Aplicatii Integrate pentru Intreprinderi
Semestrul de Toamna 2013

Laborator 8

Realizarea de aplicatii web folosind Java Servlets

Continut

- Implementarea aplicațiilor web în Java Enterprise Edition
- Tehnologia Java Servlets
- Structura serverului web Apache Tomcat 7.x
- Ciclul de viață al unui Java Servlet
- Structura unui Java Servlet
- Interfațarea Java Servlet cu un sistem de gestiune pentru baze de date
- Mecanisme pentru gestiunea comunicația în Java Servlets

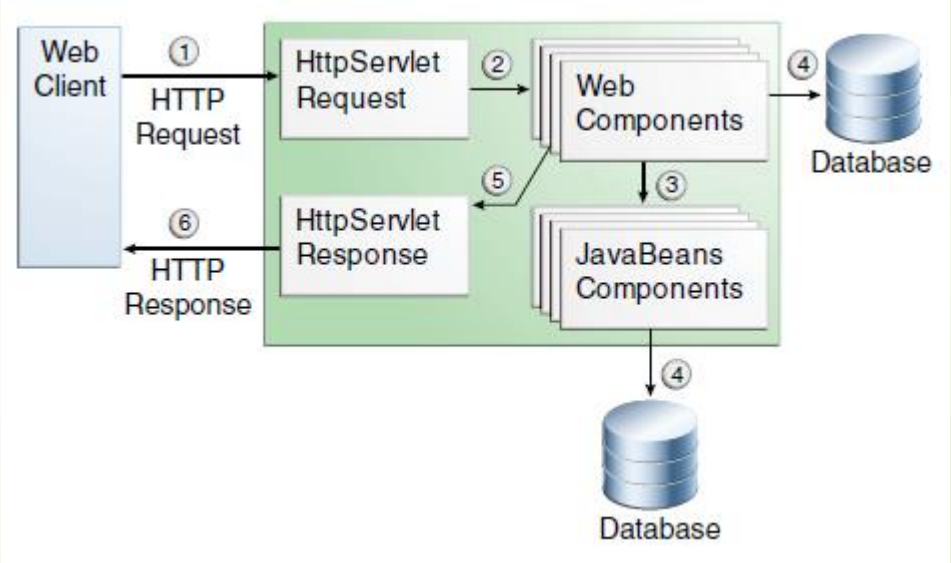
Implementarea aplicațiilor web în Java Enterprise Edition

- aplicatii web
  - extensie dinamica a unui server (web, de aplicatii)
  - transfera cerintele cu privire la resurse (programe instalate) serverului pe care sunt gazduite, functionalitatea oferita fiind accesibila printr-un client universal (browser-ul)
  - clasificare
    - ❶ **orientate pe prezentare** – pagini Internet interactive descrise folosind limbaje de adnotare (HTML, XML) avand continut dinamic generat ca raspuns la cererirle transmise de utilizator
      - ex: Faceles, Java Server Faces
    - ❷ **orientate pe servicii** – implementeaza functionalitatea unui serviciu web
      - ex: Java Servlets (date nontextuale, gestiunea controlului altor tipuri de aplicatii web)
    - orientate pe prezentare ← orientate pe servicii

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

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

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



Sursa: *The Java EE 7 Tutorial, Release 7 for Java EE Platform*, September 2013

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

- ❶ cererile HTTP ale clientilor transmise de browser sunt transformate de serverul web intr-un obiect `HttpServletRequest`
- ❷ cererea `HttpServletRequest` este transmisa unei componente web
- pentru a genera continut dinamic, componenta web poate interactiona
  - ❸ cu o clasa Java Beans
  - ❹ cu baza de date
- ❺ raspunsul este convertit intr-un obiect `HttpServletResponse`
- ❻ serverul web transforma obiectul `HttpServletResponse` intr-un raspuns HTTP care va putea fi vizualizat in browser

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

- configurarea aplicatiilor web
  - aspecte ale comportamentului acestora in diferite situatii
  - sunt incarcate la instalarea (*eng.* deploy) aplicatiei web in contextul containerului
  - mecanisme
    - ❶ adnotari Java
    - ❷ fisiere XML (descriptorul de instalare al aplicatiei web)
      - !!! trebuie sa respecte schemele specificatiei Java Servlet

