

# CGI vs servlet (II)

## Procesarea datelor folosind metoda POST și mecanismul SSI

– Sabin Corneliu Buraga, Ștefan Andrei

În precedentul articol am prezentat câteva aspecte generale referitoare la modalitățile de concepere a script-urilor CGI în limbajul C și a servlet-urile Java, insistând asupra funcționalității și a prelucrării datelor introduse prin intermediul formularelor Web. În continuare, vom ilustra procesarea datelor folosind metoda POST și mecanismul SSI, plus vom formula o serie de remarci pe care trebuie să le ia în considerație programatorii de aplicații Web.

### CGI: Procesarea unui formular prin metoda POST

Să presupunem că avem situația următoare: este necesar ca prin intermediul unui formular să preluăm de la utilizator o linie de text care va fi trimisă unui script CGI să o adauge la sfârșitul unui fișier stocat pe server. Acest fișier va putea fi modificat doar de către autorul formularului și de către script și va putea fi citit de către oricine altcineva prin intermediul unui alt script.

Dacă în exemplul anterior realizăm numai o simplă prelucrare care nu afectă în nici un fel starea serverului (nu se modifică nici un fișier de pe server) și cererea de aflare a sumei celor două numere putea fi trimisă de oricâte ori doream fără a genera probleme, în acest caz prin intermediul scriptului se va schimba conținutul fișierului vizat, deci și starea serverului. Pentru aceasta vom folosi metoda POST. Ca și la metoda GET, un program CGI are acces la întreg mediul sistemului de operare pe care rulează. Metoda POST este utilă în situațiile în care avem de trimis script-ului CGI spre prelucrare un volum mai mare de date (de exemplu, conținutul unei scrisori introdus prin intermediul unui <textarea> ori al unui fișier, prin acțiunea de *upload*) sau informații confidențiale (e.g. parole) care nu trebuie să apară în componența URI-ului transmis serverului Web.

Pentru formularele utilizând metoda POST, datele trimise scriptului vor putea fi accesate de la intrarea standard (*stdin*), iar lungimea, în octeți, a datelor trimise va fi disponibilă în variabila de mediu CONTENT\_LENGTH. Vor trebui citite atâtea caractere câte indică variabila de mediu CONTENT\_LENGTH, nu mai multe.

Formularul prin care se va solicita linia care va fi adăugată la fișier este:

```
<form action="http://www.infoiasi.ro/cgi-bin/addproj.cgi"
  method="POST">
  <p>Introduceți titlul proiectului:</p>
  <input name="data" size="50" maxlength="80" />
  <input type="submit" value="Trimite" />
</form>
```

Programul C care acceptă datele și le adaugă la fișierul *proj.txt* va fi următorul cea din caseta „Acceptare și adăugare de date”.

Propunem cititorului să conceapă, în limbajul C, scriptul CGI care să realizeze vizualizarea informațiilor despre proiecte, prin consultarea conținutului fișierului *proj.txt*.

### Server Side Includes (SSI)

**Server Side Includes** oferă posibilitatea execuției de programe CGI și de efectuare a altor acțiuni din cadrul paginilor Web prin intermediul unor comenzi incluse direct în codul HTML, fără a necesita prezența unui formular pentru activarea scriptului CGI dorit. Fiecare comandă SSI va putea fi introdusă printr-un comentariu special `<!--#command ->`, unde *command* este o comandă suportată de specificația SSI. Din moment ce este inclusă ca un comentariu HTML, navigatorul Web o va ignora, dar va fi procesată pe partea server și comentariul va fi înlocuit cu ieșirea acelei comenzi. Fișierele HTML utilizând comenzi SSI vor avea extensia *.shtml* în loc de *.html*, dar această convenție nu este obligatorie.

În continuare vom prezenta pe scurt numai comanda SSI *exec*, pentru mai multe amănunte cititorul având la dispoziție lucrarea [2] din bibliografie.

Probabil cea mai folosită comandă SSI, comanda *exec* va invoca un program CGI pe partea de server, iar rezultatul va fi plasat în locul invocării lui *exec*. Programul CGI va trebui să aibă setate permisiunile de execuție.

