



Tehnologii Java


Curs -

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică

Universitatea "Al. I. Cuza" Iași



Servleturi

Cuprins

- Introducere
- Java Servlet API
- Comunicarea cu clienții
- Ciclul de viață al unui servlet
- Contextul de execuție
- Sesiuni de lucru
- Folosirea explicită a cookie-urilor
- Comunicarea servleturilor
- Rularea servleturilor

Introducere

Ce este un servlet ?

- **Concept:** Componentă software care interacționează cu clienții (din rețea) printr-o paradigmă de tip **cerere-raspuns**
- **Java EE:** Componentă Web gestionată de un container care *extinde funcționalitatea unui server Web*:
 - poate genera conținut dinamic
 - poate controla fluxul unei aplicații Web
- Parte fundamentală a nivelului Web al platformei Java EE

Funcționarea unui servlet



Java Servlet API

Clase și interfețe

- **javax.servlet**
suport pentru servleturi independente de protocol
- **javax.servlet.http**
suport pentru protocolul HTTP

Abstracțiunea centrală o reprezintă interfața **Servlet**.

Orice servlet implementează această interfață fie direct, fie indirect prin extinderea clasei **HttpServlet**.

Metode cheie ale servleturilor HTTP

- **service** - numitorul comun al tuturor metodelor de tip `do[TipCerere]`; implicit, această metodă determină tipul unei cereri și apelează metoda `do[TipCerere]` corespunzătoare.
- **doGet** - procesarea cererilor de tip GET
- **doPost** - procesarea cererilor de tip POST
- **init** - apelată la inițializare
- **destroy** - apelată la eliberarea din memorie



Structura generală a unui servlet

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

public class MyServlet extends HttpServlet {
    public void init { ... }
    public void service { ... }
    public void doGet { ... }
    public void doPost { ... }
    public String getServletInfo { ... }
    public void destroy { ... }
}
```

Comunicarea cu clienții

Când un servlet acceptă un client primește două obiecte:

- **ServletRequest**, ce încapsulează toate informațiile trimise de client către servlet
- **ServletResponse**, ce descrie răspunsul trimis de către servlet înapoi către client

Implementările concrete ale acestor interfețe sunt:

- **HttpServletRequest**
- **HttpServletResponse**

Cererea: `HttpServletRequest`

Obiectul de tip `HttpServletRequest` permite accesul la datele trimise de client prin metodele de mai jos:

- `getParameter`, `getParameterValues`, `getParameterNames`
- `getQueryString` - returnează un șir cu cererea venită în format primitiv (doar pentru cereri de tip GET)
- `getReader`, returnează un flux de tip `BufferedReader`
- `getInputStream`, returnează un flux de tip `ServletInputStream`

Răspunsul: HttpServletResponse

Obiectul de tip `HttpServletResponse` pune la dispoziție două metode de a returna date către client:

- **getWriter**, returnează un flux de ieșire de tip `Writer`
- **getOutputStream**, returnează un flux de ieșire de tip `ServletOutputStream`

Observație

Înainte de a scrie informații pe fluxul de ieșire trebuie setat header-ul HTTP cu metoda **setContentType**, argumentul cel mai uzual fiind `"text/html"`.

Exemplu simplu

In exemplul următor, creăm un servlet care să răspundă oricărei cereri prin afișarea datei și orei curente.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Simplu extends HttpServlet {
    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException {
        response.setContentType("text/html");
        PrintWriter out = new PrintWriter(response.getWriter());
        out.println("<html><head><title>Un servlet</title></head>");
        out.println("<h1> " + new java.util.Date() + " </h1>");
        out.println("</html>");
        out.close();
    }
}
```

