Java 2 Enterprise Edition

Helder da Rocha
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

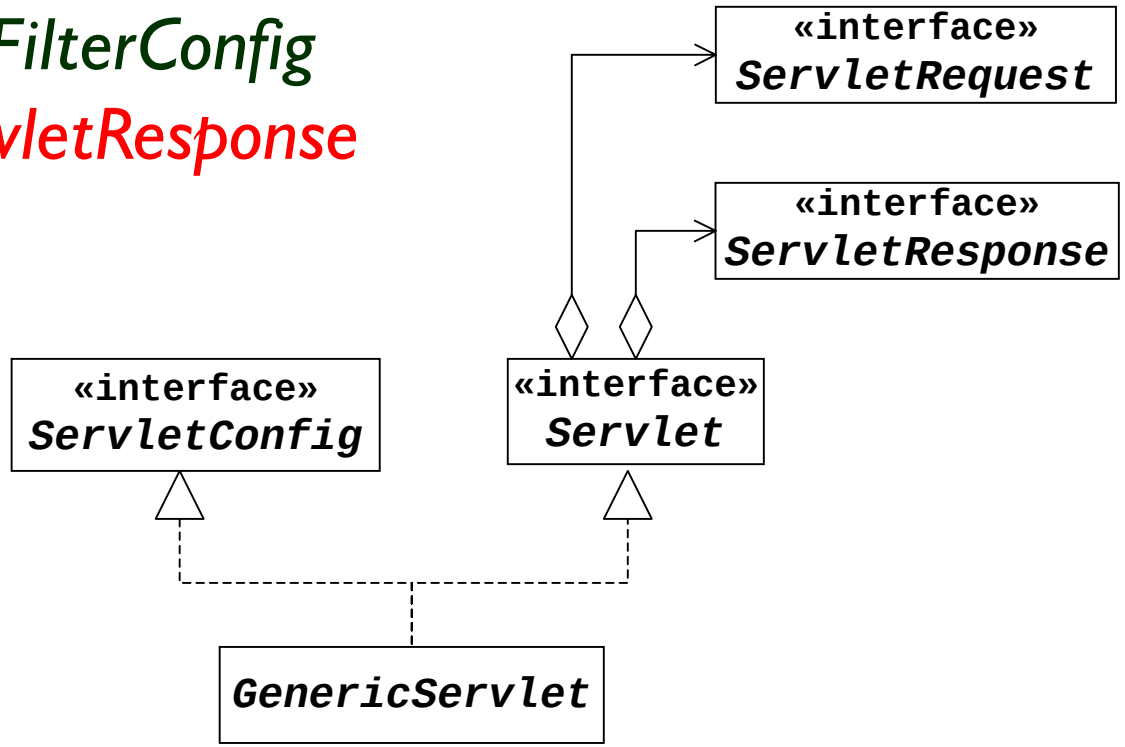
O que são servlets

- 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.