De exemplu, având un program CGI care alege aleatoriu un citat celebru pentru a fi afișat la fiecare încărcare a unei pagini Web, pentru a-l invoca va trebui să inserăm în cadrul documentului HTML următoarele (în care *citare.cgi* este scriptul CGI conceput în C care afișează la ieșirea standard un citat ales dintr-o serie de citate stocate pe server – pentru exemplificare, vezi [www.cs.tuiasi.ro](http://www.cs.tuiasi.ro)):

```
<html>
<head><title>Pagina ta</title></head>
<body>
<h5 align="right">
<i>Citatul zilei:</i><br />
<!--#exec cgi="citare.cgi" ->
</h5>
<hr />
...
</body>
</html>
```

### Java: Procesarea unui formular prin metoda POST

Continuăm cu prezentarea metodei *doPost()* care are prototipul:

```
protected void doPost(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, java.io.IOException;
```

Metoda *doPost()* este apelată de fiecare dată când apare o cerere HTTP de tip POST primită de la servlet. După cum am văzut, această

## Acceptare și adăugare de date

```
#include <stdio.h>
#include <stdlib.h>

#define MAXLEN 80
#define EXTRA 5
/* 4 pentru numele câmpului "data", 1 pentru "=" */
#define MAXINPUT MAXLEN+EXTRA+2
/* 1 pentru sfârșit de linie, 1 pentru NULL */

/* numele fișierului (cale relativă) */
#define DATAFILE "proj.txt"

/* funcție de decodificare a datelor pasate programului CGI */
void unencode(char *src, char *last, char *dest)
{
    for (; src != last; src++, dest++)
        if (*src == '+') /* înlocuiește "+" cu spațiu */
            *dest = ' ';
        else
            if (*src == '%')
                { /* înlocuiește hexa cu caracterul */
                    int code;
                    if (sscanf(src + 1, "%2x", &code) != 1)
                        code = '?';
                    *dest = code;
                    src +=2;
                }
            else /* copie celelalte caractere... */
                *dest = *src;
        *dest = '\n';
        *++dest = '\0';
}

/* funcția principală */
int main(void)
{
    char *lenstr;
    char input[MAXINPUT], data[MAXINPUT];
    long len;

    /* scrie Content-type, urmat de setul de caractere (opțional),
    în acest caz ISO-8859-2 (folosim diacritice) */
    printf("%s%c%c\n",
        "Content-type: text/html;charset=iso-8859-2", 13, 10);

    /* scrie antetul documentului HTML generat */
    printf("<html>\n<head><title>Răspuns</title></head>\n");

    /* ia numărul de caractere transmise */
    lenstr = getenv("CONTENT_LENGTH");

    /* numărul de caractere este corect? */
    if (lenstr == NULL ||
        sscanf(lenstr, "%ld", &len) != 1 ||
        len > MAXLEN)
        printf("<p>Eroare - formular incorect?\n");
    else
        {
            FILE *f;
            /* citește de la intrarea standard datele */
            fgets(input, len + 1, stdin);
            /* decodifică linia */
            unencode(input + EXTRA, input + len, data);
            /* adaugă în fișier */
            f = fopen(DATAFILE, "a");
            if (f == NULL)
                printf("<p>Eroare la adăugarea în fișier...");
            else
                fputs(data, f);
            fclose(f);
            printf("<p>Mulțumim! Linia <b>%s</b> a fost adăugată în
            fișier\n",
                data);
        }

    /* sfârșitul documentului */
    printf("</body>\n</html>\n");
    return 0;
}
```

metodă este de obicei folosită să proceseze informația colectată dintr-un formular HTML. Informația introdusă de utilizator într-un formular HTML este încapsulată într-un obiect `HttpServletRequest` și trimis metodei `doPost()`. Dacă nu se găsește o implementare a metodei `doPost()`, atunci se trimite către client un mesaj de eroare *HTTP 400 Bad Request*.

Pentru a vedea funcționalitatea unui servlet, vom prezenta două exemple de utilizare a metodelor `doGet()` și `doPost()`.

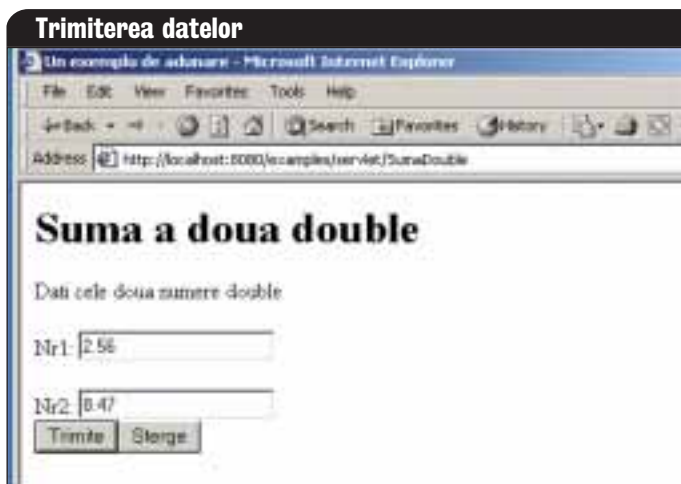
**Exemplu:** Presupunem că am scris un fișier HTML cu textul:

```
<html><head>
<title>Un exemplu de adunare</title>
</head>
<body>
<h1>Suma a doua double</h1>
<p>Dati cele doua numere double</p>
<form method="GET" action="http://localhost:8080/examples/
    servlet/SumaDoubleGet">
<p>Nr1: <input type="text" name="nr1" /></p>
<p>Nr2: <input type="text" name="nr2" /></p>
<input type="submit" value="Trimite" />
<input type="reset" value="Sterge" />
```

```
</form></body>
</html>
```