Tehnologia Java Servlets

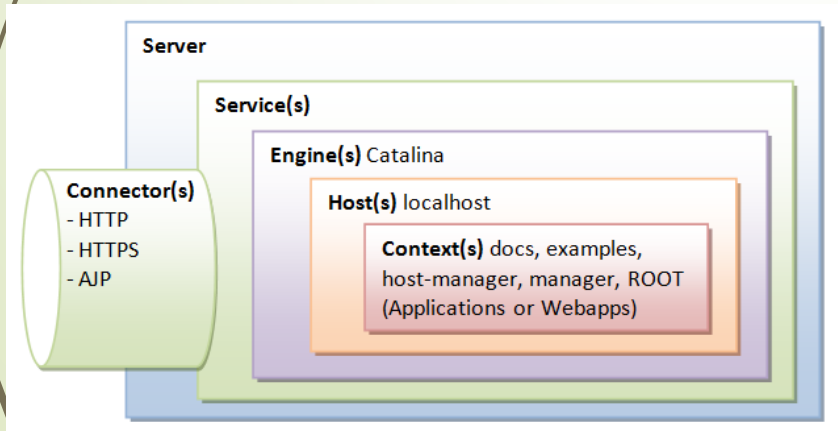
- permite dezvoltarea de aplicatii web dinamice
- alternativa la CGI (Common Gateway Interface), eliminand
  - dependenta de platform
  - scalabilitatea redusa
- caracteristici
  - ❶ eficienta – initializarea se face o singura data, in metoda `init`
  - ❷ persistenta – obiectele unui servlet exista atata timp cat acesta se afla in executie
  - ❸ portabilitate
  - ❹ robustete – acces la toate facilitatile oferite de limbajul de programare Java (ierarhie exceptii, garbage collection)
  - ❺ extensibilitate – extinderea unui servlet se face potrivit modelului de programare orientat obiect
  - ❻ securitate – modelul de securitate Java

Tehnologia Java Servlets (cont'd)

- clasa Java ce extinde capabilitatile unui server web
  - implementeaza interfata `Servlet`
  - tipuri
    - `GenericServlet – service()`
    - `HttpServlet – doMethod()` (`Method={Post,Get,Put,Trace,Options,Delete}`)
  - pachete
    - `javax.servlet`
    - `javax.servlet.http`
- aplicatii conform modelului cerere-raspuns
- adaptata protocolului HTTP
- se poate mapa oricarui tip de protocol

Structura serverului web Apache Tomcat 7.x

- server HTTP ce functioneaza drept container pentru Java Servlets
  - paginile JSP / JSF – transformate automat la clasele Java Servlets asociate atunci cand sunt accesate
- primeste cereri si genereaza raspunsuri din cadrul claselor Java Servlet
- versiunea stabila: 7.0.47 – specificatia Java Servlet 3.0
  - versiunea 8.x – RC5 (alpha)



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

Structura serverului web Apache Tomcat 7.x (cont'd)

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

Structura serverului web Apache Tomcat 7.x (cont'd)

- ❹ `logs` – jurnale specifice
  - motorul Catalina
  - gazdele pe care le gestioneaza
  - aplicatiile `manager` / `host-manager`
  - jurnalul de acces
- ❺ `webapps` – locatia in care vor fi plasate aplicatiile, fiind accesate de serverul web Apache Tomcat
- ❻ `work` – director de lucru ce contine clasele Java Servlet corespunzatoare documentelor JSF / JSP
  - organizarea se face pe motor, gazde configurate in cadrul motorului, aplicatii
- ❼ `temp` – diferite resurse temporare

Structura serverului web Apache Tomcat 7.x (cont'd)

- serviciu → unul sau mai multi conectori la motorul serverului (implicit, Catalina)
  - HTTP / 1.1
    - comunicatia client-server
    - protocolul HTTP (portul 8080)
  - AJP / 1.3
    - comunicatia dintre serverul Tomcat si serverul Apache HTTP
    - protocolul AJP (Apache JServ Protocol) – portul 8009
- container < motor < gazda < context < cluster
  - relatiile au multiplicitatea 1 < n
  - motor: Catalina, gazda: localhost

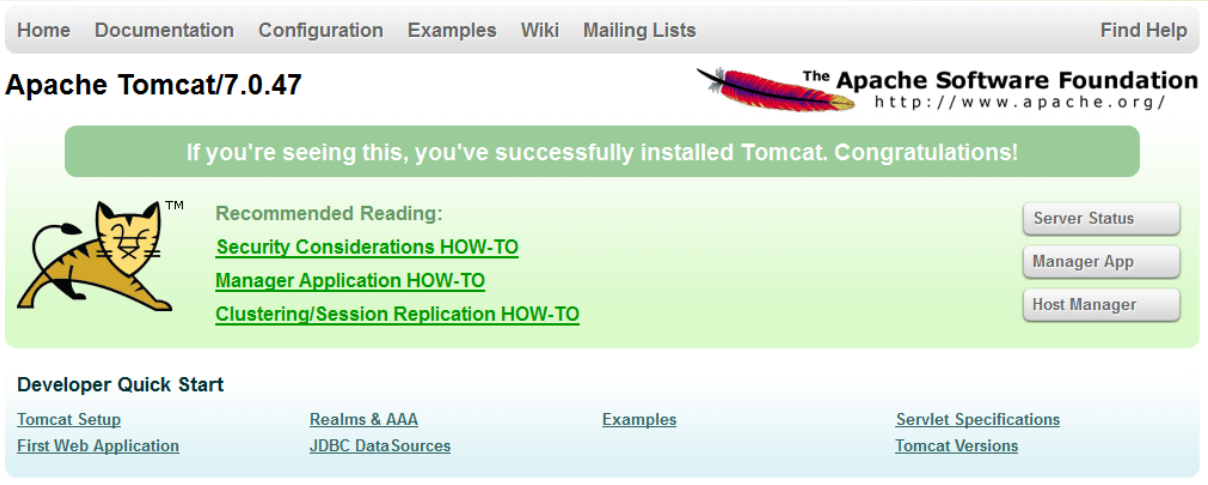
Configurarea serverului Apache Tomcat 7.x