API: Servlets

Principais classes e interfaces de javax.servlet

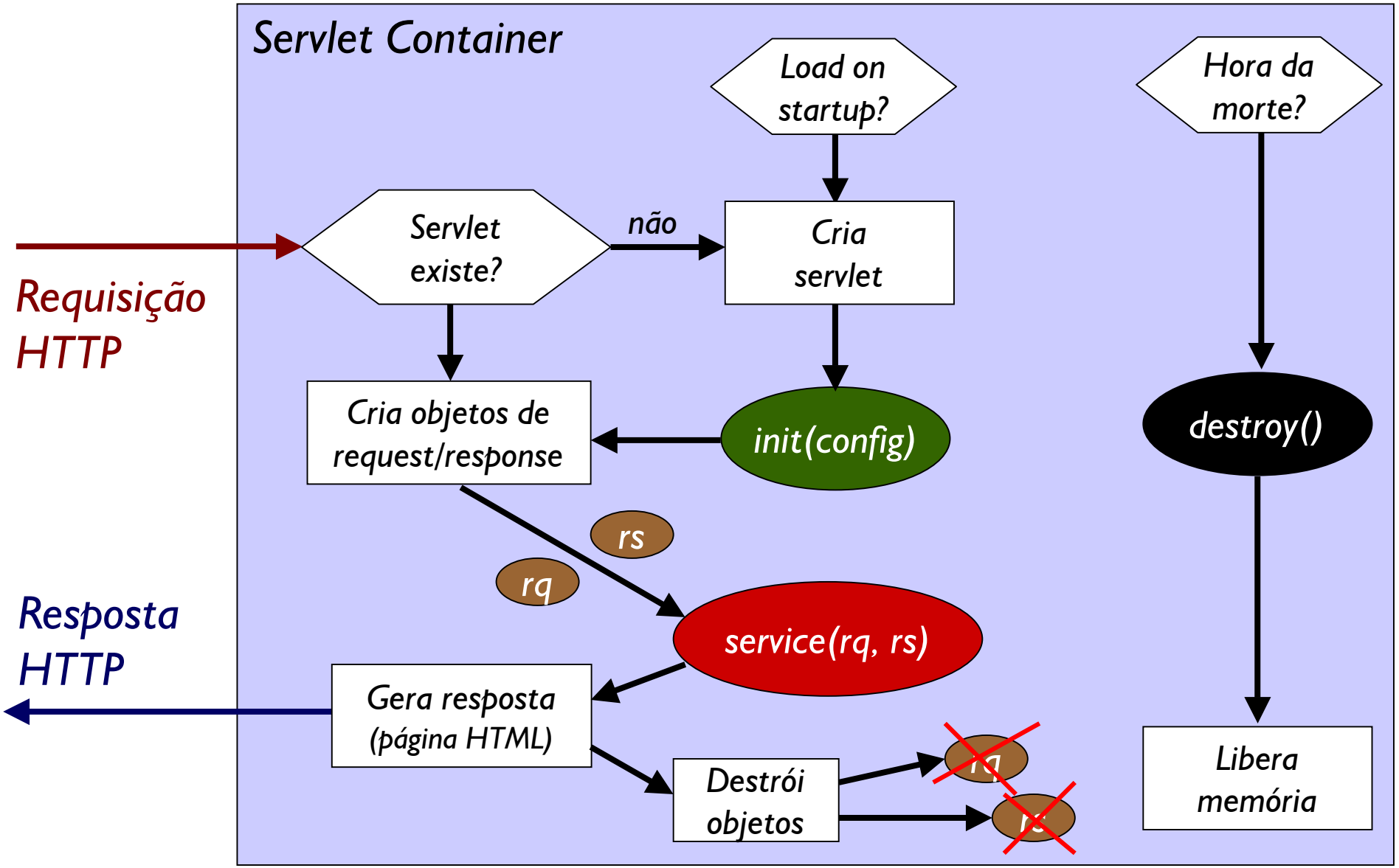
- *Interfaces*
  - *Servlet, ServletConfig, ServletContext*
  - *Filter, FilterChain, FilterConfig*
  - *ServletRequest, ServletResponse*
  - *SingleThreadModel*
  - *RequestDispatcher*
- *Classes abstratas*
  - *GenericServlet*
- *Classes concretas*
  - *ServletException*
  - *UnavailableException*
  - *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 aue 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



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)

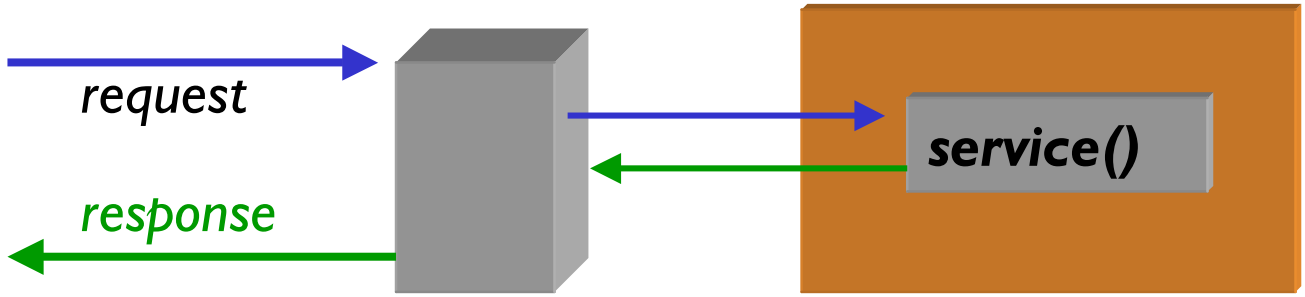
Métodos de serviço (2)

- 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).