Se observă că acest fișier XHTML utilizează metoda HTTP de tip GET. De exemplu, sub Windows, folosind serverul *Tomcat*, plasăm în directorul `C:\Tomcat\jakarta-tomcat\webapps\examples\Web-inf\classes\` fișierul `.class` asociat codului sursă Java de mai jos:

```
import javax.servlet.http.*;
import javax.servlet.ServletException;
import java.io.*;
public class SumaDoubleGet extends HttpServlet
{ // procesam un formular HTML pentru citirea a doua numere
    // de tip double si furnizam ca raspuns o pagina HTML
    protected void doGet(HttpServletRequest cerere,
        HttpServletResponse raspuns) throws ServletException,
        IOException
    {
        String numarUnu = cerere.getParameter("nr1");
        String numarDoi = cerere.getParameter("nr2");
        double nr1 = 0.0, nr2 = 0.0;
        Double d1Temp = Double.valueOf(numarUnu);
```



```
nr1 = d1Temp.doubleValue();
d1Temp = Double.valueOf(numarDoi);
nr2 = d1Temp.doubleValue();
double sumaDouble = nr1 + nr2;

PrintWriter iesire = raspuns.getWriter();
iesire.println("<html><head>");
iesire.println("<title>Un exemplu de adunare</title>");
iesire.println("</head>");
iesire.println("<body>");
iesire.println("<h1>Suma celor doua double</h1>");
iesire.println("<p>Suma celor doua numere double este "
+ sumaDouble + "</p>");
iesire.println("</body></html>");
iesire.close(); // inchidem fluxul de iesire
}
```

Astfel apelând documentul de mai sus, după completarea câmpurilor și apăsarea butonului *Submit*, acesta va apela servletul de mai sus apelând metoda `doGet()` care va trimite către client o pagină Web care va afișa, în principiu, suma celor două numere *double* citite în formular.

În cele ce urmează, vom prezenta o variantă mai elegantă de a face suma a două numere. Astfel vom vedea o funcționalitate mai mare a servletului folosind ambele funcții `doGet()` și `doPost()`.

**Exemplu:**

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



```
public class SumaDouble extends HttpServlet
{
// returnam un formular HTML pentru citirea a doua numere
// de tip double
protected void doGet(HttpServletRequest cerere,
HttpServletResponse raspuns)
throws ServletException, IOException
{
// setam tipul MIME pentru antetul HTTP
raspuns.setContentType("text/html");
PrintWriter iesire = raspuns.getWriter();
iesire.println("<html><head>");
iesire.println("<title>Un exemplu de adunare</title>");
iesire.println("</head>");
iesire.println("<body>");
iesire.println("<h1>Suma a doua double</h1>");
iesire.println("<p>Dati cele doua numere double</p>");
iesire.println("<form method="POST" action=" +
"\http://localhost:8080/examples/servlet/
SumaDouble"");
iesire.println("<p>Nr1: <input type="text"
name="nr1" /></p>");
iesire.println("<p>Nr2: <input type="text"
name="nr2" /></p>");
iesire.println("<input type="submit" value="Trimite"
/>" +
"<input type="reset" value="Sterge" />");
iesire.println("</form></body></html>");
iesire.close(); // inchidem fluxul de iesire
}

// procesam datele primite din formularul XHTML si returnam
// un document XHTML care afiseaza suma celor doua numere
protected void doPost(HttpServletRequest cerere,
HttpServletResponse raspuns)
throws IOException
{
// setam tipul MIME pentru antetul HTTP
raspuns.setContentType("text/html");
String numarUnu = cerere.getParameter("nr1");
String numarDoi = cerere.getParameter("nr2");
double nr1 = 0.0, nr2 = 0.0;
Double d1Temp = Double.valueOf(numarUnu);
nr1 = d1Temp.doubleValue();
d1Temp = Double.valueOf(numarDoi);
nr2 = d1Temp.doubleValue();
double sumaDouble = nr1 + nr2;
PrintWriter iesire = raspuns.getWriter();
iesire.println("<html><head>");
iesire.println("<title>Un exemplu de adunare</title>");
iesire.println("</head>");
iesire.println("<body>");
iesire.println("<h1>Suma celor doua double</h1>");
iesire.println("<p>Suma celor doua numere double este "
+ sumaDouble + "</p>");
iesire.println("</body></html>");
iesire.close(); // inchidem fluxul de iesire
}
} // sfarsitul definitiei clasei SumaDouble
```

