Programski Sistemi

Web programiranje u Javi: Servleti

Web aplikacije

- Web aplikacije se koriste za pravljenje dinamičkih Web sajtova
  - Izgled Web strane zavisi od podataka koje je uneo klijent
  - Web strana se generiše na osnovu podataka koji se često menjaju
  - Web strana koristi podatke iz baza podataka ili drugih izvora podataka na serveru
- Java podržava Web aplikacije kroz servlete i Java Server Pages (JSP)

Web serveri i klijenti

- Web (HTTP) server je softver koji obrađuje zahtev klijenta i šalje mu odgovor
- Web server se izvršava na fizičkoj mašini i sluša klijentske zahteve na određenom portu (obično 80)
  - Jakarta Apache
  - Microsoft IIS
- Web klijent je čitač (Firefox, Chrome, IE)
- Web serveri podržavaju Java servlete pomoću ekstenzija koje se zovu servlet kontejneri

HTML i HTTP

- Web klijent (čitač) za prikaz sadržaja koristi jezik HTML (HyperText Markup Language)
- Web server i klijent imaju zajednički protokol za komunikaciju: HTTP (HyperText Transfer Protocol)
- HTTP je sloj iznad protokola TCP/IP; klijent šalje HTTP request (zahtev), a server odgovara sa HTTP response
- Protokol HTTP je stateless; svaki par request/response je nezavisan i nema načina da se prati niz zahteva upućenih od istog klijenta (tzv. sesija)

URL

- URL (Universal Resource Locator) je jedinstvena adresa resursa na Webu
- Primer: **http://localhost:8080/FirstServletProject/jsps/hello.jsp**
- **http**:// je protokol
- **localhost** je jedinstvena adresa servera; obično se ime servera preslikava na jedinstvenu IP adresu
- **8080** je port na kome server osluškuje zahteve; ako se ne navede, zahtev odlazi na default port za određeni protokol (80 za HTTP, 443 za HTTPS, 21 za FTP itd.)
- **FirstServletProject/jsps/hello.jsp** je resurs koji se zahteva od servera; to može biti statička html strana, pdf, JSP, servlet, PHP fajl i sl.

HTTP request

- Najvažniji delovi HTTP zahteva su:
  - HTTP Method: akcija koja se izvodi, obično GET, POST, PUT, DELETE
  - URL: resurs kome se pristupa
  - Parametri forme: slično argumentima Java metoda, npr. korisničko ime i lozinka koji stižu iz polja HTML strane klijenta

```
GET /FirstServletProject/jsps/hello.jsp HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache
```

GET metod u zahtevu

- Prednost metode GET je jednostavnost;
  ne moraju se čak ni praviti polja za unos podataka, već se oni mogu dodati na kraj URL-a
- Nedostaci su ograničena dužina stringa koji se može poslati (oko 2000 znakova)
- U čitaču se prikazuju svi uneti podaci, što nije pogodno ako se šalju osetljive informacije
- Metod POST nema ograničenja broja i dužine parametara

HTML obrasci

- HTML obrasci služe za pravljenje raznih kontrola za prikupljanje ulaznih podataka od korisnika
- Svaka kontrola u obrascu obično ima ime i vrednost; ime je zadato u HTML kodu, a vrednost unosi korisnik
- Ceo obrazac (HTML tag <FORM>) povezan je sa URL adresom programa koji obrađuje podatke
- Obavezan atribut taga <FORM> je tip metoda koji se koristi (GET ili POST)

HTML obrasci: slanje podataka

- HTTP GET dodaje string na kraj zadate URL adrese, nakon znaka pitanja

  Ime1=Vrednost1&Ime2=Vrednost2...ImeN=VrednostN

- U POST metodu, serveru se prvo šalju URL zaglavlja HTTP zahteva i prazan red, a u sledećem redu se šalje string sa parametrima

```
<form method="get" action="FormServlet">
   <input type="text" name="tekst">
   <input type="submit" value="Posalji">
</form>
```

HTTP response

- Važni delovi HTTP odgovora su:
  - **Status Code**: ceo broj koji označava da li je zahtev bio uspešan ili ne. Česti statusni kodovi su 200 (uspešno), 404 (resurs nije pronađen) i 403 (pristup zabranjen)
  - **Content Type** – text, html, image, pdf itd. Poznat i kao MIME tip.
  - **Content** – stvarni podaci koje klijent prikazuje korisniku

HTTP response: kodovi

"200" ; OK
"201" ; Created
"202" ; Accepted
"204" ; No Content
"301" ; Moved Permanently
"302" ; Moved Temporarily
"304" ; Not Modified
"400" ; Bad Request
"401" ; Unauthorized
"403" ; Forbidden
"404" ; Not Found
"500" ; Internal Server Error
"501" ; Not Implemented
"502" ; Bad Gateway
"503" ; Service Unavailable

- **100-199**
  Codes in the 100s are informational, indicating that the client should respond with some other action.
- **200-299**
  Values in the 200s signify that the request was successful.
- **300-399**
  Values in the 300s are used for files that have moved and usually include a `Location` header indicating the new address.
- **400-499**
  Values in the 400s indicate an error by the client.
- **500-599**
  Codes in the 500s signify an error by the server.