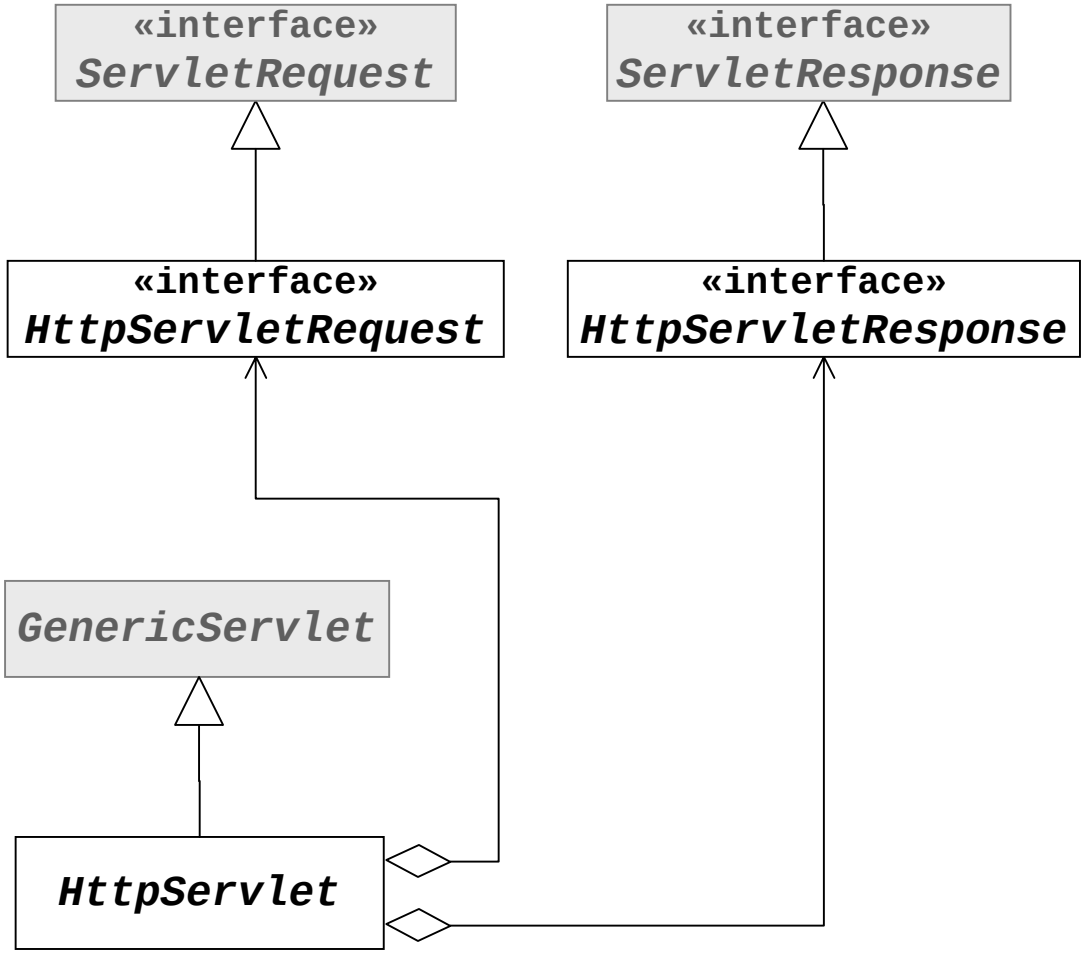


- Um servlet genérico deverá sobrepor este método e utilizar os objetos ServletRequest e ServletResponse recebidos para ler os dados da requisição e compor os dados da resposta, respectivamente

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()*

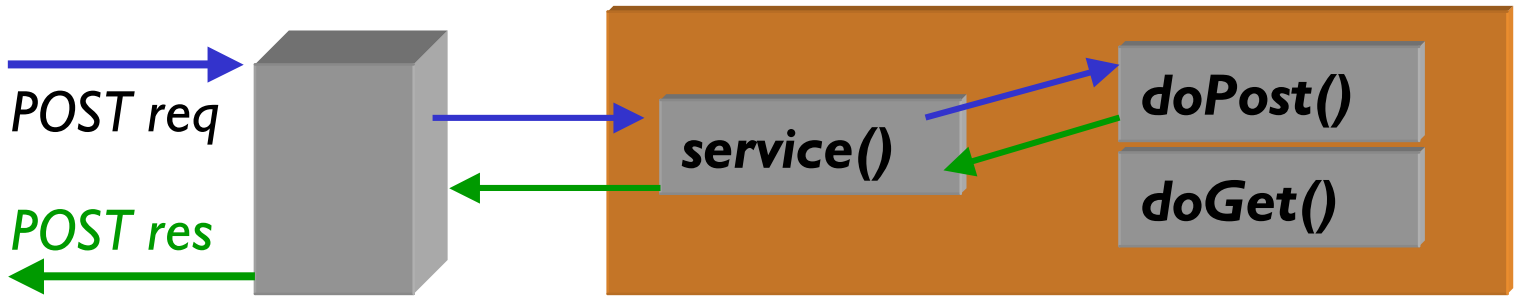
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

