Laboratorul 07

Realizarea de aplicații web folosind Java Servlets

Aplicaţii Integrate pentru Întreprinderi (AIPI)
Semestrul de Toamnă 2014
Departamentul de Calculatoare

Conținut

- **Implementarea aplicaţiilor web folosind Java Enterprise Edition**
- **Serverul HTTP Apache Tomcat 8.x**
- **Tehnologia Java Servlets – aspecte generale**
  - **Ciclul de viaţă al unui Java Servlet**
  - **Structura unui Java Servlet**
- **Implementarea unor funcționalități complexe**
- **Interfaţarea Java Servlets cu un sistem de gestiune pentru baze de date**
- **Mecanisme pentru gestiunea stării în Java Servlets**

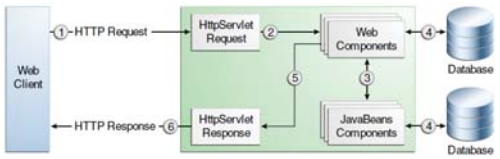
Implementarea aplicaţiilor web în Java Enterprise Edition

- aplicații web
  - extensie dinamică a unui server (web, de aplicații)
  - transferă cerințele cu privire la resurse (programe instalate) serverului pe care sunt găzduite, funcționalitatea oferită fiind accesibilă printr-un client universal (browser-ul)
  - clasificare
    - ❶ **orientate pe prezentare** – pagini Internet interactive descrise folosind limbaje de adnotare (HTML, XML) având conținut dinamic generat ca răspuns la cererile transmise de utilizator
      - ex: Facelets, Java Server Faces
    - ❷ **orientate pe servicii** – implementează funcționalitatea unui serviciu web
      - ex: Java Servlets (date nontextuale, gestiunea controlului altor tipuri de aplicații web)
    - orientate pe prezentare ← orientate pe servicii

Implementarea aplicaţiilor web în Java Enterprise Edition (cont'd)

- aplicaţii web
  - **componente**
    - Java Servlets
    - pagini Internet: JSF (JavaServer Faces), JSP (Java Server Pages)
    - servicii web (JAX-WS, JAX-RS)
    - + servicii container (gestiune cereri, concurenţă, securitate, gestiunea ciclului de viaţă)
  - resurse statice (imagini, foi de stil)
  - clase ajutătoare
  - biblioteci

Implementarea aplicaţiilor web în Java Enterprise Edition (cont'd)



Sursa: *The Java EE 7 Tutorial, Release 7 for Java EE Platform*, August 2014

Implementarea aplicaţiilor web în Java Enterprise Edition (cont'd)

- ❶ cererile HTTP ale clienţilor transmise de browser sunt transformate de serverul web într-un obiect `HttpServletRequest`
- ❷ cererea `HttpServletRequest` este transmisă unei componente web
- pentru a genera conţinut dinamic, componenta web poate interacţiona
  - ❸ cu o clasă Java Beans
  - ❹ cu o bază de date
- ❺ răspunsul este convertit într-un obiect `HttpServletResponse`
- ❻ serverul web transformă obiectul `HttpServletResponse` într-un răspuns HTTP care va putea fi vizualizat în browser

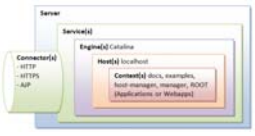
Implementarea aplicaţiilor web în Java Enterprise Edition (cont'd)

- configurarea aplicaţiilor web
  - aspecte ale comportamentului acestora în diferite situaţii
  - sunt încărcate la instalarea (*eng.* deploy) aplicaţiei web în contextul containerului
  - mecanisme
    - ❶ adnotări Java
    - ❷ fişiere XML (descriptorul de instalare al aplicaţiei web)
      - !!! trebuie să respecte schemele specificaţiei Java Servlet

Serverul HTTP Apache Tomcat 8.x

- server HTTP
  - container pentru Java Servlets
  - arhitectură modulară
- primeşte cereri şi generează răspunsuri (documente HTML) din cadrul claselor Java Servlet
  - paginile JSP / JSF – transformate automat la clasele Java Servlets asociate atunci când sunt accesate
- versiunea stabilă: 8.0.15
  - implementează specificaţiile
    - Java Servlets 3.1
    - JavaServer Pages 2.3
    - Expression Language 3.0
    - WebSocket 1.1
  - aplicaţiile web trebuie scrie folosind minim Java 7

Serverul HTTP Apache Tomcat 8.x Arhitectura



Sursa: http://www3.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_More.html