HTTP response

- Server šalje MIME tip (tip podataka koji se šalje) klijentu da bi mu pomogao da prikaže podatke korisniku
- Najčešće se koriste tipovi text/html, text/xml, application/xml

```
200 OK
Date: Wed, 07 Aug 2013 19:55:50 GMT
Server: Apache-Coyote/1.1
Content-Length: 309
Content-Type: text/html;charset=US-ASCII

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Hello</title>
</head>
<body>
<h2>Hi There!</h2>
<br>
<h3>Date=Wed Aug 07 12:57:55 PDT 2013
</h3>
</body>
</html>
```

MIME

Servleti

- Servleti su Java programi koji se izvršavaju na Web serveru i rade kao međusloj između zahteva koji stižu iz čitača Weba ili nekog drugog HTTP klijenta i baza podataka ili aplikacija na HTTP serveru
- Servleti se izvršavaju u Web serveru koji podržava Javu (to je tzv. servlet kontejner)
- Obično se koriste kao kontroleri u MVC arhitekturi
- Trenutno aktuelna verzija je Servlet API 3

Servlet kontejneri

- Zovu se i servlet engines ili application servers
- Primeri:
  - Glassfish (podržava kompletnu specifikaciju trenutno aktuelne verzije JavaEE 6)
  - Tomcat
  - Resin
  - JBoss
- Servlet kontejner može da bude prošireni Web server (tj. da radi i kao HTTP server i kao server aplikacija), ali i da bude isključivo server aplikacija

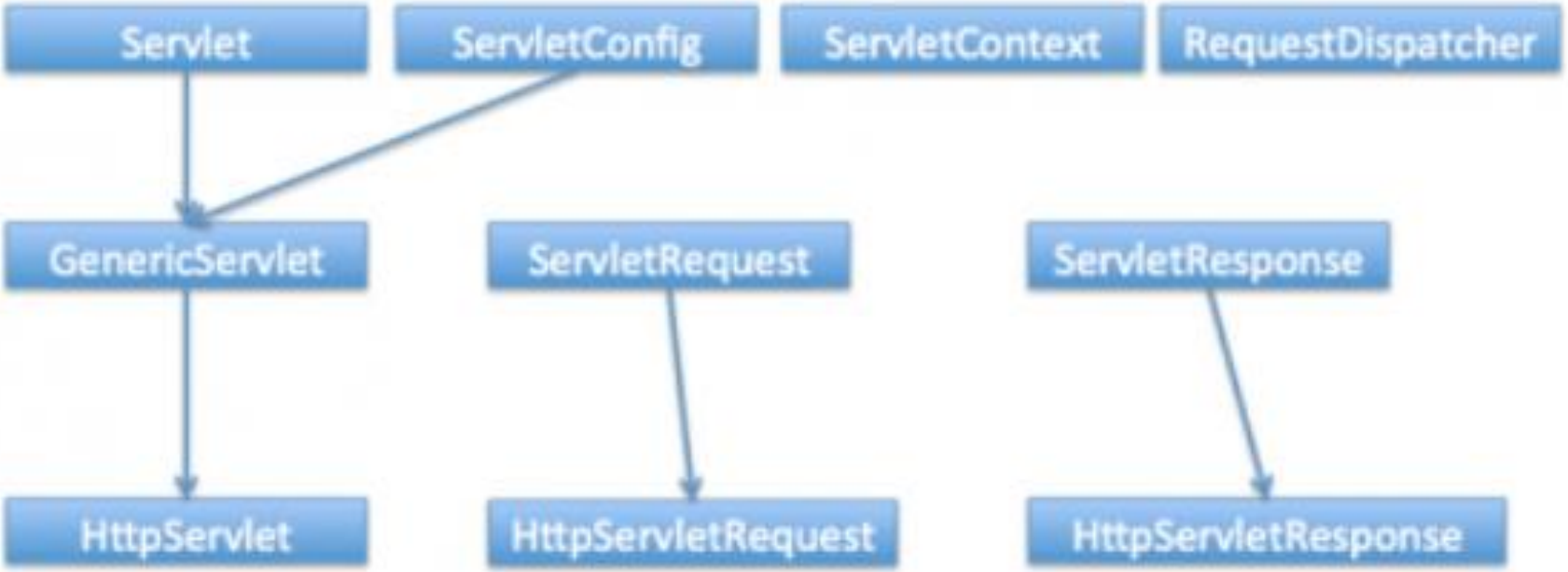
Servlet kontejneri

- Kada servlet kontejner dobije zahtev, kreira dva objekta tipa HTTPServletRequest i HTTPServletResponse
- Zatim pronalazi odgovarajući servlet na osnovu URL-a i kreira nit za zahtev
- Zatim poziva metod service() servleta; on u zavisnosti od HTTP metoda koji je korišćen poziva doGet() ili doPost()
- Metode servleta generišu dinamičku stranu i upisuju je u odgovor
- Kada se nit servleta izvrši, kontejner konvertuje odgovor u HTTP response i šalje ga klijentu

