Prof. Fellipe Aleixo (fellipe.aleixo@ifrn.edu.br)

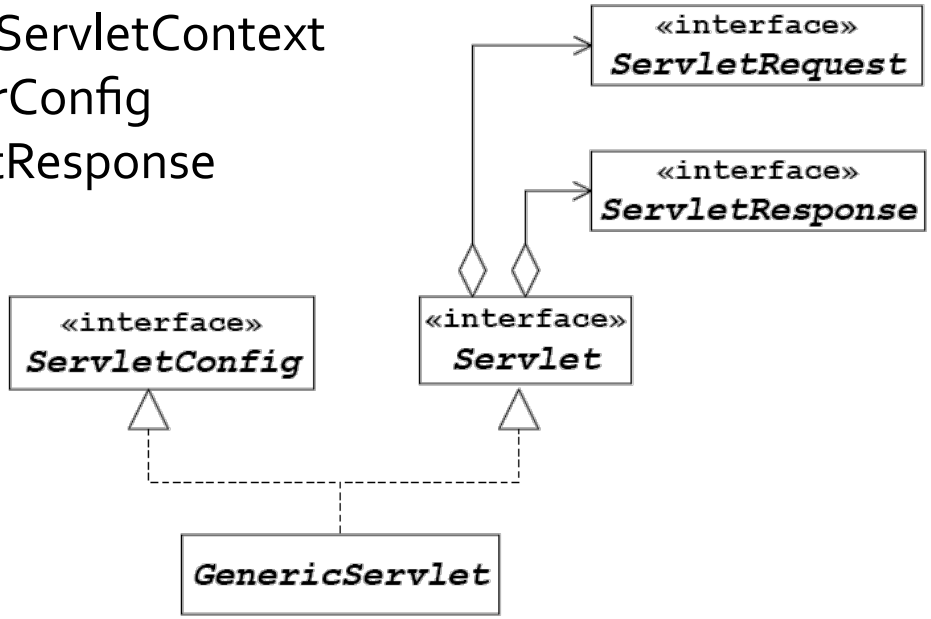
Servlets

O que são Servlets?

- Extensão de servidor escrita em Java
  - 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.