- serviciu → unul sau mai multi conectori la motorul serverului care poate rula pe mai multe maşini
  - HTTP / 1.1
    - comunicaţia client-server
    - protocolul HTTP (portul 8080)
  - AJP / 1.3
    - comunicaţia dintre serverul Tomcat şi serverul Apache HTTP
    - protocolul AJP (Apache JServ Protocol) – portul 8009
- container < motor < gazdă < context
  - relaţiile au multiplicitatea 1:n
  - motor: Catalina, gazda: localhost

Serverul HTTP Apache Tomcat 8.x Structura ierarhiei de directoare (1)

- ❶ `bin` – script-uri apelate la pornirea și oprirea serverului web
  - `startup`, `shutdown` [`.bat`|`.sh`] – lansarea în execuție / oprirea serverului
  - `setClasspath` [`.bat`|`.sh`] – `JAVA_HOME`, `JRE_HOME`
- ❷ `conf` – fișiere de configurare aplicabile tuturor aplicațiilor din contextul serverului web
  - `catalina.policy`
  - `catalina.properties`, `logging.properties`
  - `server.xml`, `web.xml`, `content.xml`, `tomcat-users.xml`
  - câte un director pentru fiecare motor conținând subdirectoare pentru toate gazdele
- ❸ `lib` – biblioteci comune, folosite de toate aplicațiile
  - `servlet-api.jar` – Java Servlet
  - `jasper.jar` / `jasper-el.jar` – Java Server Pages / EL

Serverul HTTP Apache Tomcat 8.x Structura ierarhiei de directoare (2)

- ❹ `logs` – jurnale specifice
  - motorul Catalina
  - gazdele pe care le gestionează
  - aplicațiile `manager` / `host-manager`
  - jurnalul de acces
- ❺ `webapps` – locația în care vor fi plasate aplicațiile, fiind accesate de serverul web Apache Tomcat
- ❻ `work` – director de lucru ce conține clasele Java Servlet corespunzătoare documentelor JSF / JSP
  - organizarea se face pe motor, gazde configurate în cadrul motorului, aplicații
- ❼ `temp` – diferite resurse temporare

Serverul HTTP Apache Tomcat 8.x Configurare (1)

- `bin/setClasspath`[`.bat`|`.sh`] – variabilele de mediu `JAVA_HOME`, `JRE_HOME`
  - Windows

```
set JAVA_HOME=C:\Program Files\Java\jdk1.8.0_25\
set JRE_HOME=C:\Program Files\Java\jdk1.8.0_25\jre\
```

  - Linux

```
export JAVA_HOME=/usr/lib/jvm/default-java/
export JRE_HOME=/usr/lib/jvm/default-java/jre/
```

- `conf/server.xml`

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
```

  - `webapps` – director de bază unde vor fi dezvoltate aplicațiile
  - `unpackWARs` – dezarhivarea aplicațiilor dezvoltate ca `.war` (Web Archive)
  - `autoDeploy` – instalarea automată a aplicațiilor dupa plasarea acestora în directorul corespunzător

Serverul HTTP Apache Tomcat 8.x Configurare (2)

- configurarea utilizatorilor (`conf/tomcat_users.xml`)
  - implicit, nu conține nici un fel de roluri / utilizatori
  - un utilizator nu trebuie să aibă drepturi corespunzătoare mai multor roluri

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="manager-gui" />
    <role rolename="manager-status" />
    <role rolename="manager-script" />
    <role rolename="manager-jmx" />
    <role rolename="admin-gui" />
    <role rolename="admin-script" />
    <user username="admin" password="admin" roles="manager-gui, admin-gui,
manager-status, manager-script, admin-script, manager-jmx" />
</tomcat-users>
```

Serverul HTTP Apache Tomcat 8.x Configurare (3)

- tipuri de utilizatori
  - `manager-gui` – interfața grafică cu utilizatorul
    - http://localhost:8080/manager/html sau Manager App
  - `manager-status` – informații despre starea serverului
    - http://localhost:8080/manager/server sau Server Status
  - `manager-script` – interfața în mod text prin care pot fi transmise comenzi prin parametrii din URL (solicitat de unele medii integrat de dezvoltare)
    - http://localhost:8080/manager/text/{comanda}?{parametri}
      - comanda: `list` (afisare lista aplicații), `deploy` (dezvoltare aplicații)
      - parametri – contextul aplicației pentru care se dă comanda
  - `manager-jmx` – acces la interfața JMX
    - http://localhost:8080/manager/jmxproxy/?{comanda}={parametri}

