Java 2 Enterprise Edition

Helder da Rocha
www.argonavis.com.br

Sumário

- Introdução
  - Ciclo de vida de um servlet
  - Overview da API javax.servlet
- Programação básica de servlets
  - Objetos de escopo
  - Gerência de recursos compartilhados
  - Acesso a bancos de dados
  - Inicialização
  - Construção de métodos de serviço
- Recursos avançados
  - Filtragem de requisições e respostas
  - Transferência de controle
  - Acesso ao contexto do servlet
  - Sessões
  - Finalização
  - Servlets em servidores J2EE

O que são servlets

- Extensão de servidor escrita em Java
  - Servlets são applets 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 (implementa Servlet e ServletConfig)*
- *Classes concretas:*
  - *ServletException - exceção genérica dos servlets*
  - *UnavailableException*
  - *ServletInputStream e ServletOutputStream*

API: Servlets HTTP

Classes e interfaces mais importantes de javax.servlet.http

- *Interfaces*
  - *HttpServletRequest (estende javax.servlet.ServletRequest)*
  - *HttpServletResponse (estende javax.servlet.ServletRequest)*
  - *HttpSession*
- *Classes abstratas:*
  - *HttpServlet - principal classe estendida para criar servlets HTTP*
- *Classes concretas:*
  - *Cookie*