API: Fundamental

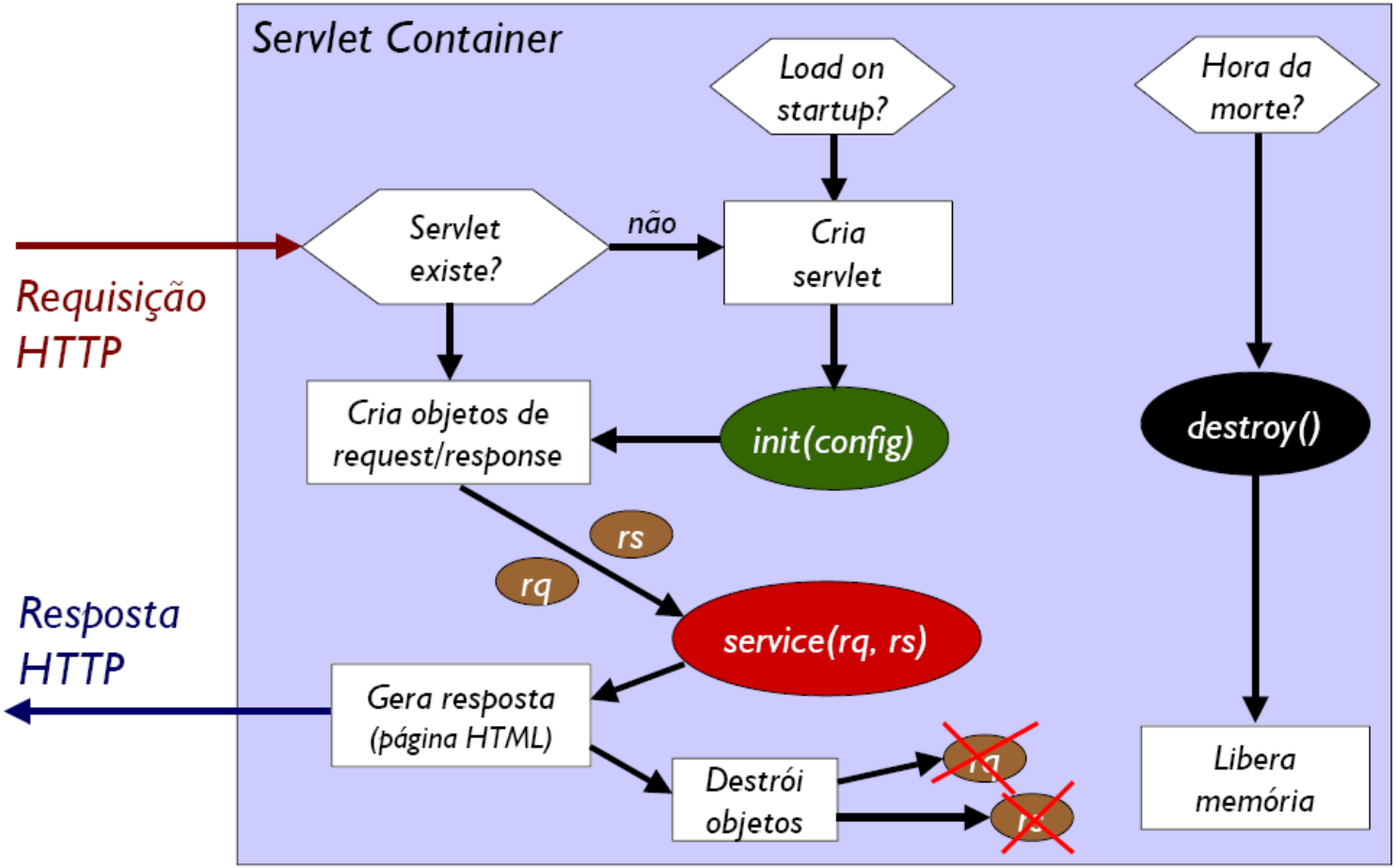
- Principais classes e interfaces de javax.servlet
  - Interfaces
    - Servlet, ServletConfig, ServletContext
    - Filter, FilterChain, FilterConfig
    - ServletRequest, ServletResponse
    - SingleThreadModel
    - RequestDispatcher
  - Classes abstratas
    - GenericServlet
  - Classes concretas
    - ServletException
    - UnavailableException
    - ServletInputStream e ServletOutputStream



Ciclo de Vida

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

Ciclo de Vida



[Argonavis]

Como Escrever um Servlet?

- Um servlet genérico deve estender GenericServlet e 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

- Inicialização: sobrescrever o método init(config) com
  - 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!");
    }
}
```

Finalização

- Quando um contêiner decide remover um servlet da memória, ele chama o seu método destroy()
  - Com o objetivo de liberar recursos (ex.: conexão de banco de dados)

```
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

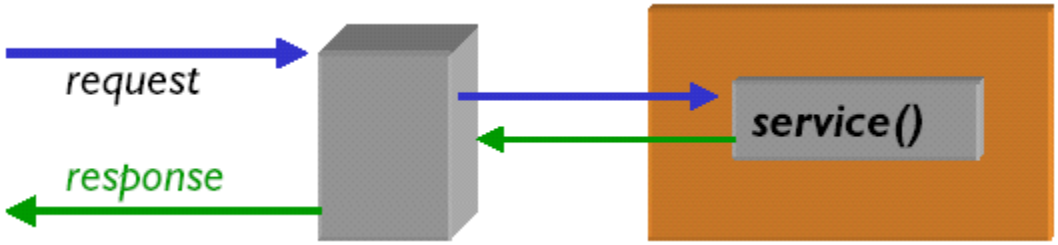
Métodos de Serviço

- Implementam operações de resposta executadas quando o cliente envia uma requisição
- Recebem dois parâmetros: um objeto ServletRequest e outro ServletResponse
- Tarefas usuais de um método de serviço
  - Extrair informações da requisição
  - Acessar recursos externos
  - Escrever a resposta (HTTP → (i) preencher os cabeçalhos de resposta, (ii) obter um stream de resposta e (iii) escrever os dados no stream