Serverul HTTP Apache Tomcat 8.x Funcționalitate

- Server Status – informații despre starea serverului
- Manager App
  - informații cu privire la aplicațiile care au fost configurate în contextul serverului web (locație, versiune, stare curentă, numărul de sesiuni deschise)
  - operații: pornire, oprire, reîncărcare, configurare
  - configurare: perioada de timp după care sesiunile nu mai sunt active
- Host Manager – modulul de gestiune a gazdei

Serverul HTTP Apache Tomcat 8.x Dezvoltarea unei aplicații web

- `webapps` – locația / contextul de accesare a aplicației
  - `META-INF` – informații legate de server (`context.xml`)
  - `WEB-INF` – informații legate de aplicație care nu vor fi accesibile clienților
    - fișierul de configurare `web.xml`
      - `servlet`: asociere nume servlet (`servlet-name`) și clasa servlet (`servlet-class`); `load-on-startup` – ordinea la încărcare
      - `servlet-mapping` – contextul de unde poate fi accesat servletul (`url-pattern`)
      - `welcome-file-list` – pagina Internet încărcată la pornirea aplicației web (`welcome-file`)
    - sursele aplicației (**`src`** / **`sources`**) – nu există o denumire standard
    - clasele aplicației – obținute în urma compilării surselor (**`classes`**)
    - bibliotecile folosite de aplicație (**`lib`**)
      - încărcate manual, cu `Class.forName("...")`
  - alte resurse disponibile clienților (foi de stil, imagini, scripturi)

Serverul HTTP Apache Tomcat 8.x Fișierul `web.xml`

```
<web-app>
    <servlet>
        <description></description>
        <display-name>ServletName</display-name>
        <servlet-name>ServletName</servlet-name>
        <servlet-class>ServletClass</servlet-class>
     </servlet>
     <servlet-mapping>
        <servlet-name>ServletName</servlet-name>
        <url-pattern>/ServletClass</url-pattern>
     </servlet-mapping>
     <welcome-file-list>
        <welcome-file>ServletName</welcome-file>
     </welcome-file-list>
</web-app>
```

- servlet-ul va fi accesibil la http://localhost:8080/<application_directory>/ServletClass/
  - `application_directory` – subdirectorul din `webapps` de unde este accesată aplicația
  - `ServletClass` – servlet-ul corespunzător
- Java Servlet ↔ pagini Internet (mai multe într-o aplicație web)

Serverul HTTP Apache Tomcat 8.x Gestiunea serverului

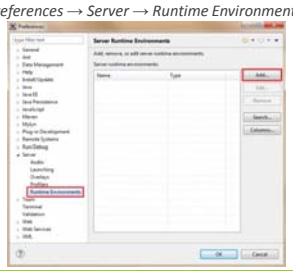
- pornire – `bin/startup[.bat|.sh]`
- oprire – `bin/shutdown[.bat|.sh]`
- dezvoltarea (*eng.* deployment) aplicațiilor web
  - odată cu lansarea în execuție a serverului web
  - sunt parcurse toate subdirectoarele din `webapps` astfel încât fiecare instanță a unei aplicații web este încărcată

```
Nov 24, 2014 12:00:00 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory BookStore
```

  - Java Servlets – necesară în cazul modificării claselor corespunzătoare unor pagini Internet
  - JavaServer Pages – transformarea paginilor Internet în clase Java Servlets corespunzătoare (+ compilarea acestora) realizată automat la solicitarea venită din cadrul browser-ului

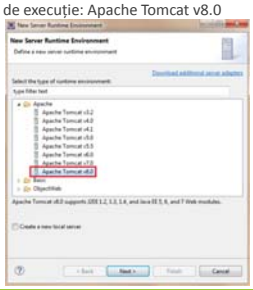
Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – Eclipse Luna (1)

1) specificarea unei referințe către serverul HTTP Apache Tomcat 8.x (*Window → Preferences → Server → Runtime Environments*)



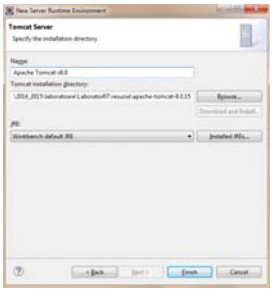
Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – Eclipse Luna (2)

2) tip al mediului de execuție: Apache Tomcat v8.0



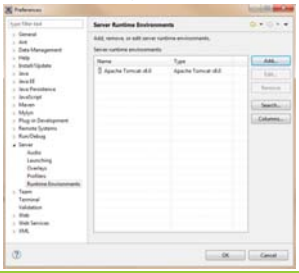
Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – Eclipse Luna (3)

3) configurare
- denumire server;
- locația în care este instalat;
- mediul de execuție Java cu care va rula.



Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – Eclipse Luna (4)

rezultat: secțiunea *Window → Preferences → Server → Runtime Environments*

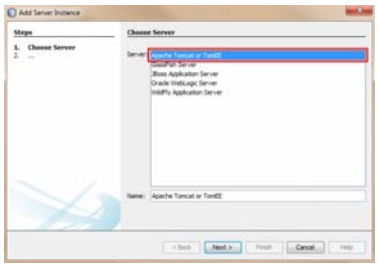


Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – NetBeans 8.0.1 (1)

1) specificarea unei referințe către serverul HTTP Apache Tomcat 8.x (*Services → Servers → click-dreapta → Add Server...*)

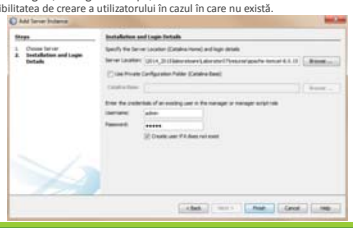
Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – NetBeans 8.0.1 (2)

2) tip al serverului: Apache Tomcat or TomEE



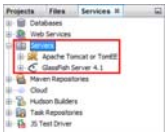
Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – NetBeans 8.0.1 (3)

3) configurare
- locația în care este instalat;
- date de autentificare (nume utilizator / parola) pentru publicarea aplicației web
  - rolul `manager` / `manager-script`;
  - posibilitatea de creare a utilizatorului în cazul în care nu există.



Serverul HTTP Apache Tomcat 8.x Interfațarea cu medii integrate de dezvoltare – NetBeans 8.0.1 (4)

rezultat: secțiunea *Services* → *Servers*
- starea: pornit / oprit
- aplicațiile web dezvoltate în contextul său

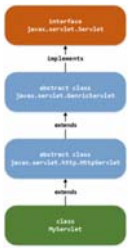


Tehnologia Java Servlets – aspecte generale

- permite dezvoltarea de aplicații web dinamice
- alternativă la CGI (Common Gateway Interface), eliminând
  - dependentă de platformă
  - scalabilitate redusă
- caracteristici
  - ❶ eficiență – initializarea se face o singură data, în metoda `init()`
  - ❷ persistența – obiectele unui servlet există atâta timp cât acesta se află în execuție
  - ❸ portabilitate
  - ❹ robustețe – acces la toate facilitățile oferite de limbajul de programare Java (ierarhie excepții, garbage collection)
  - ❺ extensibilitate – extinderea unui servlet se face potrivit modelului de programare orientat obiect
  - ❻ securitate – modelul de securitate Java

Tehnologia Java Servlets – aspecte generale (cont'd)

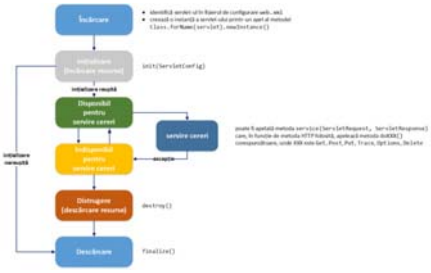
- clasă Java ce extinde capabilităţile unui server web
  - o implementeaza interfata `Servlet`
  - o tipuri
    - `GenericServlet – service()`
    - `HttpServlet – doMethod()` `(Method={Post,Get,Put,Trace,Options,Delete})`
  - o pachete
    - `javax.servlet`
    - `javax.servlet.http`
- aplicaţii conform modelului cerere-răspuns
- adaptată protocolului HTTP
- se poate mapa oricărui tip de protocol



Ciclul de viaţă al unui Java Servlet

- controlat de containerul în care a fost configurat servlet-ul
- operaţii la realizarea unei cereri asociate unui servlet
  - o ❶ dacă nu există o instanţă a servlet-ului
    - a) încărcarea clasei servlet
    - b) crearea unei instanţe a clasei servlet
    - c) iniţializarea instanţei clasei servlet prin apelarea metodei `init()`
  - o ❷ apelarea metodei `service()` care are ca parametrii obiectele cerere şi răspuns
  - o ❸ atunci când servlet-ul nu mai este necesar, se apelează metoda `destroy()`

Ciclul de viaţă al unui Java Servlet (cont'd)



Ciclul de viață al unui Java Servlet (cont'd)

- clase ascultător pentru monitorizarea evenimentelor corespunzătoare ciclului de viață al unui obiect Java Servlet
  - adnotate cu însemnarea `@WebServlet`
  - primesc ca argumente parametrii ce conțin informații despre obiectul respectiv