Trimiterea și primirea parametrilor

Uzual, trimiterea parametrilor de către clienți se face prin formulare HTML.

```
<FORM METHOD="[GET sau POST]" ACTION="[URL servlet]">
  Parametru1:
  <INPUT TYPE="[tip-param]" NAME="[nume-param]" value="[val-init]"/>
  <BR/>
  Parametru2:
  <INPUT TYPE="[tip-param]" NAME="[nume-param]" value="[val-init]"/>
  ...
  <BR/>
  <INPUT TYPE="submit" NAME="ok" VALUE="Send" />
  <INPUT TYPE="reset" NAME="reset" VALUE="Reset" />
</FORM >
```

unde `tip-param` poate fi unul din tipurile :

`text|password|checkbox|radio|submit|reset|file|hidden|image|button`

Exemplu - Formularul GET

Inregistrarea unor persoane într-un fișier.

```
<FORM METHOD="GET" ACTION="http://localhost:8080/app/Utilizatori">  
  Nume:  
  <INPUT TYPE="text" NAME="nume" size=20 value="" />  
  <BR />  
  Telefon:  
  <INPUT TYPE="text" NAME="telefon" size=20 value="" />  
  </BR>  
  <INPUT TYPE="submit" NAME="ok" VALUE="Send">  
  <INPUT TYPE="reset" NAME="reset" VALUE="Reset">  
</FORM >
```

Exemplu - Servletul (doGet)

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

public class Utilizatori extends HttpServlet {
    PrintWriter pw=null;
    public void init(ServletConfig config) {
        String filename =
            config.getServletContext().getRealPath("users.txt");
        try {
            pw = new PrintWriter(new FileWriter(filename));
        } catch(IOException e) {
            System.out.println(e);
        }
    }

    public void destroy() {
        pw.close();
    }
}
```

Exemplu - Servletul (cont.)



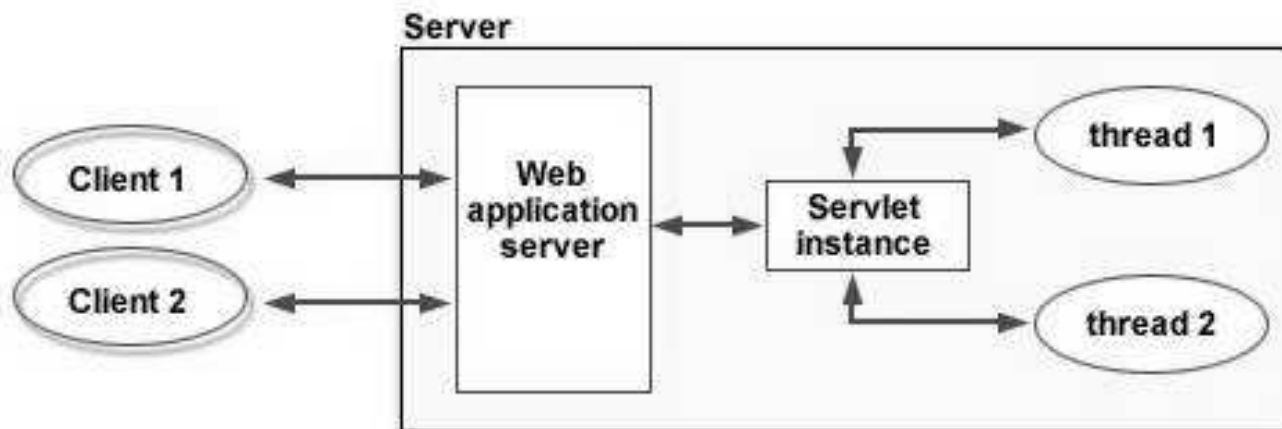
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException {
    String nume = request.getParameter("nume");
    String telefon = request.getParameter("telefon");
    pw.println(nume + "\t" + telefon);
    pw.flush();

    response.setContentType("text/html");
    PrintWriter out = new PrintWriter(response.getWriter());
    out.println("<html><head><title>Utilizatori</title></head>");
    out.println(nume + " a fost inregistrat");
    out.println("</html>");
    out.close();
}
}
```