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()

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!");
    }
}
```

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 contrátio 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 sobrepoha 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>");
```

Compilação e Implantação de servlets

- 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/)

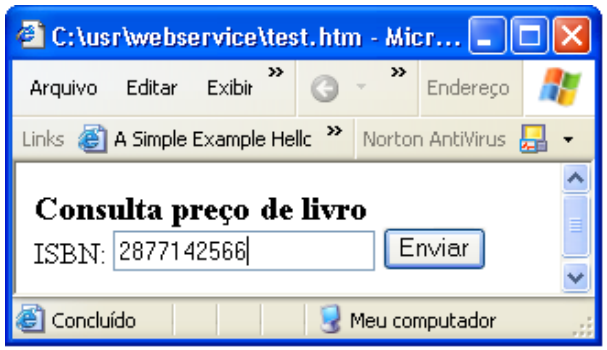
Usando o Ant

- 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>

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>
```

Exercícios

Para os exercícios abaixo, use um contexto cap06.

- 1. Crie um servlet (exercicio.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 allForm.html fornecido para testá-lo
- 2. Crie um servlet (exercicio.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)

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*

Servlet alias (mapeamento) no web.xml

- *É uma boa prática definir aliases 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>
```

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*

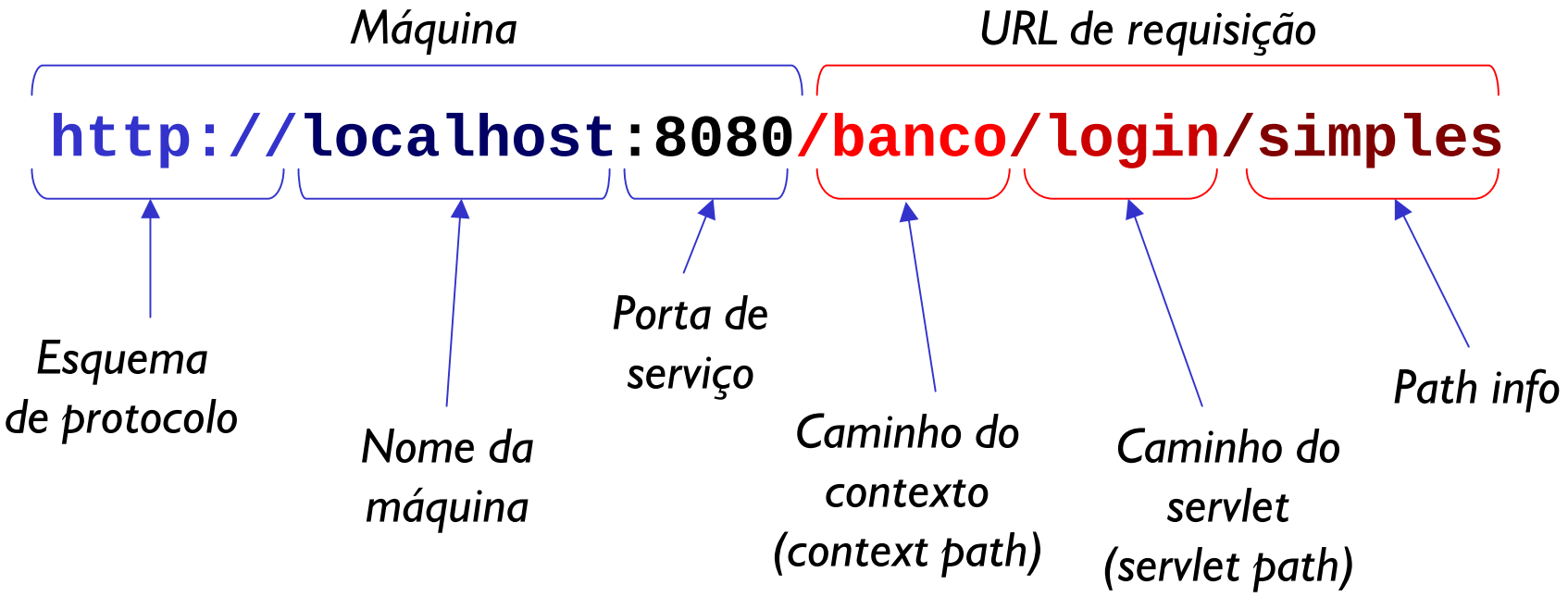
```
<url-pattern>/nome/*</url-pattern>
<url-pattern>/nome/subnome/*</url-pattern>
```

