (corpo da requisição)

```
POST /cgi-bin/catalogo.pl HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 15

isbn=2877142566
```

# Elementos para disparo de eventos

- Os elementos `<INPUT>` com atributo `TYPE` **Submit**, **Reset** e **Button** servem para disparar eventos
  - Envio do formulário (Submit)
  - Reinicialização do formulário (Reset)
  - Evento programado por JavaScript (Button)
- O `value` do botão define o texto que mostrará
- Apenas se o botão contiver um atributo **name**, o conteúdo de **value** será enviado ao servidor

### Button

	<code>&lt;input type=button&gt;</code>
	<code>&lt;input type=button value="Apertar"&gt;</code>

### Submit

	<code>&lt;input type=submit&gt;</code>
	<code>&lt;input type=submit value="Enviar"&gt;</code>

### Reset

	<code>&lt;input type=reset&gt;</code>
	<code>&lt;input type=reset value="Limpar"&gt;</code>

# Entrada de texto

- Elementos `<INPUT>` com `TYPE="text"` podem ser usados para entrada de texto

```
<input type=text>
```

```
<input type=text size=10>
```

```
<input type=text value="texto inicial">
```

- Com `TYPE="password"` o texto digitado é ocultado na tela do browser

```
<input type=password maxlength=8>
```

```
<input type=password size=10>
```

# Campos ocultos

- *Campos ocultos consistem de um par nome/valor embutido no código HTML*
- *São úteis para que o autor da página possa enviar informações ao servidor*
  - *Informações sobre configuração da aplicação*
  - *Comandos, para selecionar comportamentos diferentes da aplicação*
  - *Parâmetros especiais para controle da aplicação, sessão ou dados que pertencem ao contexto da aplicação*
- *Sintaxe*
  - **<INPUT TYPE="hidden" NAME="nome" VALUE="valor">**

# Chaves booleanas

- Há dois tipos: checkboxes e radio buttons
- **Checkboxes** permitem mais de uma seleção

Café `<input type=checkbox name=refeicoes value="Café" checked>` **Café**  
 Almoço `<input type=checkbox name=refeicoes value="Almoço">` **Almoço**  
 Jantar `<input type=checkbox name=refeicoes value="Jantar">` **Jantar**

- O código acima enviará nomes repetidos contendo valores diferentes na requisição
- **Radio Buttons**, se tiverem o mesmo nome, formam um grupo. No grupo, apenas uma seleção é aceita

## GRUPO I

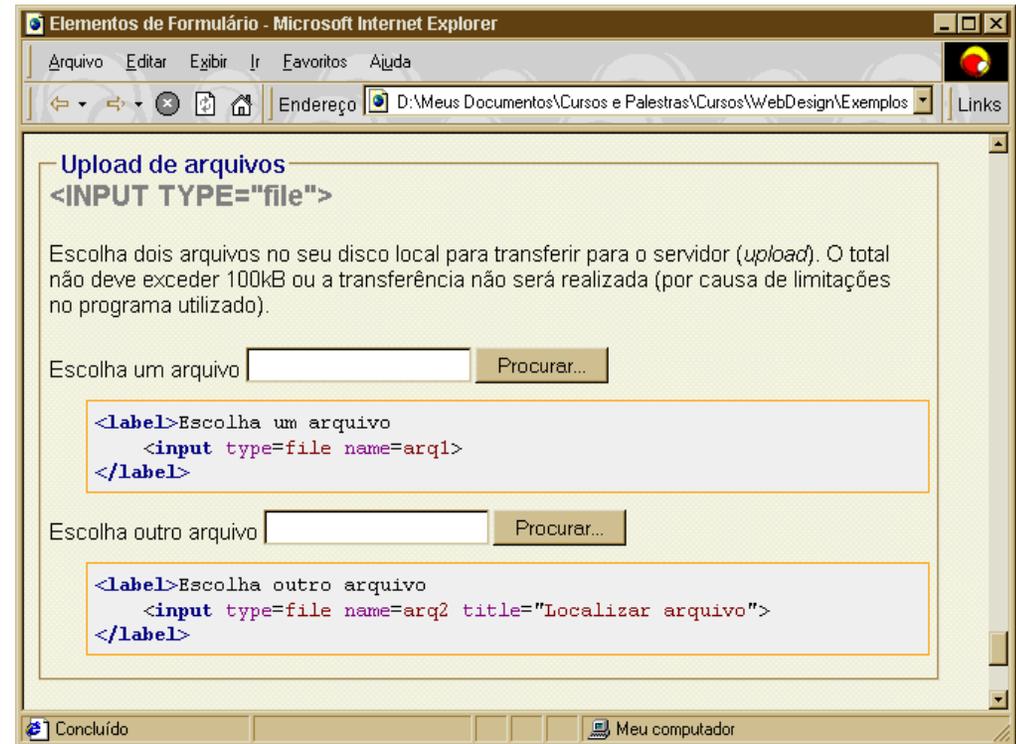
pela manhã `<input type=radio name=turno value="Manhã">` **pela manhã**  
 à tarde `<input type=radio name=turno value="Tarde">` **à tarde**  
 à noite `<input type=radio name=turno value="Noite" checked>` **à noite**