**http://localhost:8080/app/Utilizatori ?nume=ion &
telefon=123456 & ok=Send,**



Tratarea cererilor concurente

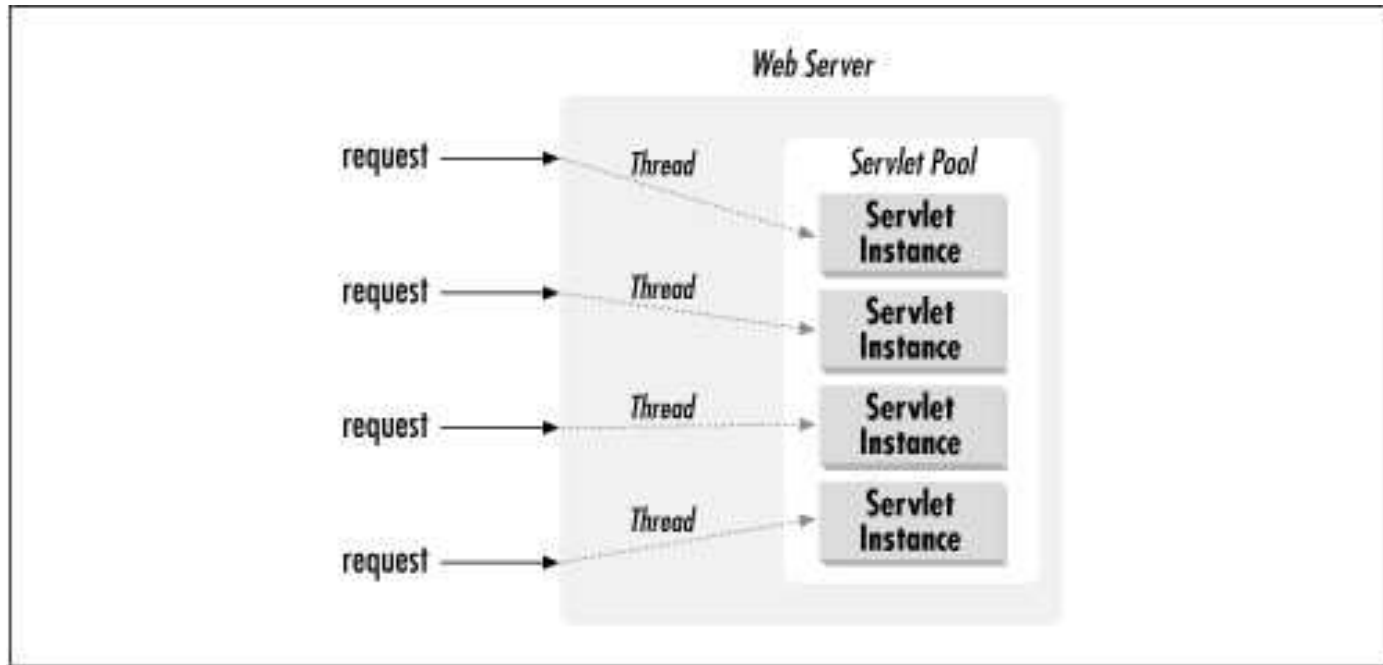


Accesul la resurse comune

- Sincronizarea metodelor ce gestionează resursa comună.
- Declararea servletului astfel încât să nu trateze decât o cerere la un moment dat.

```
public class MyServlet
    extends HttpServlet
    implements SingleThreadModel
    ...
}
```

Single Thread Model



Ciclul de viață al unui servlet

Etape



Ciclul de viață al unui servlet constă în următoarele etape:

- Serverul încarcă și inițializează servletul.
- Servletul tratează cereri venite din partea unor clienți Web.
- Serverul elimină servletul din memorie.



Inițializarea

După încărcarea în memorie a servletului, serverul va apela metoda **init** a acestuia.