- *Mapeamento de extensão*
  - *Arquivos com a extensão serão redirecionados ao servlet*

```
<url-pattern>*.ext</url-pattern>
```

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*



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

Definição de 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>
```

Leitura de 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");
}
```

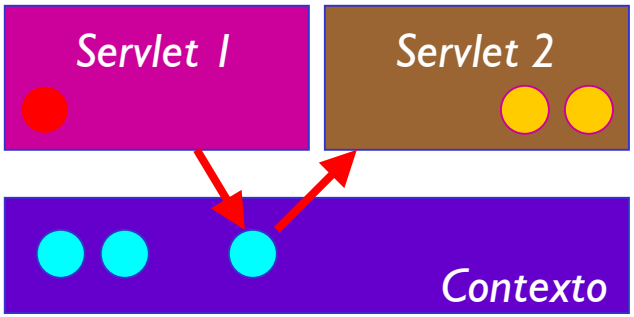
Carregamento de arquivos no contexto

- 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
}
```

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

Redirecionamento x Repasse

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

```
response.sendRedirect("url");
```

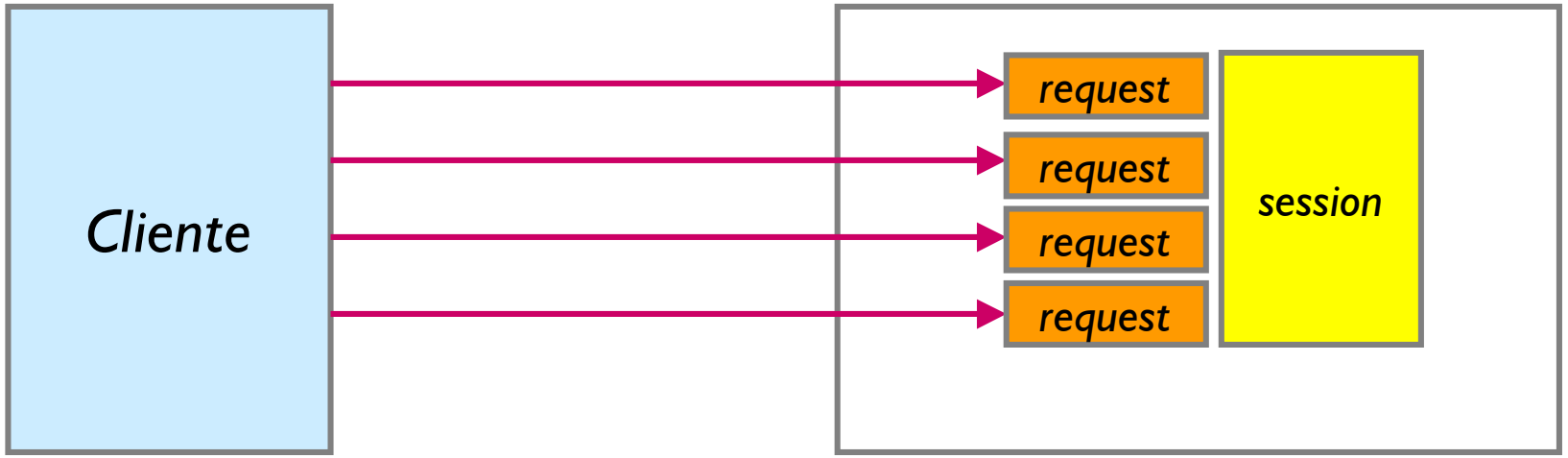
- Isto é o mesmo que fazer

```
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())

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*



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 de 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 à prova de clientes