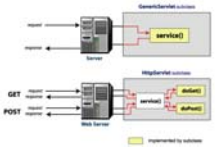
Obiect	Eveniment	Interfață (ce trebuie) Implementată	Obiect Eveniment
context web	creare, distrugere	`javax.servlet.ServletContextListener`	`ServletContextEvent`
	operații asupra atributelor (CRUD)	`javax.servlet.ServletContextAttributeListener`	`ServletContextAttributeEvent`
sesiune	creare, invalidare, activare, pasivizare, expirare	`javax.servlet.http.HttpSessionListener` `javax.servlet.http.HttpSessionActivationListener`	`HttpSessionEvent`
	operații asupra atributelor (CRUD)	`javax.servlet.http.HttpSessionAttributeListener`	`HttpSessionBindingEvent`
cerere	cererea pentru un obiect servlet este procesată de componentele web	`javax.servlet.ServletRequestListener`	`ServletRequestEvent`
	operații asupra atributelor (CRUD)	`javax.servlet.ServletRequestAtrributeListener`	`ServletRequestAttributeEvent`

Structura unui Java Servlet

- clasă derivată din `javax.servlet.http.HttpServlet`
- adnotată cu însemnarea `@WebServlet`
  - specifică cel puțin un URL
  - câmpurile `value` (1 atribut) / `urlPatterns` (mai multe atribute)
- suprascrie metodele
  - `init(ServletConfig)`
    - operații realizare o singură data, utile doar obiectului servlet din contextul căruia este apelat
      - încărcarea de informații persistente (date de configurare)
      - inițializarea resurselor
    - invocată ulterior încărcării / instanțierii servlet-ului, anterior acceptării invocărilor de la clienți
    - dacă nu este realizată cu succes, este generată o excepție `UnavailableException`
    - alternativă la atributul `initParams` al adnotării `@WebServlet`
      - conține însemnarea `@WebInitParams`

Structura unui Java Servlet (cont'd)

- suprascrie metodele
  - `service(ServletRequest, ServletResponse)`
    - implementează funcționalitatea servlet-ului
    - definită în clasa `GenericServlet`
    - clasa `HttpServlet` deleagă funcționalitatea ei în funcție de tipul de cerere primită
      - `HTTP GET` → `doGet()`
      - `HTTP POST` → `doPost()`
      - `HTTP PUT` → `doPut()`
      - `HTTP DELETE` → `doDelete()`
      - `doOptions()` / `doTrace()` – mai puțin utilizate
    - tratarea cererii
      - `getParameterNames()` – numele parametrului
      - `getParameter(String)` – valoarea parametrului
    - formularea răspunsului `getWriter()` → `PrintWriter`



Structura unui Java Servlet (cont'd)

- suprascrie metodele
  - `destroy()`
    - apelată atunci când servlet-ul este închis
    - toate resursele folosite de servlet trebuie eliberate
    - se asigură persistența prin reținerea informațiilor necesare din baza de date
    - !!! toate metodele serviciu asociate unui servlet trebuie terminate înainte de distrugerea sa
    - !!! toate firele de execuție ce deservesc clienți trebuie să fie terminate
      - contorizarea numărului de fire de execuție active
      - așteptarea terminării lor prin invocarea metodei `Thread.sleep()`
  - `getServletInfo()` – oferă informații despre servlet

Gestiunea cererilor prin Java Servlets

- implementează interfața `ServletRequest`
  - accesarea parametrilor transmiși de clienți prin intermediul unor formulare
  - obținerea valorilor unor obiecte folosite la comunicarea dintre un container al servlet-ului și servlet / între mai multi servleți
  - Informații despre protocolul prin care este transmisă cererea și despre locație
- `HttpServletRequest`
  - URL-ul cererii
    - http://[adresa]:[port]/[cale]?[interogare]
      - cale = cale contextuală + calea către servlet + alte informații
      - `getContextPath()`, `getServletPath()`, `getPathInfo()`
  - antetele HTTP
  - interogarea formată din perechi (nume, valoare) pentru parametrii transmiși
  - metode: `getContentType()`, `getCookies()`, `getHeaderNames()`, `getHeaders()`, `getSession()`, `getInputStream()`, `getMethod()`, **`getParameterNames()`**, **`getParameter()`**

Gestiunea răspunsurilor prin Java Servlets

- implementează interfața `ServletResponse`
  - conține datele transmise de la servlet către client
  - obținerea unui flux prin care se poate realiza comunicarea cu clientul
  - specificarea tipului de conținut: `setContentType("text/html")`
  - alocarea unei zone de memorie: `setBufferSize(int)`
    - datele nu sunt transmise imediat către client ci după completarea zonei de memorie
    - stabilirea unor coduri de stare / antete
- `HttpServletResponse`
  - construirea documentului care va fi transmis de la server către client într-un obiect `PrintWriter`
  - conține
    - antete HTTP / coduri de stare: indisponibilitatea resursei, redirectare
    - obiecte ce vor reține informații specifice aplicației (cookies)
  - metode
    - moștenite din `ServletResponse`: `flushBuffer()`, `get/setBufferSize()`, `get/setContentType()`, `getOutputStream()`, `setCharacterEncoding()`
    - proprii: `addCookie()`, `get/setHeader()`, `get/setStatus()`