În exemplul de mai sus, metoda `doGet()` este folosită pentru a crea un formular simplu HTML. Când se apasă butonul *Submit*, datele formularului sunt trimise către servlet și procesate de metoda `doPost()`. Metoda `doPost()` construiește un document HTML care conține suma

celor două numere *double* trimise și apoi le returnează clientului. De altfel, de obicei când se dorește citirea de informații de la client, se procedează în acest fel. Adică folosind metoda `doGet()` se creează un formular HTML și apoi se prelucrează datele primite returnându-se o pagină Web folosind metoda `doPost()`.

Execuția servletului poate fi împărțită în două etape, trimiterea datelor și primirea rezultatului (vezi figurile „Trimiterea datelor” și „Primirea rezultatului”).

## Concluzii

O serie de aspecte de care trebuie să ținem seama:

- **performanța.** Servleturile rulează de obicei în același spațiu de proces ca și serverul și sunt încărcate doar o dată. Astfel, servleturile sunt capabile să răspundă mai rapid și eficient la cererile clienților. În contrast, CGI-urile trebuie să creeze câte un nou proces pentru deservirea unei cereri. Cu toate acestea, CGI-urile scrise în limbaje compilate pot fi (mult) mai rapide.
- **compilarea in byte coduri Java** oferă execuții rapide față de programele scrise în limbaje script. Multe din erori sunt îndepărtate încă de la compilare, deci servleturile sunt mai stabile. Același lucru se observă și la CGI-urile scrise în Perl sau alte limbaje interpretate.
- **rezistente la blocarea sistemului.** Mașina virtuală Java nu permite servleturilor accesul direct la locațiile de memorie, deci se elimină posibilitatea blocării sistemului (serverului Web) ca rezultat al accesului la memoria invalidă (pointerii în C pot conduce la acest lucru). JVM va propaga o excepție sau va rezolva eroarea fără blocarea sistemului.
- **portabilitatea platformei.** Caracteristica Java „scris o dată, rulează oriunde” permite servleturilor să fie distribuite ușor fără a rescrie codul pentru fiecare platformă. Servleturile operează identic fără modificări când rulează pe Unix, Windows sau alt sistem de operare. Acest lucru este valabil și pentru CGI-urile scrise în limbaje precum Perl, Python sau chiar C (dacă nu folosim apeluri de sistem specifice unei platforme particulare).
- **portabilitatea serverelor.** Ne referim la proprietatea rulării servleturilor pe orice server Web. La adresa <http://java.sun.com/products/servlet/runners.html> există o listă completă cu serverele care furnizează suport nativ pentru servleturi în produsele lor, cum ar fi: *Apache Tomcat*, *Inprise Application Server*, *Oracle Application Server* și altele. Serverele *JRun* (firma Allaire) și *ServletExec* (firma New Atlanta Communications) adaugă suport pentru multe servere Web, printre care se numără *Apache Web Server*, *Microsoft Internet Information Server* sau *Netscape Enterprise Server*.
- **durabilitate.** Servleturile rămân în memorie până când există o instrucțiune specifică de distrugere a lor. Astfel, servleturile necesită doar o instanțiere pentru a satisface mai multe cereri. De exemplu, se obișnuiește ca la încărcarea unui servlet să se creeze și mai multe conexiuni la baze de date.
- **încarcare dinamică.** Ca și appleturile, servleturile pot fi dinamic încărcate local sau prin rețea. Încărcarea dinamică asigură servleturilor nefolosite să nu consume resurse prețioase. Ele sunt încărcate doar când este nevoie.
- **extensibilitatea.** Noile instrumente de dezvoltare, noile biblioteci de clase Java și driverele de baze de date sunt constant disponibile și astfel pot fi utilizate de servleturi. Același lucru se poate spune și despre CGI-urile scrise în Perl ori chiar în C.
- **concuranța.** Spre deosebire de C/C++, mecanismele de concurență (*thread*) sunt oferite nativ în Java și sintaxa sincronizării *thread*-urilor este consistentă pe toate platformele. Natura concurențială a servleturilor Java permit rezolvarea cererilor clienților în fire de execuție separate într-un singur proces.
- **orientarea obiect.** Servleturile furnizează o arhitectură simplă și elegantă pentru dezvoltarea aplicațiilor de rețea. Asta deoarece Servlet API încapsulează toate informațiile esențiale și funcționalitatea în obiecte bine construite. De exemplu, Servlet API furnizează clase care

lucrează cu obiecte abstracte, cum ar fi: cereri, răspunsuri, sesiuni și „cookies” (prăjituri).