Métodos de Serviço

- O método de serviço de um servlet genérico é abstrato
  `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)



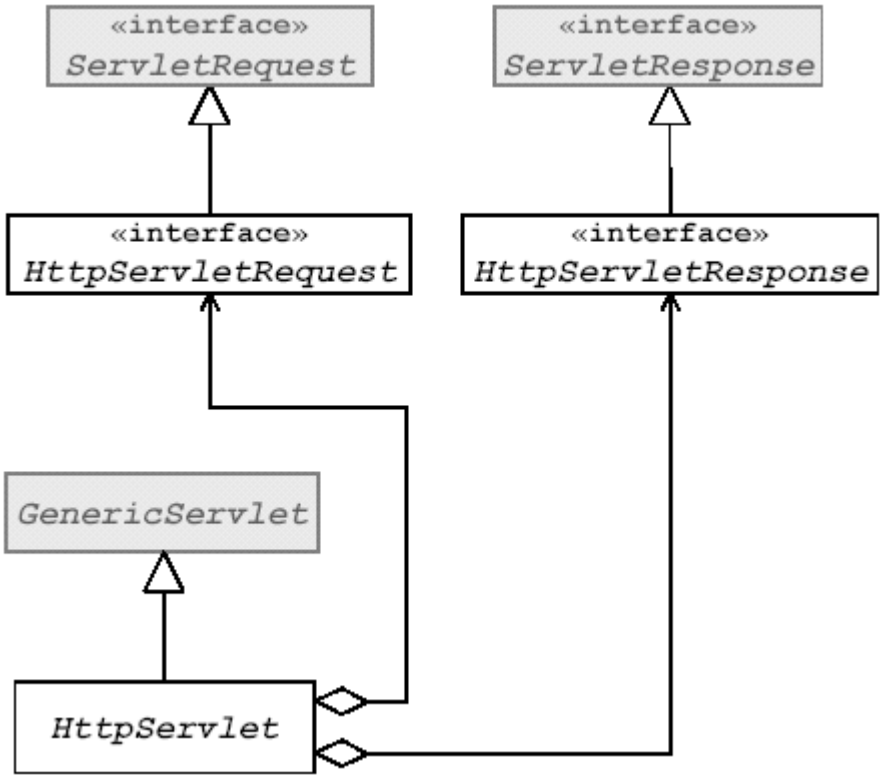
- Um servlet genérico deverá sobrepor este método e utilizar os objetos ServletRequest e ServletResponse

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 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