## GRUPO II

Masculino `<input type=radio name=sexo value="M">` **Masculino**  
 Feminino `<input type=radio name=sexo value="F">` **Feminino**

# Upload de arquivos

- O elemento `<INPUT TYPE="file">` cria uma tela que permite o Upload de arquivos para o servidor
- Formulário usado deve ter a seguinte sintaxe



```
<FORM ACTION="/servlet/UploadServlet"  
METHOD="POST"  
ENCTYPE="text/multipart-form-data"> ... </FORM>
```

# Área para entrada de texto

- Possibilitam a entrada de texto de múltiplas linhas
- Elemento: **<TEXTAREA>**

A screenshot of a web browser's text area. It contains three lines of text: "essa a página HTML.", "digitado pelo usuário", and "essa a página HTML.". The text area has a scroll bar on the right and a horizontal scroll bar at the bottom.

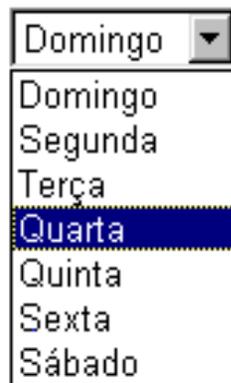
```
<textarea rows=3 cols=20></textarea>
```

A screenshot of a web browser's text area. It contains the text "Texto inicial". The text area has a scroll bar on the right and a horizontal scroll bar at the bottom.

```
<textarea rows=3 cols=20>Texto inicial</textarea>
```

# Menus de seleção

- Geram requisições similares a checkboxes e radio buttons
- Consistem de um par de elementos
  - **<SELECT>** define o nome da coleção
  - **<OPTION>** define o valor que será enviado



```
<select name=umdia>
  <option value="D">Domingo</option>
  <option value="S">Segunda</option>
  <option value="T">Terça</option>
  <option value="Q">Quarta</option>
  <option value="I">Quinta</option>
  <option value="X">Sexta</option>
  <option value="B">Sábado</option>
</select>
```



```
<select name=variosdias size=4 multiple>
  <option value="D">Domingo</option>
  <option value="S">Segunda</option>
  <option value="T">Terça</option>
  <option value="Q">Quarta</option>
  <option value="I">Quinta</option>
  <option value="X">Sexta</option>
  <option value="B">Sábado</option>
</select>
```

- 7. Crie um formulário onde uma pessoa possa digitar nome, telefone e e-mail. Faça-o chamar um servlet via POST que grave os dados em um arquivo.
  - a) Use o nome `j550.cap02.RegistroServlet`
  - b) Guarde um registro por linha
  - c) Separe cada campo com o caractere "|"
  - d) Use `RandomAccessFile`. Se o objeto se chamar, por exemplo, `raf`, use `raf.writeUTF()` para gravar no arquivo e `raf.seek(raf.length())` para achar o final do arquivo
  - e) Faça com que a resposta do servlet seja o arquivo formatado em uma tabela HTML
  - f) Se o servlet for chamado diretamente (via `GET`), deve também mostrar os dados do arquivo formatados
- 8. Embuta o formulário no próprio servlet, para que, após cada chamada, os dados e formulário sejam exibidos

*helder@acm.org*

***argonavis.com.br***