- `<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">`
  - webapps – director de baza unde vor fi dezvoltate aplicatiile
  - unpackWARs – dezarhivarea aplicatiilor dezvoltate ca .war (Web Archive)
  - autoDeploy – instalarea automata a aplicatiilor dupa plasarea lor in directorul corespunzator
- configurarea utilizatorilor (`conf\tomcat_users.xml`)
  - `manager-gui` – interfata grafica cu utilizatorul
    - http://localhost:8080/manager/html sau Manager App
  - `manager-status` – informatii despre starea serverului
    - http://localhost:8080/manager/server sau Server Status
  - `manager-script` – interfata in mod text prin care pot fi transmise comenzi prin parametri din URL
    - http://localhost:8080/manager/text/{comanda}?{parametri}
      - comanda: list (afisare lista aplicatii), deploy (dezvoltare aplicatii)
      - parametri – contextul aplicatiei pentru care se da comanda
  - `manager-jmx` – acces la interfata JMX
    - http://localhost:8080/manager/jmxproxy/?{comanda}={parametri}

Configurarea serverului Apache Tomcat 7.x (con't)

```
<?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>
```

Configurarea serverului Apache Tomcat 7.x (con't)

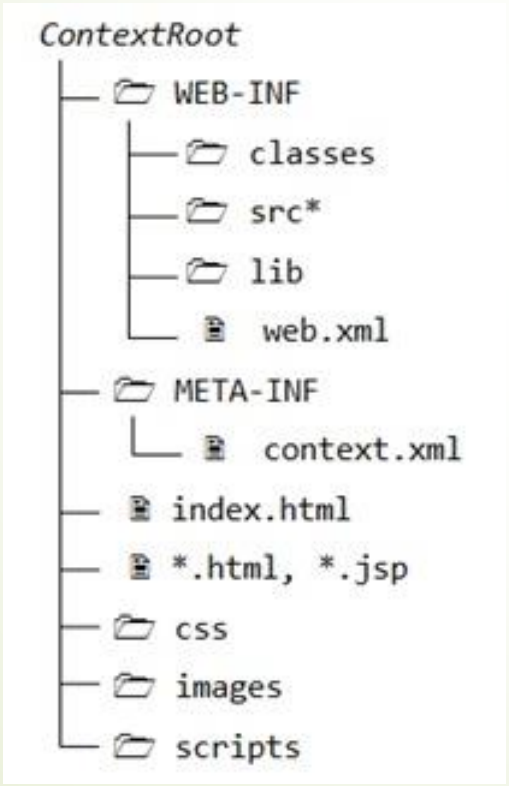


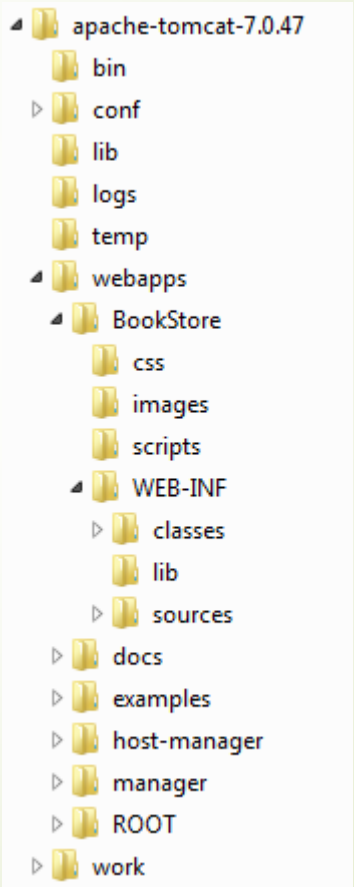
- Server Status – informatii despre starea serverului
- Manager App
  - informatii cu privire la aplicatiile care au fost configurate in contextul serverului web (locatie, versiune, stare curenta, numarul de sesiuni deschise)
  - operatii: pornire, oprire, reincarcare, configurare
  - configurare: perioada de timp dupa care sesiunile nu mai sunt active