Structura unui Java Servlet Exemplu

```
@WebServlet("/SampleServlet")
public class SampleServlet extends HttpServlet {
    final public static long    serialVersionUID = 1024L;
    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }
    @Override
    public void destroy() { }
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        ArrayList<String> values = new ArrayList<>();
        Enumeration parameters = request.getParameterNames();
        while(parameters.hasMoreElements()) {
            String parameter = (String)parameters.nextElement();
            if (parameter.contains("..."))
                values.add(request.getParameter(parameter));
        }
        response.setContentType("text/html");
        PrintWriter printWriter = new PrintWriter(response.getWriter());
        displayForm(printWriter);
        printWriter.close();
    }
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // ...
    }
}
```

Implementarea unor funcționalități complexe Filtre

- funcționalități
  - ❶ analiza unei cereri și acționarea în conformitate cu aceasta
  - ❷ blocarea unei cereri și a răspunsului corespunzător pentru a fi prelucrate mai departe
  - ❸ modificarea antetelor / conținutului obiectelor cerere / răspuns construind o versiune particularizată a acestora
  - ❹ interacțiunea cu resurse externe
- pot fi atașate mai multor resurse web de care nu sunt dependente
- utilizate la: autentificare, jurnalizare, conversie de imagini, compresie a datelor, criptare, parsarea fluxtului de date, transformări XML

Implementarea unor funcționalități complexe Filtre (cont'd)

- lanț de filtre = listă ce conține 0, 1 sau mai multe filtre executate într-o anumită ordine
- `javax.servlet` definește clasele
  - `Filter`
  - `FilterChain`
  - `FilterConfig`
- implementează interfața `javax.servlet.Filter`
- adnotat cu însemnarea `@WebFilter`
  - specifică un URL în `value` sau `urlPatterns`
  - informațiile cu privire la inițializare sunt conținute în `initParams`

Implementarea unor funcționalități complexe Filtre (cont'd)

- funcționalitatea este implementată în metoda `doFilter()`
  - analizează antetele cererii
  - particularizează obiectele cerere și răspuns modificând antetele sau conținutul
    - poate adăuga un atribut la cerere / introduce informații la răspuns
    - pentru a suprascrie metodele cererii / răspunsului, obiectele vor fi împachetate în obiecte derivate din `[Http]ServletRequestWrapper` / `[Http]ServletResponseWrapper`
  - invocă următorul filtru din lanțul de filtre (prin apelul metodei `doFilter()` a acestuia)
  - analizează antetele răspunsului
  - poate bloca comunicația mai departe
  - poate genera o excepție pentru a indica producerea unei erori
- asocieri între filtre și resurse web (prin nume / URL)
  - determină modul și ordinea în care sunt aplicate
  - jurnalizare: masca /* (se aplică tuturor evenimentelor referitoare la comunicația dintre client și server)
  - tipuri de asocieri
    - 1 filtru → mai multe resurse web
    - 1 resursă web → mai multe filtre

Implementarea unor funcționalități complexe Invocarea altor resurse web

- se obține un obiect `RequestDispatcher` folosindu-se metoda `getRequestDispatcher()` ce primește ca parametru URL-ul resursei ce va fi invocată
  - aplicată pe obiectul cerere – cale relativă
  - aplicată pe contextul web – cale absolută
  - dacă resursa nu este disponibilă sau serverul nu implementează un obiect `RequestDispatcher` pentru tipul respectiv de resursă, metoda intoarce `null`
- se realizează
  - direct
    - incluzând conținutul altei resurse
      - header, footer, informatii de copyright, meniuri
      - se folosește metoda **`include()`** (se execută componenta web – primind ca parametru cererea, afișându-se rezultatul)
    - transmițând mai departe cererea către o altă resursă
      - se folosește metoda **`forward()`**
      - URL-ul cererii se modifică la cel al paginii, acesta realizând prelucrarea ei și primind responsabilitatea răspunsului
      - nu este permisă daca au fost folosite obiecte `ServletOutputStream` sau `PrintWriter` în servlet (se generează `IllegalStateException`)
  - indirect – încorporând un URL către o alta resursă web în răspunsul transmis către client