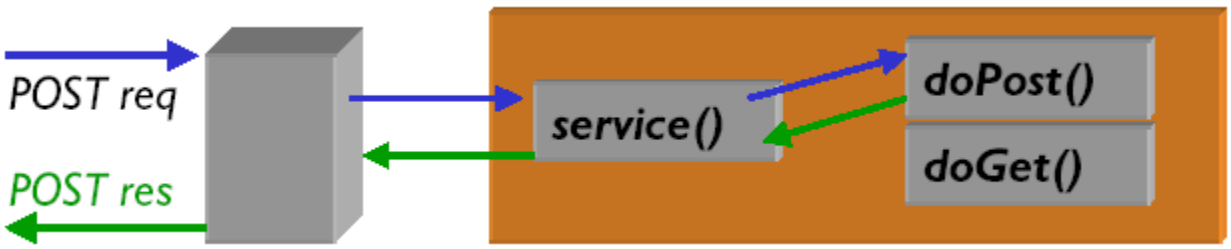
- Para escrever um servlet HTTP, deve-se estender HttpServlet e implementar um ou mais de seus métodos de serviço, tipicamente: doPost() e/ou doGet()

```
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

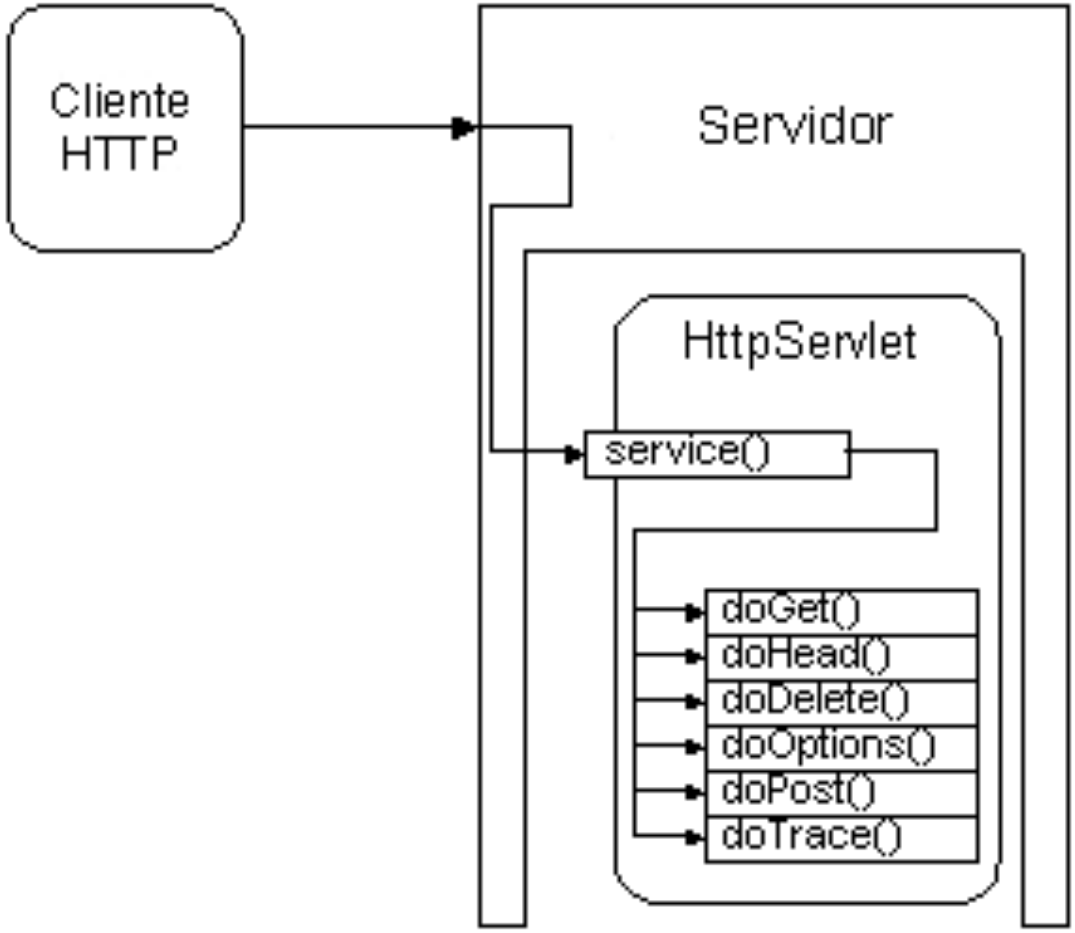
- A classe HttpServlet redireciona os pedidos encaminhados para service() para métodos que refletem os métodos HTTP (GET, POST, etc.):

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



- Um servlet HTTP deverá implementar pelo menos um dos métodos doGet() ou doPost()

Métodos de Serviço HTTP



Inicialização

- A inicialização de um HttpServlet, pode (e deve) ser feita com a versão de init() – sem argumentos

- 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!");
    }
}
```

Parâmetros de Inicialização

```
<web-app>
<servlet>
   <servlet-name>exemplo</servlet-name>
   <servlet-class>exemplo.PrimeiroServlet</servlet-class>
   <init-param>
      <param-name>JDBCDriver</param-name>
      <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
</web-app>
```

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

Obtenção de Dados de Requisições

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

A Resposta HTTP

- Uma resposta HTTP é enviada 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