- Inițializarea este executată **o singură dată** pe toată durata de viață a servletului.
- Nici o cerere nu va fi tratată până la terminarea inițializării.
- Servletul nu va deveni funcțional dacă:
 - Pe parcursul inițializării va fi aruncată o excepție de tipul **ServletException**.
 - Metoda `init` nu se termină într-o perioadă specificată de server.

Acțiuni realizate la inițializare

Cele mai uzuale acțiuni executate în metoda `init` sunt:

- Citirea unor parametri de inițializare a servletului.
- Deschiderea unor fișiere.
- Realizarea unor conexiuni la baze de date.
- Realizarea unor conexiuni în rețea.

Dacă apare o eroare la realizarea acestor operațiuni care compromite capacitatea servletului de a satisface cererile clienților trebuie aruncată o excepție de tipul **UnavailableException**.

Distrugerea

La distrugerea unui servlet serverul apelează automat metoda **destroy** a acestuia.

Uzual, în metoda `destroy` vor fi închise fișierele sau conexiunile deschise în metoda `init`.

Metoda `destroy` este apelată doar după ce firele de execuție **service** s-au terminat sau o anumită perioadă de timp specificată în parametrii de configurare ai serverului s-a scurs.

Metoda este executată **o singură dată** și, după apelul ei, nu va mai fi realizat nici un apel al metodei `service`.



Contextul de execuție



Informații contextuale

Un servlet "trăiește și moare" în interiorul unui proces ce rulează pe un server Web. În tot acest timp el poate obține diverse informații despre acest proces, care mai este numit și **contextul său de execuție**. Aceste informații pot fi grupate astfel:

- Informații primite la inițializarea servletului.
- Informații despre și de la server - disponibile pe toată durata de viață a servletului.
- Informații contextuale specifice cererilor primite.

Informații primite la inițializare

Sunt transmise servletului prin argumentul **ServletConfig** al metodei **init**.

Fiecare server Web oferă propria sa modalitate de a transmite diverși **parametri de inițializare**, uzual aceștia fiind specificați în fișierul de proprietăți ce descrie servletul.

Valorile parametrilor de inițializare pot fi obținute prin metoda

getInitParameter([nume-parametru])

Informații despre/de la server

Legătura unui servlet cu serverul Web este realizată printr-un obiect de tip **ServletContext**. Acesta permite:

- Accesarea resurselor disponibile servletului.
- Monitorizarea evenimentelor (log).
- Setarea/obținerea unor attribute disponibile și altor servleturi din același context.

Obținerea sa se realizează prin metoda **getServletContext** a clasei **ServletConfig**.

Exemplu ServletContext

```
public class SimpleServlet extends HttpServlet {
    ServletConfig config = null;
    public void init(ServletConfig config) {
        this.config = config;
        config.getServletContext().log("Init OK");
    }
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException {
        ServletContext context = config.getServletContext();
        context.log(context.getServerInfo());
        context.log("doGet OK");
    }
}
```

Contextul unei cereri

Informațiile contextuale legate de fiecare cerere sunt oferite de obiectul `ServletRequest` primit ca argument de metoda `service`. Câteva exemple de astfel de metode sunt:

- `getProtocol`
- `getScheme` (`https`, `http`, `ftp`, etc.)
- `getServerName`
- `getServerPort`



Comunicarea servleturilor



Folosirea unor resurse externe

- Realizarea unei cereri HTTP către URL-ul resursei respective.
- Dacă resursa se găsește pe același server cu servletul, poate fi folosit un obiect de tip **RequestDispatcher**. Acesta poate efectua una din acțiunile:
 - **forward**
 - **include**