Implementarea unor funcționalități complexe Invocarea altor resurse web (cont'd)

```
RequestDispatcher requestDispatcher = null;
switch(getUserRole(userName,userPassword)) {
    case Constants.USER_ADMINISTRATOR:
        requestDispatcher = getServletContext().getRequestDispatcher("/AdministratorServlet");
        break;
    case Constants.USER_CLIENT:
        requestDispatcher = getServletContext().getRequestDispatcher("/ClientServlet");
        break;
}
if (requestDispatcher!=null)
    requestDispatcher.forward(request,response);
```

- accesarea contexului web – `getServletContext()` → `ServletContext`
  - parametrii de inițializare
  - resurse asociate cu contextul web
  - atribute având asociate tipuri de obiecte
  - capabilități legate de jurnalizare

Implementarea unor funcționalități complexe Încărcarea de fișiere (cont'd)

- adnotarea `javax.annotation.MultipartConfig`
  - indică faptul că servletul pentru care este declarată poate prelucra cereri folosind tipul MIME `multipart/form-date`
  - componentele `javax.servlet.http.Part` pot fi obținute folosind metodele
    - `Collection<Part> getParts()` – pentru fișiere având tipuri diferite
    - `Part getPart(String)` – o parte identificată printr-un nume
    - analiza fiecărei părți: nume, dimensiune, tip de conținut, procesarea antetelor transmise împreună cu partea respectivă, salvarea pe disc, ștergerea ei
  - atribute
    - `location` – calea absolută către un director din sistemul de fișiere pentru a stoca fișiere temporare în timp ce părțile fișierului sunt procesate sau când dimensiunea fișierului depășește `fileSizeThreshold` / valoarea implicită = ""
    - `fileSizeThreshold` – dimensiunea fișierului (exprimată în octeți) după care acesta va fi stocat temporar pe disc / implicit = 0
    - `maxFileSize` – dimensiunea maximă pentru încărcarea de fișiere, exprimată în octeți / implicit = nelimitat
    - `maxRequestSize` – dimensiunea maximă pentru o cerere `multipart/form-date`, exprimată în octeți / implicit = nelimitat
    - pot fi specificate în `web.xml` în secțiunea `<multipart-config>`

Implementarea unor funcționalități complexe Procesare asincronă

- necesară în cazul în care unele fire de execuție sunt blocate așteptând anumite resurse, având un impact negativ asupra scalabilității aplicațiilor
- presupune crearea unui fir de execuție pentru fiecare operație blocantă
- `@WebServlet(asyncSupported=true)` – servletul are capacitatea de a realiza procesare asincronă
- `request.startAsync()` → `javax.servlet.AsyncContext` – cererea va fi transferată unui context asincron
  - răspunsul nu va fi transmis clientului la ieșirea din metoda `service()`
  - răspunsul trebuie generat în contextul asincron după terminarea operației blocante sau gestionat de alt servlet

Implementarea unor funcționalități complexe Procesare asincronă (cont'd)

- clasa `AsyncContext`
  - `void start(Runnable run)` – este creat un nou fir de execuție unde va fi procesată operația blocantă
    - codul ce tratează prelucrarea trebuie sa fie specificat în clasa ce implementează interfața `Runnable`
  - `ServletRequest getRequest()` – întoarce cererea folosită pentru a initializa contextul asincron (pentru a obține parametrii cererii în contextul asincron)
  - `ServletResponse getResponse()` – întoarce răspunsul folosit pentru a inițializa contextul asincron (pentru a construi răspunsul cu rezultatele operației blocante)
  - `void complete()` – termină operația asincronă și transmite răspunsul asociat contextului asincron
  - `void dispatch(String path)` – transmite obiectele cerere / răspuns către calea indicată (spre a delega altui servlet responsabilitatea cu privire la ele, după terminarea operației)

Implementarea unor funcționalități complexe Procesare asincronă (cont'd)

```
@WebServlet(urlPatterns={"/asyncServlet"}, asyncSupported=true)
public class AsyncServlet extends HttpServlet {
    @Override
    public void service(HttpServletRequest request, HttpServletResponse response) {
        response.setContentType("text/html");
        final AsyncContext asyncContext = request.startAsync();
        asyncContext.start(new Runnable() {
            public void run() {
                HttpServletRequest request = asyncContext.getRequest();
                String value = request.getParameter(attribute);
                HttpServletResponse response = asyncContext.getResponse();
                response.getWriter().println(blockingOperation(value));
                asyncContext.complete();
            }
        });
    }
}
```