Structura unei aplicatii web folosind Apache Tomcat 7.x

- webapps
  - `META-INF` – informatii legate de server (`context.xml`)
  - `WEB-INF` – informatii legate de aplicatie care nu vor fi accesibile clientilor
    - fisierul de configurare `web.xml`
      - servlet: asociere nume servlet (`servlet-name`) si clasa servlet (`servlet-class`)
      - servlet-mapping – contextul de unde poate fi accesat servletul (`url-pattern`)
      - welcome-file-list – pagina Internet incarcata la pornirea aplicatiei web (`welcome-file`)
    - sursele aplicatiei (**`src`** / **`sources`**)
    - clasele aplicatiei – obtinute in urma compilarii surselor (**`classes`**)
    - bibliotecile folosite de aplicatie (**`lib`**)
      - incarcate manual, cu `Class.forName("…")`
  - alte resurse disponibile clientilor (foi de stil, imagini, scripturi)

Structura unei aplicatii web folosind Apache Tomcat 7.x (cont'd)





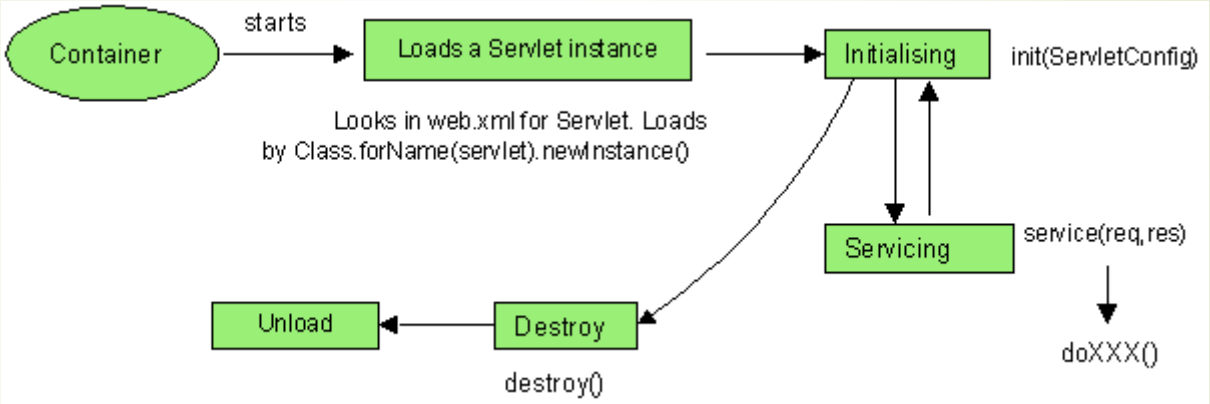
Fisierul de configurare `web.xml`

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

- servlet-ul va fi accesibil la http://localhost:8080/<context>/ClasaServlet
  - context – subdirectorul din `webapps` de unde este accesata aplicatia
- la reinitializarea serverului sunt parcurse toate aplicatiile din `webapps`

```
Nov 25, 2013 12:00:00 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory BookStore
```

Ciclul de viata al unui Java Servlet



Sursa: *Servlet Execution Flow*, http://www.visualbuilder.com/jsp/webcomponent/servlet-execution-flow/

- controlat de containerul in care a fost configurat servlet-ul
- operatii la realizarea unei cereri asociate unui servlet
  - ❶ daca nu exista o instanta a servlet-ului
    - a) incarcarea clasei servlet
    - b) crearea unei instante a clasei servlet
    - c) initializarea instantei clasei servlet prin apelarea metodei `init`
  - ❷ apelarea metodei `service` care are ca parametrii obiectele cerere si raspuns
  - ❸ atunci cand servlet-ul nu mai este necesar, se apeleaza metoda `destroy`

Clase ascultator (adnotate `@WebListener`) asociate evenimentelor din ciclul de viata al unui servlet

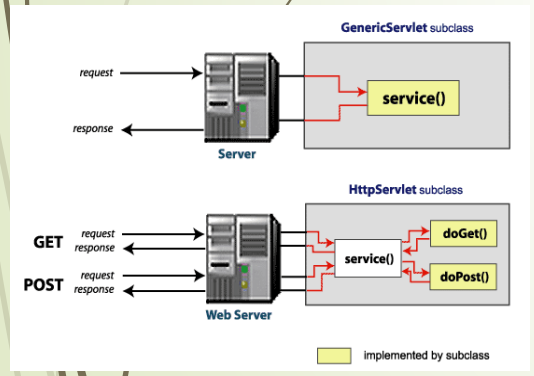
Obiect	Eveniment	Interfață (ce trebuie) Implementată	Obiect Eveniment
context web	creare, distrugere	`javax.servlet.ServletContextListener`	`ServletContextEvent`
	operații asupra atributelor (CRUD)	`javax.servlet.ServletContextAttributeListener`	`ServletContextAttributeEvent`
sesiune	creare, invalidare, 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