Zadaci servlet kontejnera

- Podrška za komunikaciju između Web servera s jedne strane, i servleta/JSP sa druge strane
- Upravljanje životnim ciklusom servleta (kreiranje, inicijalizacija, pozivanje metoda i uništavanje)
- Podrška za višenitni rad
- Kompajliranje JSP strana u servlete
- Upravljanje resursima, optimizacija memorije, bezbednosna podešavanja, podrška za izvršavanje više aplikacija itd.

Servlet API


Servlet
ServletConfig
ServletContext
RequestDispatcher
GenericServlet
ServletRequest
ServletResponse
HttpServlet
HttpServletRequest
HttpServletResponse

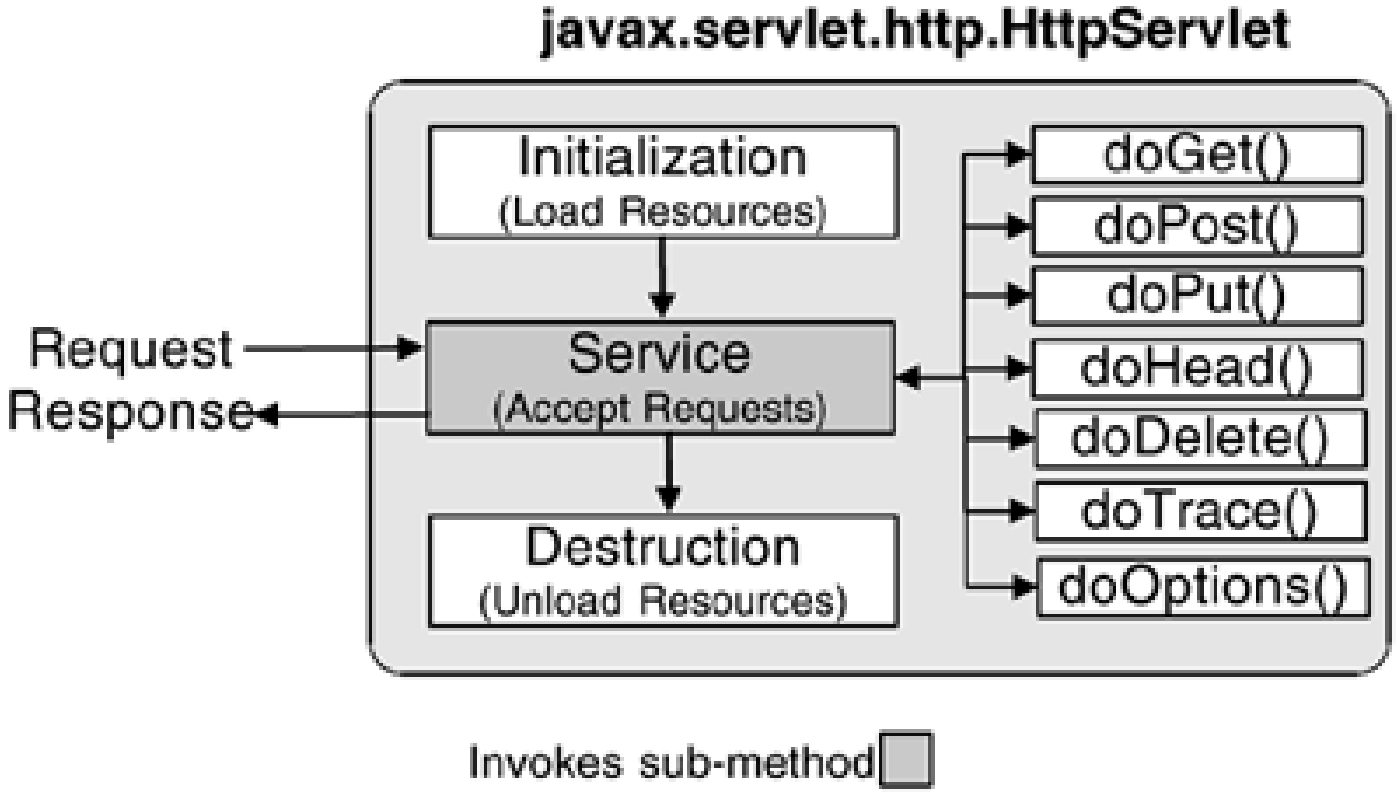
Osnovna struktura servleta

- Da bi bila servlet, Java klasa treba da implementira interfejs **Servlet** iz paketa **javax.servlet,** ili češće da bude izvedena iz klase **HttpServlet** koja već implementira taj interfejs
- Potrebno je nadjačati metode **doGet()** ili **doPost()**, u zavisnosti od toga da li se podaci šalju GET ili POST metodom
- Ako isti servlet treba da obrađuje i GET i POST metode, onda doGet treba da poziva doPost (ili obrnuto) ili da oba metoda pozivaju **processRequest()**