Implementarea unor funcționalități complexe Operații de intrare/ieșire asincrone

- necesare atunci când operațiile de intrare / ieșire sunt mai rapide pe server decât pe client
- asociate de regulă cu procesarea asincronă pentru eliminarea timpilor morți din firele de execuție
- se folosesc clase ascultător care detectează momentul în care operațiile pot fi realizate non-blocant
- `javax.servlet.ServletInputStream` → `void setReadListener(ReadListener rl)` – asociază fluxului de intrare un obiect ascultător care conține metode pentru a citi date asincron
  - metode: `onDataAvailable()`, `onAllDataRead()`, `onError(Throwable)`
  - `isReady()` – datele pot fi citite non-blocant
  - `isFinished()` – toate datele au fost citite
- `javax.servlet.ServletOutputStream` → `void setWriteListener(WriteListener wl)` – asociază fluxului de ieșire un obiect ascultător care conține metode pentru a scrie date asincron
  - metode: `onWritePossible()`, `onError(Throwable)`
  - `isReady()` – datele pot fi scrise non-blocant

Interfațarea Java Servlets cu un sistem de gestiune pentru baze de date

- folosind metodele puse la dispoziție de API-ul Java DataBase Connectivity
- driver-ul de conectare la baza de date trebuie încărcat explicit înainte de a realiza accesul propriu-zis la informații
  - `Class.forName("com.mysql.jdbc.Driver")`
  - se apelează în mod automat `DriverManager.registerDriver()` care primește ca argument instanța creată, inclusă în lista de "drivere" pentru obținerea de conexiuni
- pot fi reutilizate metodele de acces la baza de date folosite în cazul aplicațiilor desktop
- clientul are acces la informațiile din baza de date fără a avea nevoie de nici un utilitar suplimentar, acestea găsindu-se pe server

Mecanisme pentru gestiunea stării folosind Java Servlets

- necesare datorită faptului ca HTTP este un protocol fără stare (spre diferență de FTP)
  - cererile și răspunsurile sunt tranzacții izolate
  - trebuie corelate accesările care provin de la același utilizator
- soluții
  - ❶ câmpurile ascunse
    - `<INPUT type="hidden" ... />` - conținute în formularele din paginile HTML
    - pot fi identificate cu ușurință
  - ❷ rescrierea URL-urilor
    - adăugarea informațiilor la URL-urile paginilor transmise utilizatorilor
    - se folosește împreună cu protocolul `HTTP GET`

Mecanisme pentru gestiunea stării folosind Java Servlets (cont'd)

- soluții
  - ❸ cookies
    - perechi (cheie, valoare) create pe server și transmise ca instrucțiuni clientului în antetul mesajului HTTP
    - `javax.servlet.http.Cookie`
      - constructor cu 2 parametri de tip String (cheie, valoare)
      - metode `set/get{Name/Value}()`
      - includerea în răspuns `response.addCookie(Cookie)`
      - preluarea din cerere `request.getCookies()` → `Cookie[]`

Mecanisme pentru gestiunea stării folosind Java Servlets (cont'd)

- soluții
  - ❹ sesiuni – identificarea conexiunii dintre client și server prin crearea unui obiect specific acesteia
    - `javax.servlet.http.HttpSession`
    - `request.getSession()` → `HttpSession`
    - atribute (având ca valori diferite obiecte) pot fi asociate sesiunii prin nume
      - obținerea atributelor: `getAttributeNames()`, `getAttribute(String)` – pe obiectul `request`
      - stabilirea atributelor: `setAttribute(String,Object)`, `removeAttribute(String)` – pe obiectul `response`
    - `javax.servlet.http.HttpSessionBindingListenerInterface` – asocierea / disocierea unui obiect cu o sesiune
    - `javax.servlet.http.HttpSessionActivationListener` – monitorizarea activării / pasivizarii sesiunii

Mecanisme pentru gestiunea stării folosind Java Servlets (cont'd)

- implementate prin cookie-uri / rescrierea URL-urilor
  - identificatorul conexiunii dintre server și client este reținut pe client (cookie) sau inclus în URL-ul transmis clientului
  - `encodeURL(URL)` – pe obiectul răspuns
    - rescrie URL-ul dacă clientul nu accepta cookie-uri
    - lasă URL-ul neschimbat dacă clientul acceptă cookie-uri
- expirarea sesiunii
  - `{set/get}MaxInactiveInterval()`
  - perioada după care sesiunea nu mai este necesară, resursele folosite de aceasta putând fi eliberate
  - timpul de expirare este reinițializat prin accesarea metodelor serviciu
  - `invalidate()` – apelată când interacțiunea cu un client este încheiată