A Resposta HTTP

- Alguns métodos de HttpServletResponse
  1. addHeader(String nome, String valor) - adiciona cabeçalho HTTP
  2. setContentType(tipo MIME) - define o tipo MIME que será usado para gerar a saída (text/html, image/gif, etc.)
  3. sendRedirect(String location) - envia informação de redirecionamento para o cliente (Location: url)
  4. Writer getWriter() - obtém um Writer para gerar a saída. Ideal para saída de texto.
  5. OutputStream getOutputStream() - obtém um OutputStream. Ideal para gerar formatos diferentes de texto (imagens, etc.)
  6. addCookie(Cookie c) - adiciona um novo cookie
  7. encodeURL(String url) - envia como anexo da URL a informação de identificador de sessão (sessionid)
  8. reset() - limpa toda a saída inclusive os cabeçalhos
  9. 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 &
- Duas formas para passar parâmetros:
  - 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

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

id=agente007&acesso=3
```

Lendo 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
- Seja o método POST ou GET, os valores dos parâmetros, podem ser recuperados pelo método getParameter()
  - 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");

Gerando uma Resposta

1. 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 - correspondem ao mesmo *stream*
2. Definir o tipo de dados a ser gerado (Content-type) para que o navegador saiba como exibir as informações
   - response.setContentType("text/html");
3. Depois, pode-se gerar os dados (HTML), imprimindo-os no objeto de saída (out) obtido anteriormente

Processando Formulários

```
<html>
  <head>Um Formulário Básico</head>
  <body>
      <h1>Entre com seus dados:</h1>
      <form action=”/processaForm" method="POST">
          Tratamento: <select size="1" name="titulo">
              <option>Sr.</option>
              <option>Sra.</option>
          </select>
          Nome: <input type="text" name="nome" size="20"><br>
          Cidade: <input type="text" name="cidade" size="20"><br>
          <p>Selecione seus interesses:</p>
          <input type="checkbox" name="interesses" value="esportes">Esportes<br>
          <input type="checkbox" name="interesses" value="musica">Musica<br>
          <input type="checkbox" name="interesses" value="leitura">Leitura<br>
          <p><input type="submit" value="Envie"></p>
      </form>
  </body>
</html>
```

Processando Formulários