Zadaci servleta

- Čitanje podataka koje je poslao klijent
- Traženje svih informacija o zahtevu koje su ugrađene u HTTP request
- Generisanje rezultata
- Formatiranje rezultata unutar dokumenta
- Podešavanje odgovarajućih parametara za HTTP response
- Slanje povratnog dokumenta klijentu

Životni ciklus servleta



Životni ciklus servleta

- Kada se servlet prvi put kreira, poziva se njegov metod **init()**; on treba da sadrži kod koji se izvršava samo jednom. Ovaj metod treba nadjačavati samo ako je za servlet potrebna dodatna inicijalizacija.
- Nakon toga, svaki zahtev od klijenta stvara novu nit koja poziva metod **service()** prethodno kreirane instance servleta.
- Metod **service()** ne treba nadjačavati
- Više konkurentnih zahteva obično kreiraju više niti koje istovremeno pozivaju metod **service()**

Životni ciklus servleta

- Metod **service()** zatim poziva metode **doGet()** ili **doPost()**, u zavisnosti od tipa primljenog HTTP zahteva
- Nadjačavanjem metoda **doGet()** ili **doPost()** postiže se željeno ponašanje servleta
- Kada server odluči da obriše instancu servleta, poziva njegov metod **destroy()**
  - Ovaj metod obično obavlja oslobađanje resursa koje je servlet zauzimao (zatvaranje datoteka, prekid konekcije sa bazom i sl.)

Šablon servleta

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

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

ServletConfig

- Metod **getServletConfig()** interfejsa Servlet daje objekat ServletConfig
- Svaki servlet ima sopstveni objekat ServletConfig koji instancira servlet kontejner
- Pomoću metode **ServletConfig.getInitParameter()** mogu se pročitati početni parametri iz deployment descriptora (fajla web.xml)
- Metod **getInitParameterNames()** vraća enumeration imena početnih parametara definisanih za servlet

ServletContext

- Interfejs **javax.servlet.ServletContext** obezbeđuje skup metoda koje servlet može da koristi za komunikaciju sa Web serverom
- Objekat ServletContext je sadržan u objektu javax.servlet.ServletConfig, preko koga se dobija pozivom metode **getServletContext()**
- ServletContext je jedinstven objekat dostupan svim servletima unutar Web aplikacije

ServletContext

- Pomoću objekta ServletContext servlet može da:
  - Postavlja i čuva atribute kojima će pristupati drugi servleti iz istog konteksta
  - Vodi log (dnevnik) događaja
  - Dobija URL reference na resurse
  - Dobija atribute za inicijalizaciju; to su parametri povezani sa servletom, a ne sa pojedinačnim zahtevom
  - Dobija MIME tip fajlova
  - Dobija informacije o servlet kontejneru, kao što su njegovo ime i verzija

ServletContext

- Ime ovog interfejsa trebalo bi zapravo da bude **ApplicationContext** jer je on zajednički za aplikaciju, tj. nije specifičan ni za jedan servlet
- Takođe, ne treba ga mešati sa servlet kontekstom koji se prosleđuje u URL-u da bi se pristupilo Web aplikaciji

Interakcija sa klijentom

- Kada servlet primi zahtev od klijenta, on prihvata dva objekta
  - Prvi je **ServletRequest**, koji održava komunikaciju od klijenta do servera
  - Drugi je **ServletResponse**, koji održava komunikaciju od servera ponovo ka klijentu
- ServletRequest and ServletResponse su interfejsi definisani u paketu javax.servlet

HttpServletRequest

- Klasa **HttpServletRequest** implementira interfejs **ServletRequest** i omogućuje servletu da pristupi:
  - parametrima koje šalje klijent
  - protokolu koji koristi klijent
  - nazivu udaljenog računara sa koga je stigao zahtev koji je server primio
  - broju porta na kome sluša server
  - informacijama u zaglavlju HTTP zahteva

HTTPServletRequest

- Prihvata argumente koje klijent šalje kao deo svog zahteva preko metoda **getParameter()**, koji vraća vrednost imenovanog parametra
- Ako parametar ima više od jedne vrednosti, koristi se metod **getParameterValues()**
- **request.getParameterNames() ili**
- **request.getParameterMap()**
  - daju Enumeration ili Map objekata od poslatih elemenata
  - Obično se koriste za debagovanje