- clasa derivata din `javax.servlet.http.HttpServlet`
- adnotata cu insemnarea `@WebServlet`
  - specifica cel putin un URL
  - campurile `value` (1 atribut) / `urlPatterns` (mai multe atribute)
- suprascrie metodele
  - `init(ServletConfig)`
    - operatii realizare o singura data, utile doar obiectului servlet din contextul caruia este apelat
      - incarcarea de informatii persistente (date de configurare)
      - initializarea resurselor
    - invocata ulterior incarcarii / instantierii servlet-ului, anterior acceptarii invocarilor de la clienti
    - daca nu este realizata cu succes, este generata o exceptie `UnavailableException`
    - alternativa la atributul `initParams` al adnotarii `@WebServlet`
      - contine insemnarea `@WebInitParams`

Structura unui Java Servlet (cont'd)

- suprascrie metodele
  - `service(ServletRequest, ServletResponse)`
    - implementeaza funcionalitatea servlet-ului
    - definita in clasa `GenericServlet`
    - clasa `HttpServlet` deleaga functionalitatea ei in functie de tipul de cerere primita
      - `HTTP GET` → `doGet`
      - `HTTP POST` → `doPost`
      - `HTTP PUT` → `doPut`
      - `HTTP DELETE` → `doDelete`
      - metodele `doOptions` / `doTrace` sunt mai putin utilizate
    - tratarea cererii
      - `getParameterNames()` – numele parametrului
      - `getParameter(String)` – valoarea parametrului
    - formularea raspunsului `getWriter()` → `PrintWriter`



Structura unui Java Servlet (cont'd)

- suprascrie metodele
  - `destroy()`
    - apelata atunci cand servlet-ul este inchis
    - toate resursele folosite de servlet trebuie eliberate
    - se asigura persistenta prin retinerea informatiilor necesare din baza de date
    - !!! toate metodele serviciu asociate unui servlet trebuie terminate inainte de distrugerea sa
    - !!! toate firele de executie ce deservesc clienti trebuie sa fie terminate
      - contorizarea numarului de fire de executie active
      - asteptarea terminarii lor prin invocarea metodei `Thread.sleep()`
  - `getServletInfo()` – ofera informatii despre servlet

Gestiunea cererilor prin Java Servlets

- implementeaza interfata `ServletRequest`
  - accesarea parametrilor transmisi de clienti prin intermediul unor formulare
  - obtinerea valorilor unor obiecte folosite la comunicarea dintre un container al servlet-ului si servlet / intre mai multi servleti
  - informatii despre protocolul prin care este transmisa cererea si despre locatie
- `HttpServletRequest`
  - URL-ul cererii
    - http://[adresa]:[port]/[cale]?[interogare]
      - cale = cale contextuala + calea catre servlet + alte informatii
      - `getContextPath`, `getServletPath`, `getPathInfo`
  - antetele HTTP
  - interogarea formata din pereche (nume, valoare) pentru parametrii transmisi
  - metode: `getContentType`, `getCookies`, `getHeaderNames`, `getHeaders`, `getSession`, `getInputStream`, `getMethod`, **`getParameterNames`**, **`getParameter`**

Gestiunea raspunsurilor prin Java Servlets

- implementeaza interfata `ServletResponse`
  - contine datele transmise de la servlet catre client
  - obtinerea unui flux prin care se poate realiza comunicarea cu clientul
  - specificarea tipului de continut: `setContentType("text/html")`
  - alocarea unei zone de memorie: `setBufferSize(int)`
    - datele nu sunt transmise imediat catre client ci dupa completarea zonei de memorie
    - stabilirea unor coduri de stare / antete
- `HttpServletResponse`
  - construirea documentului care va fi transmis de la server catre client intr-un obiect `PrintWriter`
  - contine
    - antete HTTP / coduri de stare: indisponibilitatea resursei, redirectare
    - obiecte ce vor retine informatii specifice aplicatiei (cookies)
  - metode
    - mostenite din `ServletResponse`: `flushBuffer`, `get/setBufferSize`, `get/setContentType`, `getOutputStream`, `setCharacterEncoding`
    - proprii: `addCookie`, `get/setHeader`, `get/setStatus`

Structura unui Java Servlet Exemplu

```
import javax.servlet.*;
import javax.servlet.http.*;
@WebServlet("/sample")
public class SampleServlet extends HttpServlet {
    final public static long    serialVersionUID = 10001000L;
    @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 {
        // ...
    }
}
```

Structura unui Java Servlet Filtre

- functionalitati
  - ❶ analiza unei cereri si actionarea in conformitate cu aceasta
  - ❷ blocarea unei cereri si a raspunsului corespunzator de a fi prelucrate mai departe
  - ❸ modificare antetelor / continutului obiectelor cerere / raspuns construind o versiune particularizata a acestora
  - ❹ interactiunea cu resurse externe
- pot fi atasate mai multor resurse web de care nu sunt dependente
- utilizate la: autentificare, jurnalizare, conversie de imagini, compresie a datelor, criptare, parsarea fluxtului de date, transformari XML

Structura unui Java Servlet Filtre (cont'd)