Crearea unui RequestDispatcher

Se realizează cu metoda **getRequestDispatcher** a clasei `ServletContext` care primește ca argument URL-ul relativ sau absolut al resursei accesate.

```
RequestDispatcher dispatcher =
    context.getRequestDispatcher( "ThankYou.html" );

String url = "http://localhost:8080//servlet//ThankYouServlet"
RequestDispatcher dispatcher =
    context.getRequestDispatcher(url);
if (dispatcher==null) {
    //resursa nu este disponibila
    response.sendError(response.SC_NO_CONTENT);
}
...
```

Acțiunea *forward*

Transferă responsabilitatea generării răspunsului către client resursei apelate. Se realizează prin metoda **forward** a clasei `RequestDispatcher`.

```
RequestDispatcher dispatcher =  
    context.getRequestDispatcher("resursa");  
dispatcher.forward(request, response);  
//nu este deschis nici un flux de iesire catre client
```

Atenție

Servletul nu trebuie să deschidă vreu flux de ieșire către client, altfel va fi generată o excepție de tipul `IllegalStateException`.

Acțiunea *include*

Include răspunsul primit de la resura apelată în cadrul răspunsului propriu: Se realizează prin metoda **include** a clasei `RequestDispatcher`.

```
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("resursa");
PrintWriter out = response.getWriter();
out.println("..."); //raspuns propriu servletului

//este inclus raspunsul resursei apelate
dispatcher.include(request, response);}

out.println("..."); //continuare raspuns propriu servletului
out.close();
```

Comunicarea prin attribute

Servleturile care rulează pe același server pot comunica prin intermediul **atributelor**. Atributele sunt perechi de tipul *nume-valoare*, gestionate de **contextul de execuție**.

Pentru a evita conflictele numele atributelor au uzual forma: *nume-servlet.nume-atribut*.

Metodele de manipulare a atributelor sunt în clasa **ServletContext**:

- **setAttribute**
- **getAttribute**
- **removeAttribute**



Sesiuni de lucru



Ce este o sesiune de lucru ?

Păstrarea unei sesiuni de lucru se referă la memorarea clientului care a adresat o cerere unui servlet.

Protocoloalele Internet sunt de două tipuri:

- **cu stare (stateful): telnet, FTP**
- **fără stare (stateless): HTTP**

Așadar, servleturile trebuie să utilizeze diverse tehnici pentru păstrarea sesiunilor.

Păstrarea unei sesiuni

- Folosirea **cookie**-urilor.
- Rescrierea URL-ului apelat (URL rewriting).
- Câmpuri ascunse în formulare.

Fiecare sesiune are un identificator unic (ID).
O sesiune persistă pentru o anumită perioadă de timp,
care poate fi și indefinită.

Păstrarea sesiunilor va fi realizată folosind clasa
HttpSession.

Clasa HttpSession

Un obiect al acestei clase permite stocarea datelor unei sesiuni și asocierea între un utilizator și datele sesiunii sale de pe server.

Etapele de baza în folosirea unui obiect `HttpSession` sunt:

- **Obținerea (crearea)** unei sesiuni.
- **Citirea sau scrierea** datelor folosind acest obiect.
- **Terminarea** sesiunii (invalidarea ei).

Obținerea (crearea) unei sesiuni

Crearea unei sesiuni se realizează cu metoda **getSession** a clasei **HttpServletRequest**.

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws IOException {
    HttpSession session = req.getSession(true);
    ...
    out = res.getWriter();
    ...
}
```

Alte metode utile:

- isNew
- getCreationTime
- getLastAccessedTime

Gestionarea unei sesiuni

Un obiect de tip `HttpSession` gestionează:

- **Proprietăți standard** ale unei sesiuni, cum ar fi identificatorul sesiunii.
- **Date specifice** sesiunii, memorate sub forma unei colecții de perechi **cheie-valoare**.

Cheia este de tip `String` și este uzual de forma:
nume-servlet.nume-proprietate.

Valoarea este de tip `Object`.

Metodele folosite pentru manipularea acestor informații sunt: **setAttribute**, **getAttribute**

Exemplu de folosire a unei sesiuni

Suma unor numere trimise de client.