Čitanje parametara iz zahteva

- **request.getParameter(“imeParametra")**
  - Dobija se vrednost prvog elementa koji se zove *imeParametra*
  - Ponaša se isto i za GET i za POST zahteve
  - Rezultat je null ako ne postoji takav parametar u elementima HTML obrasca
- **request.getParameterValues(“imeParametra")**
  - Dobija se niz vrednosti za sve elemente koji se zovu ime
  - Dobija se niz sa jednim elementom, ako se ime pojavljuje samo jednom
  - Rezultat je null ako ne postoji takav parametar u elementima HTML obrasca

HTML strana sa parametrima

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>

<FORM ACTION="/servlet/coreservlets.ThreeParams">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY>
</HTML>
```

Čitanje parametara

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

public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Čitanje HTTP zaglavlja

- Postiže se pozivom metoda **getHeader()** klase HttpServletRequest
  - Vraća String ako zahtev sadrži traženo zaglavlje, ili null u suprotnom.
- Imena zaglavlja nisu case sensitive:
  svejedno je da li se koristi request.getHeader("Connection") ili request.getHeader("connection")

Čitanje HTTP zahteva

- Neka zaglavlja su toliko česta da objekat HTTPServletRequest ima posebne metode za njih
  - **getCookies()** vraća sadržaj zaglavlja Cookie, parsiran kao niz objekata Cookie
  - **getAuthType()** i **getRemoteUser()** daju sastavne delove zaglavlja Authentication
  - **getContentLength()** vraća int dužinu zaglavlja Content-Length
  - **getContentType()** vraća String vrednost zaglavlja Content-Type
  - **getMethod()** vraća tip metoda (GET ili POST)

HttpServletResponse

- Omogućuje servletima generisanje odgovora na zahtev
- Nudi metode za definisanje sadržaja odgovora i tipa podataka (MIME)
- Obezbeđuje izlazni tok, tj. objekte ServletOutputStream i Writer preko kojih servlet šalje podatke klijentu

HTTP response: metode

- **String encodeURL(java.lang.String url)** kodira zadati URL dodavanjem ID sesije, ako je potrebno
- **void sendRedirect(String location)** – privremeno preusmerava klijenta na URL lokaciju zadatu kao parametar
- **void setStatus(int sc)** – postavlja statusni kod za response

HttpServletResponse

- Daje dve mogućnosti za vraćanje podataka klijentu
  - **response.getWriter()** vraća Writer, koristi se kada su podaci koji se vraćaju klijentu tekstualnog tipa
  - **response. getOutputStream()** vraća ServletOutputStream, koristi se za binarne podatke
- Pozivanje metoda **close()** ovih objekata nakon slanja odgovora omogućava serveru da zna kada je odgovor kompletiran

Generisanje HTML koda

- Potrebno je obavestiti čitač Weba da se šalju podaci koji su tipa HTML
  - response.setContentType("text/html");
- Menja se naredba println da bi se dobila korektna Web stranica
- Naredbe print koriste HTML tagove

Servlet koji generiše HTML

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

public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Preusmeravanje

- Servleti često moraju da preusmere obradu klijentskog zahteva na neki drugi URL, servlet ili JSP stranu
- To se može uraditi pomoću dve metode: **HttpServletResponse.sendRedirect()** ili **RequestDispatcher.forward()**
- JavaSript takođe omogućuje redirekciju na strani klijenta, ali to ovde nije tema

RequestDispatcher

- Interfejs koji se koristi za preusmeravanje zahteva na drugi resurs (HTML, JSP ili drugi servlet unutar istog konteksta)