- lant de filtre = lista ce contine 0, 1 sau mai multe filtre executate intr-o anumita ordine
- `javax.servlet` defineste clasele
  - `Filter`
  - `FilterChain`
  - `FilterConfig`
- implementeaza interfata `javax.servlet.Filter`
- adnotat cu insemnarea `@WebFilter`
  - specifica un URL in `value` sau `urlPatterns`
  - informatiile cu privire la initializare sunt continute in `initParams`

Structura unui Java Servlet Filtre (cont'd)

- functionalitatea este implementata in metoda `doFilter`
  - analizeaza antetele cererii
  - particularizeaza obiectele cerere si raspuns modificand antetele sau continutul
    - poate adauga un atribut la cerere / introduce informatii la raspuns
    - pentru a suprascrie metodele cererii / raspunsului, obiectele vor fi impachetate in obiecte derivate din `[Http]ServletRequestWrapper` / `[Http]ServletResponseWrapper`
  - invoca urmatorul filtru din lantul de filtre (prin apelul metodei `doFilter` a acestuia)
  - analizeaza antetele raspunsului
  - poate bloca comunicatia mai departe
  - poate genera o exceptie pentru a indica producerea unei erori
- asocieri intre filtre si resurse web (prin nume / URL)
  - determina modul si ordinea in care sunt aplicate
  - jurnalizare: masca `/*` (se aplica tuturor evenimentelor referitoare la comunicatia dintre client si server)
  - tipuri de asocieri
    - 1 filtru → mai multe resurse web
    - 1 resursa web → mai multe filtre

Invocarea altor resurse web

- se obtine un obiect `RequestDispatcher` folosindu-se metoda `getRequestDispatcher` ce primeste ca parametru URL-ul resursei ce va fi invocate
  - aplicata pe obiectul cerere – cale relativa
  - aplicata pe contextul web – cale absoluta
  - daca resursa nu este disponibila sau serverul nu implementeaza un obiect RequestDispatcher pentru tipul respective de resursa, metoda intoarce null
- se realizeaza
  - direct
    - incluzand continutul altei resurse
      - header, footer, informatii de copyright, meniuri
      - se foloseste metoda **`include`** (se executa componenta web – primind ca parametru cererea, afisandu-se rezultatul)
    - transmitand mai departe cererea catre o alta resursa
      - se foloseste metoda **`forward`**
      - URL-ul cererii se modifica la cel al paginii, acesta realizand prelucrarea ei si primind responsabilitatea raspunsului
      - nu este permisa daca au fost folosite obiecte `ServletOutputStream` sau `PrintWriter` in servlet (se genereaza `IllegalStateException`)
  - indirect – incorporand un URL catre o alta resursa web in raspunsul transmis catre client

Invocarea altor resurse web (exemplu). Accesarea contextului web

```
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 initializare
  - resurse asociate cu contextul web
  - atribute avand asociate tipuri de obiecte
  - capabilitati legate de jurnalizare

Incarcarea de fisiere

- adnotarea `javax.annotation.MultipartConfig`
  - indica faptul ca servletul pentru care este declarata poate prelucra cereri folosind tipul MIME `multipart/form-date`
  - componentele `javax.servlet.http.Part` pot fi obtinute folosind
    - `Collection<Part> getParts()` – pentru fisiere avand tipuri diferite
    - `Part getPart(String)` – o parte identificata printr-un nume
    - analiza fiecarei parti: nume, dimensiune, tip de continut, procesarea antetelor transmise impreuna cu partea respective, salvarea pe disc, stergerea ei
  - atribute
    - `location` – calea absoluta catre un director din sistemul de fisiere pentru a stoca fisiere temporare in timp ce partile fisierului sunt procesate au cand dimensiunea fisierului depaseste `fileSizeThreshold` / valoarea implicita = “”
    - `fileSizeThreshold` – dimensiunea fisierului (in octeti) dupa care acesta va fi stocat temporar pe disc / implicit = 0
    - `maxFileSize` – dimensiunea maxima pentru incarcarea de fisiere, exprimata in octeti / implicit = nelimitat
    - `maxRequestSize` – dimensiunea maxima pentru o cerere `multipart/form-date`, exprimata in octeti / implicit = nelimitat
    - pot fi specificate in `web.xml` in sectiunea `<multipart-config>`

Procesarea asincrona

- necesara in cazul in care unele fire de executie sunt blocate asteptand anumite resurse, avand un impact negativ asupra scalabilitatii aplicatiilor
- presupune crearea unui fir de executie pentru fiecare operatie blocanta
- `@WebServlet(asyncSupported=true)` – servletul are capacitatea de a realiza procesare asincrona
- `request.startAsync()` → `javax.servlet.AsyncContext` – cererea va fi transferata unui context asincron
  - raspunsul nu va fi transmis clientului la iesirea din metoda `service`
  - raspunsul trebuie generat in contextul asincron dupa terminarea operatiei blocante sau gestionat de alt servlet

Procesarea asincrona (cont'd) Clasa `AsyncContext`

- `void start(Runnable run)` – este creat un nou fir de executie unde va fi procesata operatia blocanta
  - codul ce trateaza prelucrarea trebuie sa fie specificat in clasa ce implementeaza interfata `Runnable`
- `ServletRequest getRequest()` – intoarce cererea folosita pentru a initializa contextul asincron (pentru a obtine parametrii cererii in contextul asincron)
- `ServletResponse getResponse()` – intoarcere raspunsul folosit pentru a initializa contextul asincron (pentru a construi raspunsul cu rezultatele operatiei blocante)
- `void complete()` – termina operatia asincrona si transmite raspunsul asociat contextului asincron
- `void dispatch(String path)` – transmite obiectele cerere / raspuns catre calea indicata (spre a delega altui servlet responsabilitatea cu privire la ele, dupa terminarea operatiei)

Procesarea asincrona (cont'd) Exemplu