```
public class ProcessaFormulario extends HttpServlet {
    private void processa(HttpServletRequest req, HttpServletResponse res) throws
    IOException {
          ServletOutputStream out = res.getOutputStream();
          res.setContentType("text/html");
          String tratamento = req.getParameter("titulo");
          String nome = req.getParameter("nome");
          String cidade = req.getParameter("cidade");
          String interesses[] = req.getParameterValues("interesses");
          out.println("<html><head><title>Resp do Servlet</title></head>");
          out.println("<body>");
          out.println("Olá " + tratamento + " " + nome);
          out.println("Você mora em " + cidade + " e seus interesses são: ");
          out.println("<ul>");
          for (String interesse : interesses) {
                  out.println("<li>");
                  out.println(interesse);
                  out.println("</li>");
        }
          out.println("</ul></body></html>");
    }
}
```

Redirecionamentos

- Ao receber uma requisição, um servlet pode:
  1. Respondê-la diretamente, produzindo algum conteúdo HTML a ser enviado ao cliente (navegador)
  2. Encaminhá-la para algum outro recurso responsável por responder a esse tipo de requisição

- Duas formas de encaminhamento:
  1. Chamada ao método *HttpServletResponse.sendRedirect(String url)*
  2. Criação de um objeto do tipo RequestDispatcher e chamada ao método *forward(HttpServletRequest, HttpServletResponse)*

Redirecionamentos

- A chamada do método *sendRedirect(String url)* faz com que o servidor envie ao cliente uma mensagem para que este envie a solicitação a uma nova URL
  - O servidor envia ao cliente uma mensagem HTTP com código 302 (o recurso foi movido para outra URL)
- O redirecionamento através da classe RequestDispatcher é interno ao servidor
  - Não há comunicação com o cliente

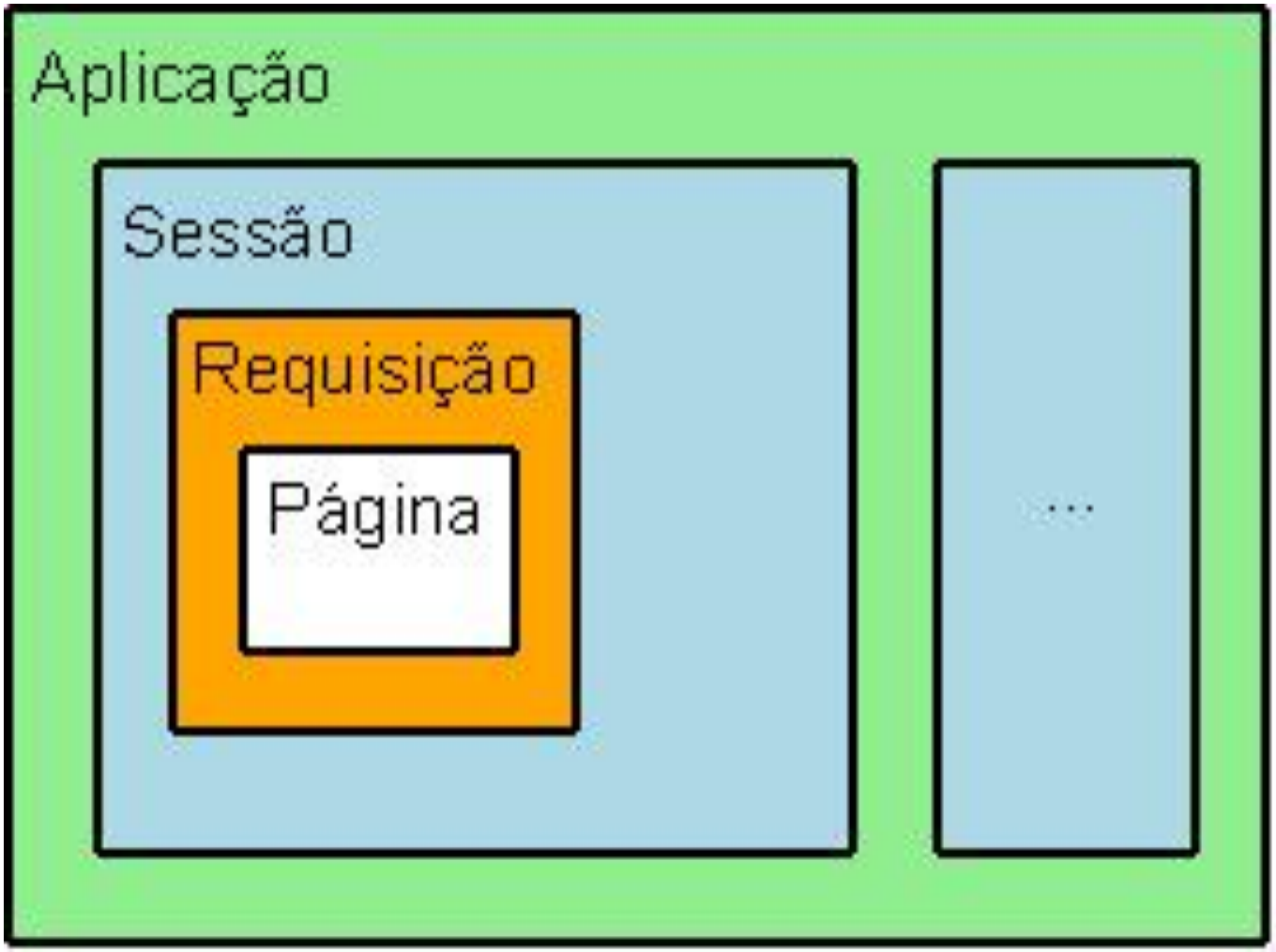
Redirecionamentos

- Em termos de práticos, para o cliente a primeira opção modifica o endereço URL no navegador, enquanto que a segunda não
- Em aplicações web, redirecionamentos são utilizados entre Servlets/JSPs como um mecanismo de delegação de tarefas

Escopo

- O escopo de um objeto indica quanto tempo o objeto existe depois de ter sido criado
- Estes variam do tempo de vida do contêiner até o tempo de vida de uma página individual
- Quatro níveis de escopos: (i) **Aplicação**, (ii) **Sessão**, (iii) **Requisição** e (iv) **Página**

Escopo



Escopo – Aplicação

- Objetos compartilhados por todos os servlets em uma dada aplicação
- Exemplo: conexão com banco de dados, lista de produtos numa aplicação de controle de estoque, etc.

Obtendo Acesso aos Tipos de Escopos

- Aplicação
  - A classe HTTPServlet possui um método *getServletContext()* que retorna um objeto do tipo *ServletContext*
  - Este objeto nos permite, por exemplo, criar atributos que existirão enquanto o servidor estiver no ar