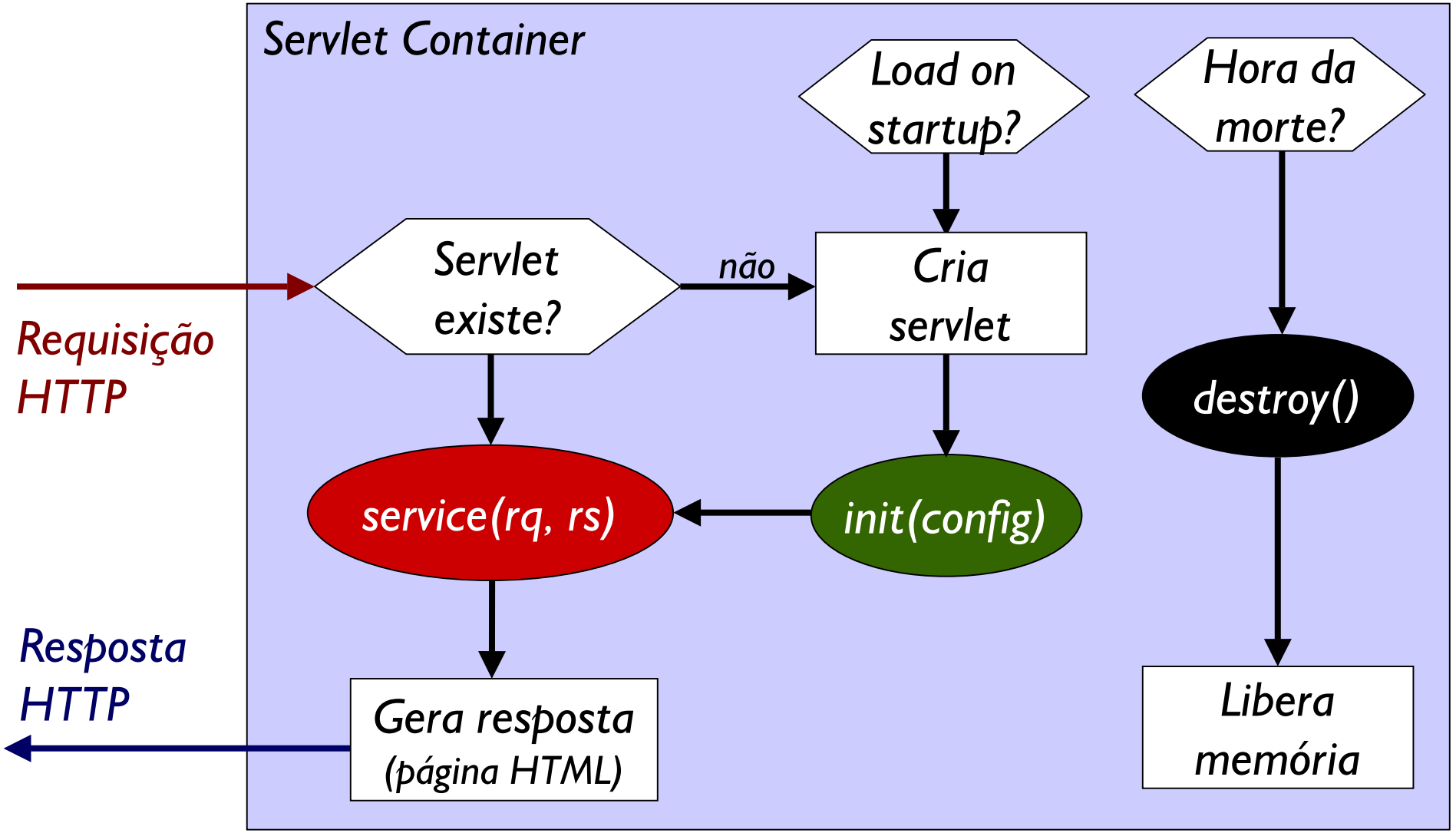
Exemplos de servlets

- Veja diretório cap06 que contém vários exemplos de servlets HTTP simples
- Para instalar (sem WAR)
  - Compile cada classe usando, no CLASSPATH, servlet.jar ou j2ee.jar (que contém as classes necessárias)
  - Copie as classes compiladas para um contexto existente no Jakarta-Tomcat (ROOT/WEB-INF/classes)
  - Execute os servlets através da URL http://localhost:8080/servlet/nome.do.Servlet
- Aplicação completa no servidor J2EE
  - Siga os passos para implantar Duke's Bookstore no J2EE Tutorial (use inicialmente o deploytool para montar o web.xml)
  - Configure os aliases, defina os filtros, bancos de dados, mapeamento de exceções, etc.

Ciclo de vida

- Controlado pelo container
- Quando o servidor recebe uma requisição, ela é repassada para o container que a delega a um servlet. O container
  - Carrega a classe na memória
  - Cria uma instância da classe do servlet
  - Inicializa a instância chamando o método init()
- Depois aue o servlet foi inicializado, cada requisição é executada em um método service()
  - Container passa um objeto requisição (Request) e resposta (Response) e chama service()
- Quando o container decidir remover o servlet da memória, ele o finaliza chamando destroy()

Ciclo de vida



Eventos do ciclo de vida

- Pode-se monitorar o ciclo de vida de um servlet definindo objetos listener cujos métodos são chamados quando eventos ocorrem
- Controle de ciclo de vida do servlet
  - ServletContextListener (e ServletContextEvent): inicialização e destruição do servlet
  - ServletContextAttributeListener (e evento): alteração de atributos do contexto
- Controle do ciclo de vida de uma sessão
  - HttpSessionListener (e evento): criação, invalidação e timeout da sessão do cliente
  - HttpSessionAttribute (e evento): alteração de atributos da sessão
- Exemplo: Duke's bookstore: listeners.ContextListener

Compartilhamento com objetos de escopo

- Componentes Web podem compartilhar informações de várias maneiras
  - Usando meios persistentes (bancos de dados, arquivos, etc)
  - Usando objetos de escopo (na memória)
- Objetos de escopo oferecem quatro níveis diferentes de persistência (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)
  - Página: dura uma página JSP (javax.servlet.jsp.PageContext)
- Para gravar dados em um objeto de escopo

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

- Para recuperar os dados

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

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 para cada instância
  - É preciso sincronizar blocos críticos para evitar problemas decorrentes do acesso paralelo
  - Para situações onde multithreading é inaceitável, servlet deve implementar a interface SingleThreadModel (só um thread estará presente no método service() ao mesmo tempo.)