```
@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();
            }
        });
    }
}
```

Operatii de intrare/iesire asincrone

- necesare atunci cand operatiile de intrare / iesire sunt mai rapide pe server decat pe client
- asociate de regula cu procesarea asincrona pentru eliminarea timpilor morti din firele de executie
- se folosesc clase ascultator care detecteaza momentul in care operatiile pot fi realizate non-blocant
- `javax.servlet.ServletInputStream` → `void setReadListener(ReadListener rl)` – asociaza fluxului de intrare un obiect ascultator care contine 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)` – asociaza fluxului de iesire un obiect ascultator care contine metode pentru a scrie date asincron
  - metode: `onWritePossible()`, `onError(Throwable)`
  - `isReady()` – datele pot fi scrise non-blocant

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

- folosind metodele puse la dispozitie de API-ul Java DataBase Connectivity
- driver-ul de conectare la baza de date trebuie incarcat manual inainte de a realiza accesul propriu-zis la informatii
  - `Class.forName("com.mysql.jdbc.Driver")`
- pot fi reutilizate metodele de acces la baza de date folosite in cazul aplicatiilor desktop
- clientul are acces la informatiile din baza de date fara a avea nevoie de nici un utilitar suplimentar, acestea gasindu-se pe server

Mecanisme pentru gestiunea comunicatiei folosind Java Servlets

- necesare datorita faptului ca HTTP este un protocol fara stare
  - cererile si raspunsurile sunt tranzactii isolate
  - trebuie corelate accesarile care provin de la acelasi utilizator
- solutii
  - ❶ campurile ascunse
    - `<INPUT type="hidden" … />` - continute in formularele din paginile HTML
    - pot fi identificate cu usurinta
  - ❷ rescrierea URL-urilor
    - adaugarea informatiilor la URL-urile paginilor transmise utilizatorilor
    - se foloseste impreuna cu protocolul `HTTP GET`

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

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

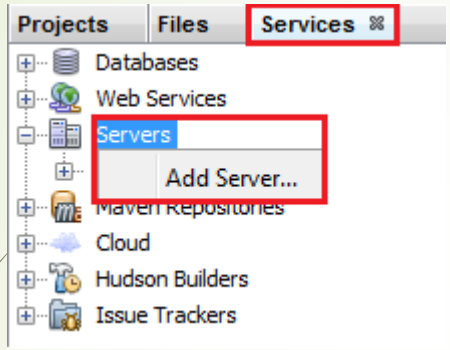
Mecanisme pentru gestiunea comunicatiei folosind Java Servlets (cont'd)

- solutii