  - Outra alternativa é através do método *init()* de HTTPServlet, o qual possui um parâmetro que é uma instância da classe *ServletConfig*; esta classe possui um método denominado *getServletContext()* que retorna a mesma referência citada no sub-item anterior

Escopo – Sessão

- Objetos compartilhados numa sessão vinculada a um usuário
- Exemplo: carrinho num site de compras

Obtendo Acesso aos Tipos de Escopos

- Sessão
  - A classe HTTPServletRequest (primeiro parâmetro dos métodos doGet()/doPost()) possui o método *getSession()*, que retorna uma referência para a sessão corrente
  - Caso não exista sessão corrente ativa, uma nova é criada

Escopo – Página

- Objetos compartilhados entre JSPs e servlets na página de execução atual
- Exemplo: variáveis locais declaradas em páginas JSPs

Escopo – Requisição

- Objetos compartilhados disponíveis para JSPs/Servlets numa solicitação
- Difere do escopo de página por permitir o compartilhamento também para JSPs/ Servlets incluídas ou redirecionadas
- Exemplo: parâmetros de campo de formulário

Exercício em Sala de Aula

- Acompanhe na sua máquina a construção de um servlet HelloWorld
  - Acompanhe a construção da classe (no Eclipse)
  - Analise o web.xml
  - Implante em um contêiner
  - Execute e verifique o resultado

Exercícios

1. Escreva um servlet que receba o nome e o telefone do usuário e retorne tais dados formatados em uma *string* passada ao servlet como parâmetro de inicialização.
   i. Use o método estático format da classe String
2. Crie um servlet que imprima, em uma tabela, todos os nomes de parâmetros enviados e seus valores.
3. Crie um servlet que imprima, em uma tabela, todos os nomes de cabeçalhos HTTP da requisição e seus valores.
4. Crie um servlet que retorne uma imagem.

Exercícios

5. Escreva um servlet simples que devolva uma página contendo o dia, mês, ano e hora.
6. Desenvolva a seguinte aplicação Web com as seguintes características
   i. Dois formulários para fazer autenticação de um usuário
      a. O primeiro deve obter o nome do usuário no sistema
      b. Caso seja um usuário cadastrado, a solicitação deve ser direcionada para um outro formulário que pedirá a senha
      c. Este segundo formulário deve ser enviado de forma segura, assim como a senha
   ii. A senha deve ser fornecida por, no máximo, 10 seg; caso seja fornecida após, o sistema retorna para a página inicial
   iii. As informações de usuário/senha podem estar no próprio servlet ou em algum banco de dados
   iv. Após a autenticação correta, deve ser enviada ao cliente uma página de boas-vindas
   v. Após falha na autenticação, o usuário deve ser direcionado para uma página de erro com um link para nova tentativa

Mais Informações

- Tutorial JEE 6
  - http://java.sun.com/products/servlet/index.html
- Caelum – Apostila Java para Desenvolvimento Web
  - http://www.caelum.com.br/apostila-java-web/servlets/
- DevMedia- Introdução a Servlets com NetBeans
  - http://www.devmedia.com.br/introducao-a-java-servlets-com-netbeans-parte-i/2181
- DevManuals – Servlet tutorials
  - http://www.devmanuals.com/tutorials/java/servlet/