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*

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

- *Veja exemplos no Duke's Bookstore: database.BookDB*

Inicialização

- *Tarefa realizada uma vez na chamada de init(config)*
- *Deve-se sobrepor init(config) com instruções que serão realizadas uma vez na vida do servlet*
  - *Carregar parâmetros de inicialização, dados de configuração*
  - *Obter outros recursos via JNDI*
- *Falha na inicialização deve provocar UnavailableException*

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

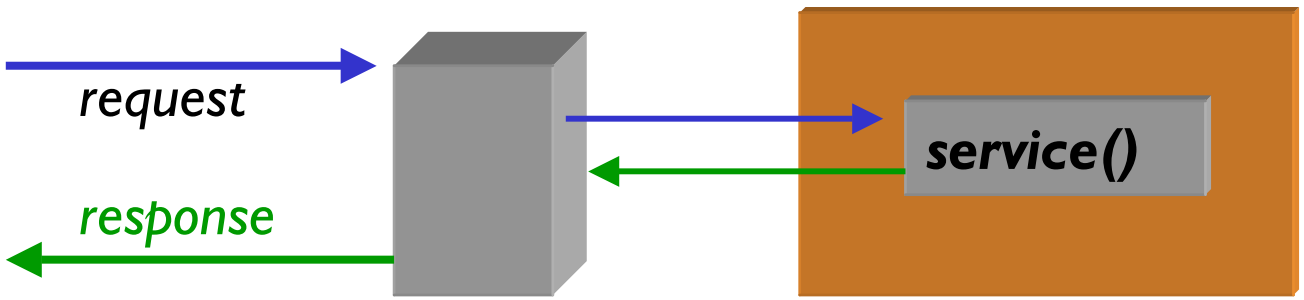
Métodos de serviço

- 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 é o método abstrato
  - public void service(ServletRequest, ServletResponse)

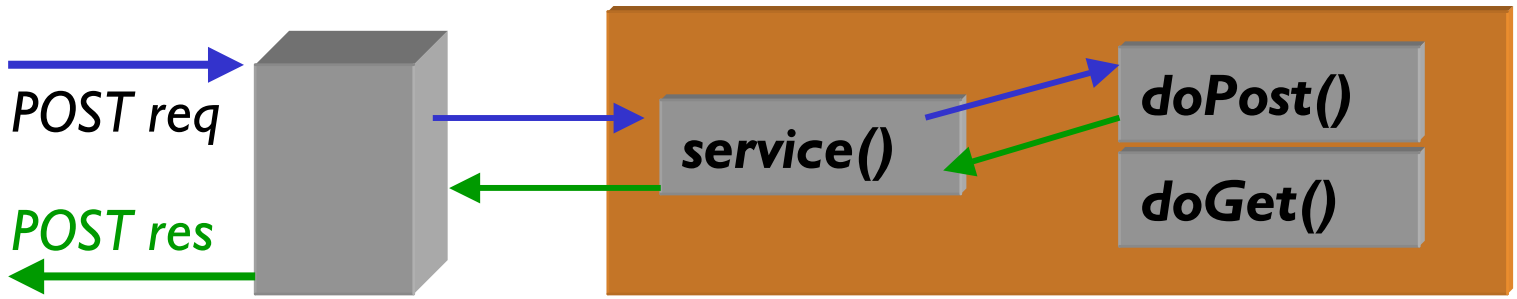
  definido em javax.servlet.Servlet.
- Sempre que um servidor repassar uma requisição a um servlet, ele chamará o método service().