  - ❹ sesiuni – identificarea conexiunii dintre client si server prin crearea unui obiect specific acesteia
    - `javax.servlet.http.HttpSession`
    - `request.getSession()` → `HttpSession`
    - atribute (avand ca valori diferite obiecte) pot fi asociate sesiunii prin nume
      - obtinerea 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 activarii / pasivizarii sesiunii

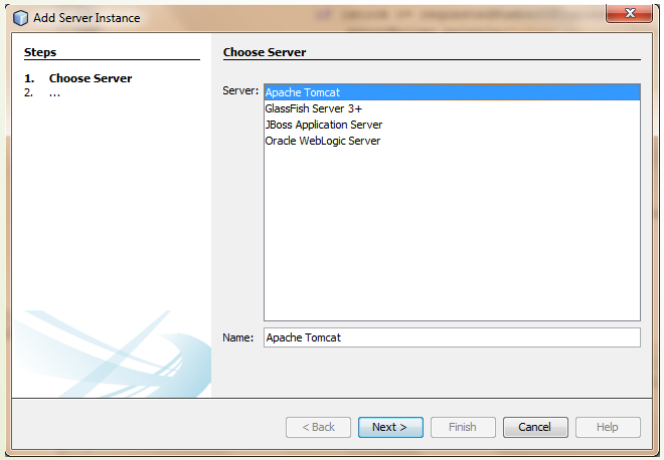
Mecanisme pentru gestiunea comunicatiei folosind Java Servlets (cont'd) – Sesiuni

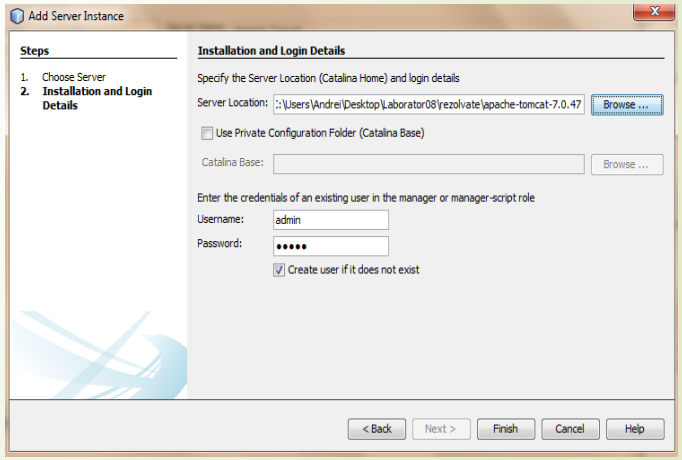
- implementate prin cookie-uri / rescrierea URL-urilor
  - identificatorul conexiunii dintre server si client este retinut pe client (cookie) sau inclus in URL-ul transmis clientului
  - `encodeURL(URL)` – pe obiectul raspuns
    - rescrie URL-ul daca clientul nu accepta cookie-uri
    - lasa URL-ul neschimbat daca clientul accepta cookie-uri
- expirarea sesiunii
  - `{set/get}MaxInactiveInterval`
  - perioada dupa care sesiunea nu mai este necesara, resursele folosite de aceasta putand fi eliberate
  - timpul de expirare este reinitializat prin accesarea metodelor serviciu
  - `invalidate` – apelata cand interactiunea cu un client este incheiata

Interfatarea NetBeans 7.4 cu Apache Tomcat 7.0.47

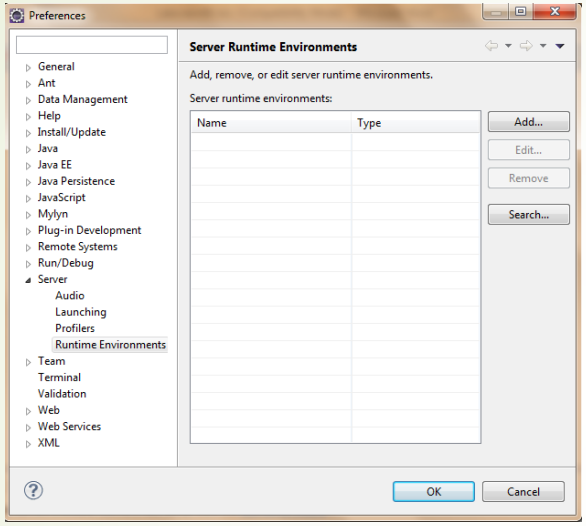


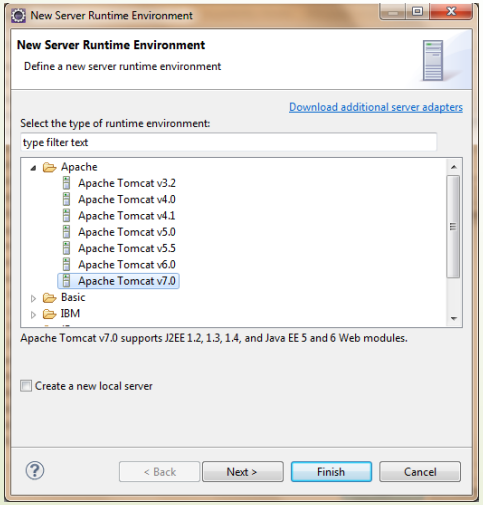
- ❶ crearea unui server
- ❷ specificarea tipului (Apache Tomcat)
- ❸ specificarea locatiei si a parametrilor pentru autentificarea utilizatorului `manager-script`
- ❹ "instalarea" aplicatiei cu *Deploy* din meniul contextual al proiectului





Interfatarea Eclipse 4.3 for Java EE Developers cu Apache Tomcat 7.0.47





- ❶ adaugarea serverului Apache Tomcat 7.0.47 la mediul de dezvoltare Eclipse 4.3 for Java EE Developers (Window → Preferences → Server → RuntimeEnvironments)
- ❷ se specifica tipul serverului (Add → Apache Tomcat v7.0)
- ❸ se indica directorul de instalare Apache Tomcat si JRE-ul folosit
- ❹ din meniul contextual al proiectului se selecteaza *Run As… → Run On Server* (se bifeaza optiunea *Always use this server when running this project*)

Interfatarea Eclipse 4.3 for Java EE Developers cu Apache Tomcat 7.0.47 (cont'd)