- Referenca na objekat RequestDispatcher može se dobiti preko metode **ServletContext.getRequestDispatcher(String putanja)**
- Putanja mora da počinje sa / i tumači se relativno u odnosu na koren konteksta.

RequestDispatcher: metode

- **void forward(ServletRequest request, ServletResponse response)**
  - prosleđuje zahtev od servleta ka drugom resursu (servletu, JSP ili HTML strani) na serveru
- **void include(ServletRequest request, ServletResponse response)**
  - Uključuje sadržaj resursa (servleta, JSP ili HTML strane) u response

Preusmeravanje

- Za razliku od metode forward() RequestDispatchera koja preusmerava na strani servera preko originalnih objekata request i response, metod response.sendRedirect() šalje request sa novom URL adresom nazad klijentu, koji se zatim povezuje sa tim URL-om, što znači da se pravi novi par objekata request/response
- sendRedirect() je sporiji od metode forward(), ali omogućuje preusmeravanje na bilo koji URL, dok forward() radi samo unutar iste Web aplikacije

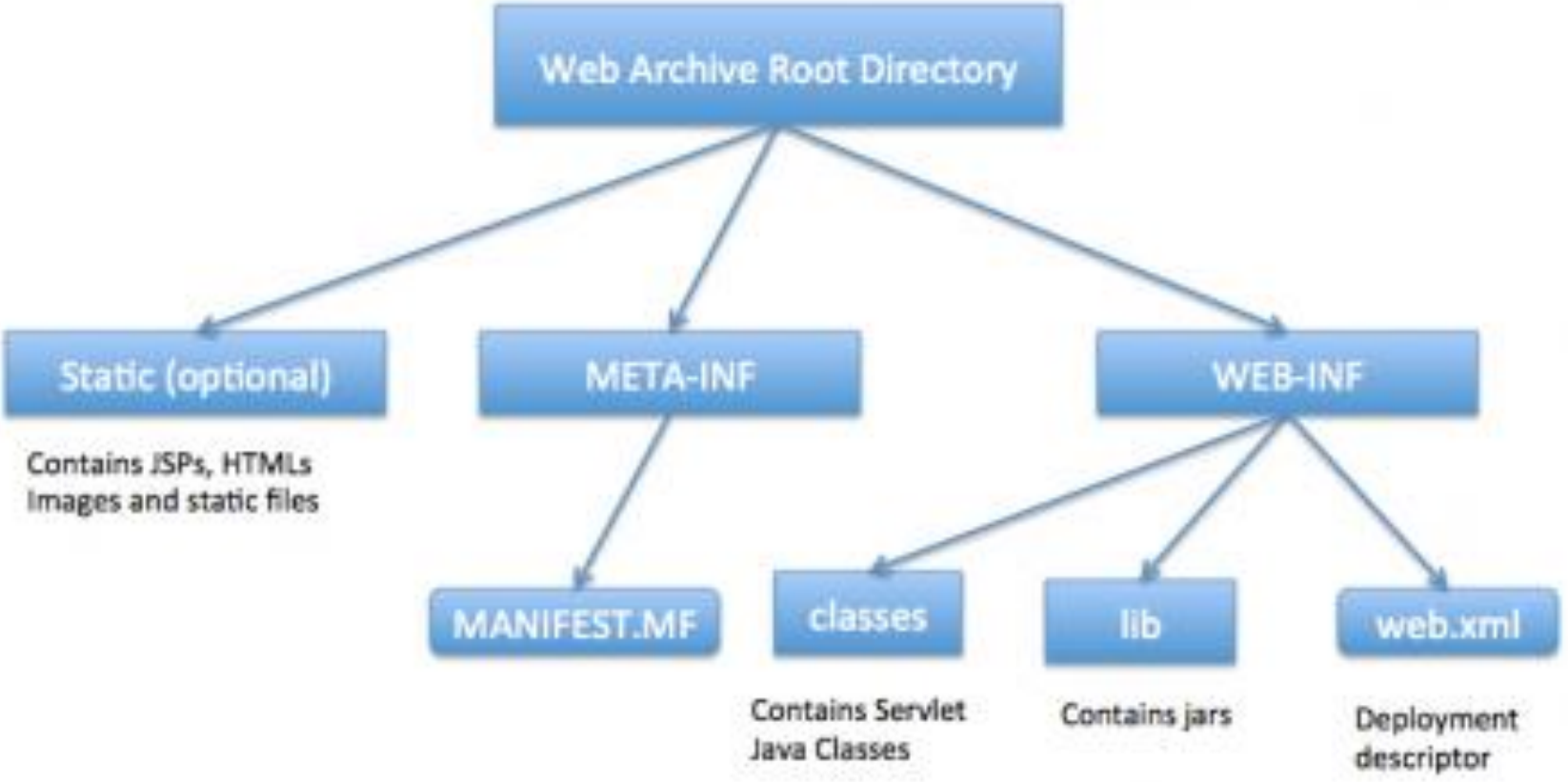
Atributi servleta

- Koriste se za komunikaciju između servleta, tj. na nivou cele aplikacije
- Imaju tri oblasti važenja: **request scope**, **session scope** i **application scope**
- Interfejsi ServletRequest, HttpSession i ServletContext imaju metode get/set/remove atributa iz zahteva, sesije i aplikacije.
- Atributi servleta nisu isto što i početni parametri definisani u web.xml za objekte ServletConfig ili ServletContext

Servlet deployment

- Sam po sebi, servlet nije kompletna Java aplikacija, već mora da bude deo Web aplikacije kojom upravlja servlet kontejner
- Korišćenje servleta za generisanje dinamičkih odziva obuhvata kako programiranje servleta, tako i njegovo postavljanje u Web aplikaciju; taj postupak se zove **deploy**

Struktura Java Web aplikacije



Servlet deployment

- **korenski direktorijum (/)**
  - polazna tačka Web aplikacije; zove se kontekst
  - sadrži sve resurse Web aplikacije (statičke kao što su html i jsp strane, slike i sl. ali i dinamičke)
- **/WEB-INF/**
  - sadrži Web Application Deployment Descriptor (tj. konfiguracioni fajl **web.xml**) koji opisuje deploy
- **/WEB-INF/classes/**
  - Za razliku od statičkih resursa, servlet se ne postavlja direktno u korenski direktorijum, već unutar direktorijuma /WEB-INF/classes, zajedno sa svim drugim kompajliranim Java klasama

Pozivanje servleta

- Servleti se mogu pozivati direktno upisom URL putanje unutar čitača
- Format URL putanje zavisi od servera koji se koristi
  - Pristup servletu – opšti šablon
    `http://[domen]/[context]/[servlet[/[ServletskaKlasa]`
  - npr. za Web server Jakarta Tomcat kod koga je u direktorijumu TOMCAT_HOME/webapps instalirana aplikacija (kontekst) servletintro, poziv servleta HelloServlet bi izgledao ovako:
    `http://localhost:8080/servletintro/servlet/HelloServlet`

Deployment descriptor

- Umesto direktnog pozivanja servleta, u konfiguracionom fajlu **web.xml** zadaje se preslikavanje servleta u URL koji definiše administrator
- Primer
  - Servletska klasa: **HelloServlet.class**
  - Kontekst aplikacije
    `http://localhost:8988/servletintro/`
  - Direktno pozivanje
    `http://localhost:8988/servletintro/servlet/HelloWorld`
  - Preslikavanje putem fajla web.xml
    `http://localhost:8988/servletintro/hello`

web.xml