```
ServletContext context = config.getServletContext();
HttpSession session = request.getSession(true);
double numar; //parametrul trimis ca argument de client
numar = Double.parseDouble(request.getParameter("numar"));
if (session.isNew())
    suma = new Double(0);
else
    suma = (Double)session.getAttribute("MyServlet.suma");
}
suma = new Double(suma.doubleValue() + numar);
session.putAttribute("MyServlet.suma", suma);
...
```

Terminarea unei sesiuni

- Apelul metodei **invalidate**,
- Automat la expirarea timpului maxim de inactivitate pe care servletul l-a specificat pentru sesiunea respectivă cu metoda **setMaxInactiveInterval**.

```
...
if (numar==0) {
    session.invalidate();
    context.log("sesiune invalidata");
    //raspundem clientului ca s-a terminat sesiunea
    out.println("<h1> Sesiune terminata </h1>");
    out.println("</html>");
    out.close();
    return;
}
...
```

Cookie-uri dezactivate

Implicit, obiectele de tip `HttpSession` folosesc cookie-uri pentru păstrarea sesiunii.

Pentru a fi utilizabil în toate situațiile un servlet trebuie să:

- **genereze dinamic formularele**
- **rescrie URL-ul** de la atributul `ACTION`, dacă este necesar.

Rescrierea URL-ului unui servlet

```
...
String urlServlet="http://localhost:8080/servlet/MyServlet";
if (<Apel pentru formular>) {
    //Generam formularul
    out.println("<HTML><FORM METHOD=GET ACTION=" +
        response.encodeUrl(urlServlet)} + " >");
    ...
    out.println("</FORM>"</HTML>);
    out.close();
    return;
}
//S-au primit datele din formular
//Construim dinamic raspunsul propriu-zis
...
http://localhost:8080/servlet/MyServlet
;jsessionId=935244D5D07441895052477146C371D3
?param1 = val1& param2 = val2 ...
```

Folosirea explicită a cookie-urilor

Cookie-uri



Cookie-urile reprezintă o modalitate pentru o aplicație ce rulează la nivelul serverului să trimită o informație clientului care să fie memorată local pe mașina acestuia. Un cookie este compus din trei șiruri:

(nume, valoare, comentarii),
și este modelat de către clasa **Cookie**.



Salvarea / Restaurarea cookie-urilor

Salvarea

- Instanțierea unui obiect de tip `Cookie`
- Setarea atributelor
- Trimiterea cookie-ului prin intermediul obiectului `HttpServletResponse`

Restaurarea

- Obținerea tuturor cookie-urilor primite odată cu cererea clientului
- Găsirea cookie-ului cu numele căutat
- Extragerea valorii din cookie-ul găsit

Exemplu de utilizare

Să rescriem exemplul cu suma numerelor primite:

```
//Preluarea vechii valori
suma = 0;
Cookie theCookie=null;
Cookie cookies[] = request.getCookies();
if (cookies != null) {
    for (int i=0; i<cookies.length; i++) {
        theCookie = cookies[i];
        if (theCookie.getName().equals("CookieServlet.suma")) {
            suma = Double.parseDouble(theCookie.getValue());
            break;
        }
    }
}
...
//Salvam suma actualizata
theCookie = new Cookie("CookieServlet.suma", Double.toString(suma));
theCookie.setComment("suma numerelor primite");
response.setContentType("text/html");
response.addCookie(theCookie);
...
```

Durata de viață a unui cookie

Implicit, durata de viață a unui cookie coincide cu durata sesiunii de lucru. Explicit, ea poate fi controlată prin metoda **setMaxAge** ce poate avea ca argument:

- > 0 : durata de viață în secunde
- 0 : șterge cookie-ul de pe mașina clientului
- < 0 : situația implicită

```
if (numar==0)
    theCookie.setMaxAge(0);
else
    theCookie.setMaxAge(5);
```