- **independența de protocol.** De obicei, servleturile sunt folosite pentru extinderea funcționalității serverelor HTTP. Servleturile sunt complet independente de protocol, acestea suportând comenzi FTP, SMTP, POP3, sesiuni Telnet, grupuri de știri NNTP sau orice alt protocol (fie unul standard sau creat de programator).
- **securitate.** Deoarece servleturile sunt scrise în Java, nu sunt posibile accesul nevalide la memorie sau violări de tip. În plus, există trei proprietăți care fac servleturile mai sigure decât CGI-urile. În primul rând, servleturile folosesc un manager de securitate pentru crearea unor reguli de securitate. Un manager de securitate poate restricționa rețeaua sau accesul la un fișier pentru un servlet în care nu are încredere. Pentru CGI-uri, acest lucru se realizează mai ușor apelând la restricțiile oferite de unele servere Web (de exemplu, pentru Apache se pot utiliza fișierele `.htaccess`). Pe de altă parte, un manager de securitate poate acorda drepturi depline pentru un servlet de încredere. În al treilea rând, un servlet are acces la toate informațiile conținute în fiecare cerere a clientului. Această informație conține date de autentificare HTTP. Când se folosesc în conjuncție cu protocoalele de securitate cum ar fi SSL, servleturile pot verifica identitatea fiecărui client. Dezavantajul este că odată obținute fișierele cod binar Java (`.class`), acestea se pot decompila, ceea ce este mult mai dificil la CGI-urile scrise în C/C++.

În continuare, propunem cititorilor interesați, rezolvarea și implementarea următoarelor exerciții/proiecte:

1. (**mersul trenurilor**) Presupunem că dispunem de o bază de date (sub forma unui fișier binar/text, bază de date relațională, fișier XML etc.) care cuprinde mersul trenurilor din România. Scrieți un script CGI și un servlet care să modeleze completarea datelor despre toate rutele dintre două orașe (oraș plecare, oraș destinație, număr maxim de schimbări, ruta - opțional) din România și care să furnizeze o pagină Web care listează toate posibilitățile existente cu informațiile aferente (timp, rută, schimbare, tip tren) sortate după numărul de ore și minute anticipate! Atenție: se va ține cont și de faptul că unele trenuri nu circulă sâmbătă/duminică.
2. (**webopt**) Să se conceapă o aplicație Web permițând înscrierea *online* a studenților la cursurile opționale, pe ani de studii și module propuse.
3. (**student**) Un manager Web a situației notelor și prezenței studenților la seminariile/laboratoarele susținute de un anumit cadru didactic. Această aplicație va putea genera, pe baza interogărilor, documente HTML cu situația parțială/finală a unor studenți sau cu liste conform unor criterii dorite de utilizator. Editarea se va realiza pe baza autentificării.

Autorii sunt cadre didactice la Facultatea de Informatică, Universitatea „Al.I. Cuza” din Iași și pot fi contactați prin email la adresele [busaco@infoiasi.ro](mailto:busaco@infoiasi.ro) și [stefan@infoiasi.ro](mailto:stefan@infoiasi.ro), respectiv. ■ 98

## Bibliografie:

- [1] Budd, T., *Understanding Object-Oriented Programming with Java*, Addison-Wesley (2000)
- [2] Buraga, S. C., *Tehnologii Web*, Editura Matrix Rom, București (2001): <http://www.infoiasi.ro/~busaco/books/web.html>
- [3] Buraga, S. C., Ciobanu, G., *Atelier de programare în rețele de calculatoare*, Editura Polirom, Iași (2001)
- [4] Callaway, D. R., *Inside Servlets. Server-Side Programming for the Java™ Platform*, Addison Wesley (2001)
- [5] Hughes, M., Shoffner, M., Hamner, D., Bellur, U., *Java. Network Programming*, Manning (1999)
- [6] Morgenthal, J.P., La Forge, B., *Enterprise Application Integration with XML and Java*, Prentice Hall PTR (2001)