- 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

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()*

Obtenção de dados de requisições

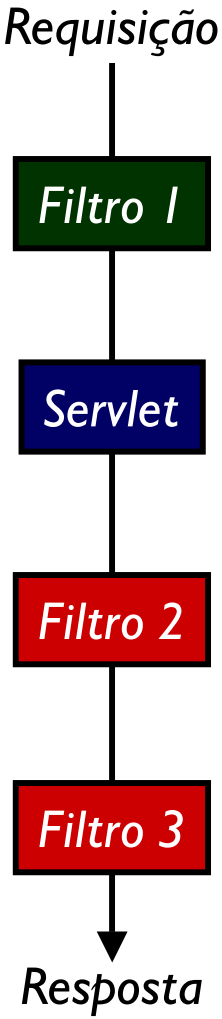
- Os métodos de HttpServletRequest podem ser chamados dentro de qualquer doXXX() ou service
- Alguns métodos de HttpServletRequest
  - Cookie[] getCookies() - recebe cookies do cliente
  - String getRemoteUser() - obtém usuário remoto (se autenticado, caso contrátio devolve null
  - HttpSession getSession() - retorna a sessão
  - getLocale() - obtém Locale atual
  - getParameter(param) - obtém parâmetro HTTP
  - setAttribute("nome", obj) - define um atributo obj chamado "nome".
  - getAttribute("nome") - recupera atributo chamado nome
  - getRequestDispatcher() obtém RequestDispatcher

Preenchimento de uma resposta

- Alguns métodos de HttpServletResponse
  - addCookie(Cookie c) - adiciona um novo cookie
  - addHeader(String nome, String valor) - adiciona um cabeçalho HTTP
  - encodeURL(String url) - envia a URL com informação de identificador (sessionid)
  - sendRedirect(String location) - envia informação de redirecionamento para o cliente
  - Writer getWriter() - obtém um Writer para escrever na saída padrão
  - reset() - limpa toda a saída inclusive os cabeçalhos
  - resetBuffer() - limpa toda a saída, exceto cabeçalhos

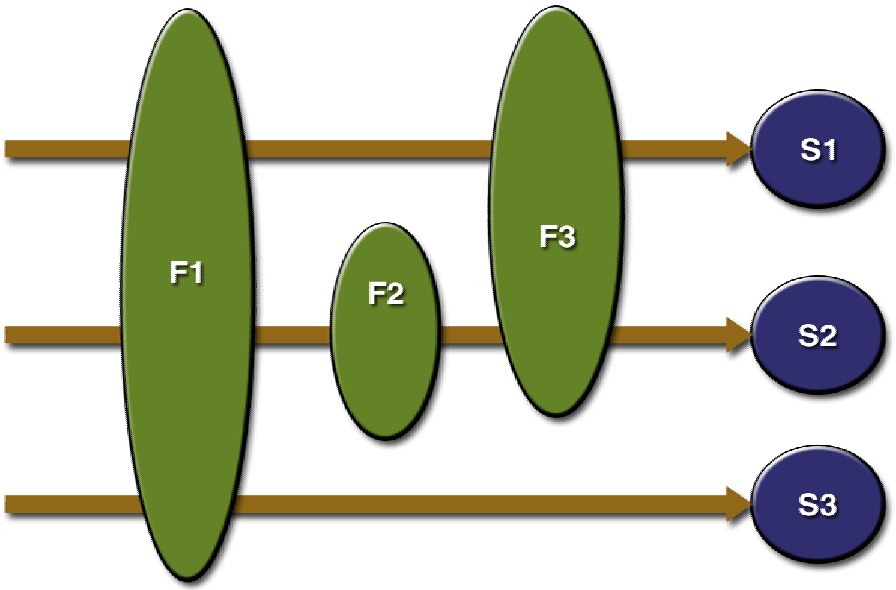
Filtros

- *Um filtro é um objeto que pode transformar o cabeçalho, conteúdo, ou ambos, de uma resposta ou requisição*
  - *Interceptam uma requisição ou resposta*
  - *Podem ser encadeados*
- *Principais tarefas que um filtro pode executar*
  - *interromper o fluxo da requisição/resposta*
  - *modificar os dados da requisição ou resposta*
  - *interagir com recursos externos*
- *Aplicações típicas*
  - *autenticação, logging e criptografia*
  - *compressão de dados, transformação XML*
- *Para instalar e usar um filtro é preciso*
  - *programar o filtro*
  - *programar requisições e respostas filtraddas*
  - *especificar a ordem de chamada para cada recurso*