```
<?xml version="1.0" encoding="UTF-8"?> 2: <!DOCTYPE web-app
➥PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
➥'http://java.sun.com/dtd/web-app_2_3.dtd'> 3:
<web-app>
    <display-name>A Simple Application</display-name>
    <servlet>
        <servlet-name>catalog</servlet-name>
        <servlet-class>CatalogServlet</servlet-class>
        <init-param>
            <param-name>catalog</param-name>
            <param-value>Spring</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>catalog</servlet-name>
        <url-pattern>/catalog/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    <error-page>
        <error-code>404</error-code>
        <location>/error404.html</location>
    </error-page>
</web-app>
```

web.xml

XML tag	Opis
web-app	korenski tag deployment descriptora
display-name	kratko ime aplikacije, ne mora da bude jedinstveno
servlet-name	zvanično ime servleta, mora da bude jedinstveno
servlet-class	kompletno ime servletske klase
init-param	inicijalizacioni parametri servleta; slede parovi ime-vrednost
param-name	ime parametra
param-value	vrednost parametra
servlet-mapping	preslikavanje servleta u URL
session-config	definiše period trajanja sesije
error-page	preslikavanje statusnog HTTP koda u Web resurs, npr. html stranu, koja će se prikazivati umesto standardnih stranica čitača

WAR (Web Application Archive)

- JEE definiše standardan način za deploy Web aplikacija njihovim pakovanjem u WAR arhive
- Sve komponente Web aplikacije treba da budu spakovane u fajl sa ekstenzijom.war, koji ima isti format kao .jar fajlovi
- WAR arhive se mogu napraviti programom jar, kao i obične arhive koje sadrže mnogo fajlova u komprimovanom obliku kompatibilnom sa zip fajlovima

WAR

- Struktura direktorijuma u WAR arhivi treba da bude sledeća:
  - Prvi nivo sadrži resurse koji se i inače stavljaju u korenski direktorijum aplikacije (HTML i JSP strane)
  - Poddirektorijum **META-INF** sadrži fajl manifest.mf sa informacijama o fajlovima u arhivi
  - Direktorijum **WEB-INF** sadrži deployment descriptor (web.xml)
  - Poddirektorijum **classes** sadrži kompajlirane servlete i ostale Java klase
  - U poddirektorijum **lib** smeštaju se dodatne jar arhive

Upravljanje sesijom

- Protokol HTTP i Web serveri su stateless, tj. nisu u stanju da prepoznaju da npr. dva uzastopna zahteva stižu od istog klijenta
- Često je u aplikaciji potrebno da prepoznamo da zahtevi stižu od istog klijenta u određenom vremenskom intervalu; to je tzv. sesija
- Jedini način da održimo sesiju je da se između klijenta i servera pri svakoj request/response komunikaciji razmenjuje jedinstvena informacija o sesiji (id sesije).

Upravljanje sesijom

- Postoji više načina da se upravlja sesijom:
- **Autentikacija korisnika**
  - Na strani za login tražimo od korisnika autentikaciju i onda informacije o korisničkom imenu prenosimo između klijenta i servera da bi održali sesiju. Ovaj metod neće raditi ako se isti korisnik prijavi istovremeno iz različitih čitača.
- **Skriveno HTML polje**
  - Kada korisnik započne navigaciju, generišemo jedinstvenu vrednost skrivenog polja i šaljemo je. Ovaj metod ne može da se koristi sa linkovima jer form mora da bude poslat sa skrivenim poljem kad god se generiše zahtev od klijenta. Takođe, nije bezbedan jer se vrednost skrivenog polja može pročitati iz HTML-a.

Upravljanje sesijom

- **URL Rewriting**
  - Parametar ID sesije se dodaje svakom paru request/response. Ovo je prilično komplikovano jer moramo stalno da pratimo njegovu vrednost.
- **Cookies**
  - Kolačić je informacija koju Web server šalje u zaglavlju odgovora, a čuva se kod klijenta. Kada klijent ponovo pošalje zahtev, zaglavlju zahteva se dodaje kolačić i on se koristi za praćenje sesije. Međutim, klijent može i da isključi podršku za kolačiće.
- Session Management API
  - Podržava dva prethodna rešenja

Upravljanje sesijom

- Sesija je zajednička za sve servlete u aplikaciji kojoj klijent pristupa
- Objektu klase **javax.servlet.http.HttpSession** koja identifikuje sesiju pristupa se preko metoda **request.getSession()**

HttpSession

Metod	Informacije
getId()	Jedinstveni identifikator sesije
getLastAccessedTime()	Poslednji trenutak u kome je klijent poslao zahtev povezan sa sesijom
getCreationTime()	Trenutak u kome je sesija kreirana
getMaxInactiveInterval()	Maksimalni vremenski interval (u sekundama) koliko servlet kontejner održava sesiju otvorenom između zahteva
getAttribute(), setAttribute()	Objekti povezani sa sesijom
isNew()	Provera da li je sesija kreirana
invalidate()	Poništava sesiju i sve objekte povezane sa njom

Upravljanje greškama u servletima

- Pošto čitač razume samo HTML, kada aplikacija baci izuzetak, servlet kontejner ga obrađuje i generiše HTML odgovor
- Logika obrade je specifična za servlet kontejner
- Problem je u tome što prikaz greške korisniku ne znači ništa, a osim toga prikazuje klase i detalje o serveru što nije dobro sa tačke gledišta bezbednosti

Upravljanje greškama u servletima

- Sve izuzetke generisane u servletima treba pažljivo uhvatiti i obraditi, jer će u suprotnom klijent videti stack trace u prozoru čitača
- Servlet API podržava servlete Exception i Error Handler koje treba definisati u fajlu web.xml
- Svrha ovih servleta je da obrade izuzetak i prikažu HTML response koji je koristan, npr. tako što će prikazati stranu sa linkom na početnu stranu ili neke detalje na osnovu kojih će korisnik zaključiti šta nije u redu

Upravljanje greškama u servletima

- error-page treba zadati u web.xml
- Svaki element error-page treba da ima ili **error-code** ili **exception-type**. Servlet koji obrađuje grešku zadaje se u elementu **location**
- JSP strana takođe može da bude exception handler, samo treba navesti lokaciju jsp fajla umesto servleta

Upravljanje greškama u servletima

Primer: ako aplikacija baci grešku 404 ili ServletException, njih će obraditi servlet AppExceptionHandler

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <display-name>ServletExceptionHandling</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>404</error-code>
    <location>/AppExceptionHandler</location>
  </error-page>

  <error-page>
  <exception-type>javax.servlet.ServletException</exception-type>
  <location>/AppExceptionHandler</location>
  </error-page>
</web-app>
```