- 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

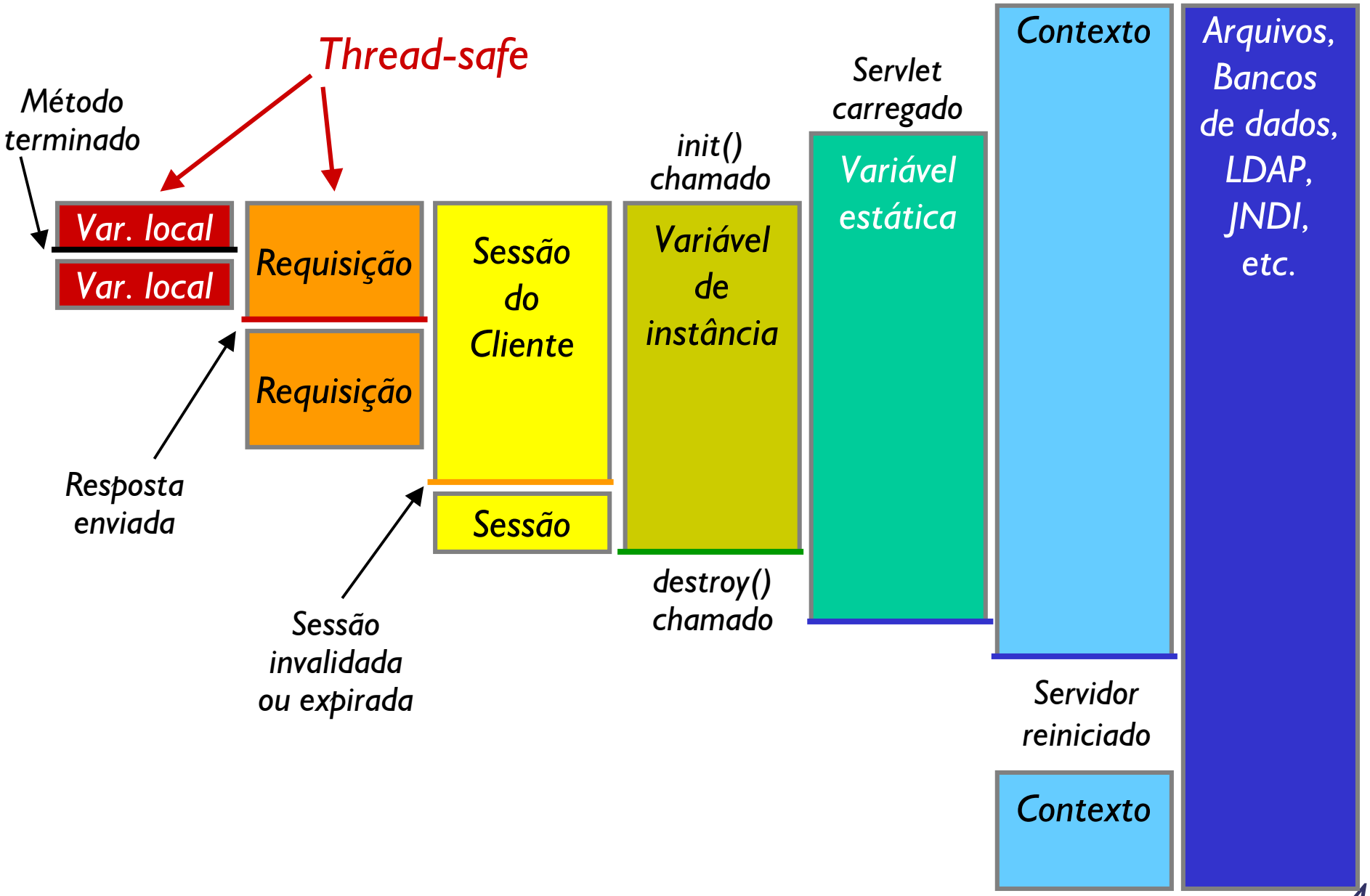
- Servlets podem compartilhar informações de várias maneiras
  - Usando meios persistentes (bancos de dados, arquivos, etc)
  - Usando objetos na memória por escopo (requisição, sessão, contexto)
  - Usando variáveis estáticas ou de instância
- Servlets oferecem três níveis diferentes de persistência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



Lidando com 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!