Filtragem de requisições e respostas

- API consiste das classes Filter, FilterChain e FilterConfig
  - Para criar um filtro é preciso implementar a interface Filter e seu método doFilter(req, res, corrente), métodos init() e destroy()
- Filtros são associados a servlets no web.xml
  - A ordem em que aparecem no web.xml determina a ordem que irão ser chamados
  - Quando um servlet que possui filtros associados é chamado, seus filtros são chamados em seqüência



Inclusão de recursos

- *Objetos RequestDispatcher servem para repassar requisições para outra página ou servlet. Seus dois principais métodos são*

```
nome_dispatcher.include(request, response)
nome_dispatcher.forward(request, response)
```

- *Esses métodos não podem definir cabeçalhos*
- *Para obter um RequestDispatcher*

```
RequestDispatcher rd = getRequestDispatcher("url");
```

- *Para estender a requisição para outra máquina*

```
rd.forward(request, response)
```

- *Para redirecionar para outra página (em nova requisição)*

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

Acessando o contexto

- *O contexto é representado pela classe SessionContext. A partir dele se obtém*
  - *Parâmetros de inicialização (a partir de objeto config):*

```
SessionContext contx = config.getSessionContext();

String param = contx.getInitParameter("teste");
```

  - *Recursos associados ao contexto*

```
Object recurso = contx.getResource("url");
```

  - *Caminhos absolutos e relativos*

```
String caminho = contx.getRealPath();
```

  - *Serviços de log*

```
contx.log("teste");
```

Sessões

- *Como o HTTP não mantém estado, são as aplicações Web que precisam cuidar de mantê-lo quando necessário*
- *Sessões são representados por objetos HttpSession e são obtidas a partir de uma requisição*

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

  - *Método retorna a sessão ou cria uma nova*
- *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)*
- *get/setAttribute() (em session) guarda objetos em uma sessão*
  - *Através do nome, podem ser recuperados por qualquer componente do mesmo contexto e que faça parte da mesma sessão*
- *Eventos de ligação com uma sessão podem ser controlados*
  - *HttpSessionBindingListener e HttpSessionActivationListener*

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

Gerência de sessões

- *HTTP não oferece meios de saber que um cliente não precisa mais de uma sessão*
  - *O servidor define um timeout para a duração da sessão desde a última requisição do cliente*
  - `getMaxInactiveInterval()` *recupera valor de timeout*
  - `setMaxInactiveInterval()` *define novo valor para timeout*
  - *Pode também ser definido no web.xml*
- *Para destruir uma sessão*

```
session.invalidate();
```

Session tracking

- *A sessão é implementada com cookies se o cliente os suportar*
  - *Caso o cliente não suporte cookies, o servidor precisa usar outro maio de manter a sessão (copiar o código da sessão para a página seguinte, por exemplo)*
- *Sempre que uma página contiver uma URL para outra página da aplicação, a URL deve estar dentro do método encodeURL()*

```
out.print("<a href='" +
                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 (url?jsessionid=A424JX08S99)*

Finalização

- *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. Veja técnicas e exemplos no J2EE Tutorial em Servlets12.html*

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);
```

Conexões de 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
```

Outras conexões

- *Outras fábricas de objetos e conexões podem ser obtidas além de javax.sql.DataSource*
- *As fábricas padrão são*
  - *javax.jms.QueueConnectionFactory (cap 4*)*
  - *javax.jms.TopicConnectionFactory (cap 4)*
  - *javax.mail.Session (cap 17)*
  - *java.net.URL (cap 17)*
- *Outras fábricas de objetos podem ser definidas através da implementação JCA*
  - *Veja exemplo simples de conexão direta Oracle via JCA no último capítulo do J2EE Tutorial*
  - *javax.resource.cci.ConnectionFactory*
- *Referências de ambiente <resource-env-ref>*
  - *javax.jms.Queue e javax.jms.Topic (cap 4)*

* *módulos deste curso que abordam acesso a esses objetos*

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@ibpinet.net

www.argonavis.com.br