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("jbdc/Banco");
Connection con = ds.getConnection();
```

Recursos em servidores J2EE

- *Servlets rodando em servidores compatíveis J2EE podem acessar recursos através de JNDI (dominio java:comp/env)*
  - *Variáveis (environment entries)*
  - *Referências para componentes EJB*
  - *Referências para fábricas de recursos (conexões de banco de dados, URLs, serviço de e-mail, JMS, conectores EIS via JCA)*
  - *Serviços*
- *Para usar esses recursos*
  - *Servlet deve estar empacotado em um WAR*
  - *Nome das variáveis e referências devem ser declarados no web.xml*
  - *Servlet deve usar como contexto inicial o domínio java:comp/env*
- *Elementos (filhos de <web-app>) usados no web.xml*
  - *<env-entry>*
  - *<ejb-ref>*
  - *<resource-ref>*

Environment Entries

- *Alternativa global (para o WAR) aos <init-param>*
  - *São acessíveis dentro de qualquer servlet ou JSP da aplicação WAR*
  - *Não são visíveis por outras aplicações do servidor (não é um nome JNDI global - está abaixo de java:comp/env - é local à aplicação)*
  - *Acessíveis via ferramentas de deployment (podem ser redefinidas)*
- *Exemplo de uso dentro do <web-app>*

```
<env-entry>
  <env-entry-name>cores/fundo</env-entry-name>
  <env-entry-value>rgb(255, 255, 200)</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

  - *Tipos de dados legais são String e wrappers (Double, Integer, etc.)*
- *Uso dentro do servlet*

```
Context initCtx = new InitialContext();
String fgColor = (String)
                 initCtx.lookup("java:comp/env/cores/fundo");
```

Componentes EJB

- *Servlets e JSPs podem se comunicar com EJBs da aplicação declarando uma referência associada ao bean chamado*
  - *A referência deve informar o tipo do bean (Session, Entity ou MessageDriven e suas interfaces remota e home.*

```
<ejb-ref>
   <description>Cruise ship cabin</description>
   <ejb-ref-name>ejb/CabinHome</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <home>com.titan.cabin.CabinHome</home>
   <remote>com.titan.cabin.Cabin</remote>
</ejb-ref>
```

- *Componentes EJB são retornados como objetos CORBA que precisam ser reduzidos através da função narrow.*

```
InitialContext initCtx = new InitialContext();
Object ref = initCtx.lookup("java:comp/env/ejb/CabinHome");
CabinHome home = (CabinHome)
         PortableRemoteObject.narrow(ref, CabinHome.class);
```

Banco de dados

- *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
```

Conclusões

- Neste módulo abordamos os assuntos mais importantes sobre servlets, contextos e sessões
  - Para mais detalhes sobre esses assuntos consulte as referências no CD ou os slides do curso J550
- O próximo módulo abordará JSP, cuja API básica é a mesma usada nos servlets
  - Familiarizar-se com a construção e uso da API de servlets ajuda na implementação de páginas JSP mais eficientes
  - Mesmo com aplicações consistindo basicamente de páginas JSP, servlets têm papel importante na separação de responsabilidades (arquiteturas MVC, FrontController, etc.)

Exercícios

- 3. Crie aliases para os servlets criados nos exercícios anteriores 1 e 2:
  - /parameters para exercicio.ParameterList
  - /headers para exercicio.HeaderList
- 4. Parâmetros de inicialização
  - a) Guarde, como parâmetros de inicialização de ParameterList, duas cores: corDeFundo e corDoTexto que receba nomes de cor
  - b) Gere a página usando os parâmetros para colori-la
  - c) Use um parâmetro de contexto para gravar a fonte usada em todos os servlets do contexto.
- 5. Uso de Session
  - a) Crie um arquivo index.html contendo um link para /headers e um formulário, que receba um 'nome' e envie-o para /parameters usando POST
  - b) Em /parameters, grave o 'nome' na sessão
  - c) Em /headers, leia o nome da sessão e imprima-o

Exercício J2EE (opcional)

- 6. Utilize o servlet e EJBs fornecidos e faça com que seja possível executar a aplicação a partir do servlet
  - Tudo foi implementado menos o EAR e as referências EJB dentro do web.xml
  - Monte a aplicação, complete o web.xml e execute-a
  - Guie-se pelos comentários no código

Fontes

[1] *Jason Hunter, William Crawford. Java Servlet Programming, 2nd edition, O'Reilly and Associates, 2001*

[2] *J2EE Tutorial*

[3] *Marty Hall, Core Servlets and JSP, Prentice-Hall, 2000*

[4] *Jim Farley et. al. Java Enterprise in a NutShell, 2nd. edition, O'Reilly and Associates, April 2002*

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)