Upravljanje greškama u servletima

- Ako sve moguće izuzetke treba obraditi u jednom exception handleru, tip izuzetka treba da bude Throwable

```
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/AppExceptionHandler</location>
</error-page>
```

- Ako ima više error-page elemenata, npr. jedan za Throwable i drugi za IOException, a desi se izuzetak tipa IOException, on će biti obrađen error handlerom za IOException

Servleti filtri

- Filter je specijalna vrsta servleta koja dinamički presreće zahteve pre nego što stignu do servera, i(li) odgovore pre nego što stignu do klijenta i transformiše informacije u njima
- Najvažnija prednost filtara je to što se mogu dodati u postojeće aplikacije, a da se one ne moraju ponovo kompajlirati (pluggable)
- Filter je servlet koji implementira interfejs **javax.servlet.Filter**
- Umesto u metodama doGet() ili doPost(), filter svoje zadatke obavlja u metodi **doFilter()**

Filtri

- Filtri se mogu koristiti za nekoliko važnih uloga:
  - Pakovanje zadataka koji se često obavljaju u klasu koja se može uvek pozvati
  - Formatiranje podataka koji se šalju klijentu
  - Autorizacija i blokiranje zahteva
  - Logging i auditing (npr. beleženje request parametara u log fajl)
  - Komprimovanje response podataka koji se šalju klijentu
- Filtri i servleti su potpuno nezavisni; filtar se dodaje i uklanja samo iz fajla web.xml

Filtri

- Filtri se mogu povezivati u FilterChain; servlet kontejner konstruiše lanac filtara prema redosledu kojim se filtri pojavljuju u deployment descriptoru (fajlu web.xml)
- Da bi filter mogao da pristupi inicijalizacionim parametrima, prosleđuje mu se objekat FilterConfig u njegovoj metodi init()
- Filter može da pristupi objektu ServletContext preko objekta FilterConfig i da tako učita sve resurse potrebne za zadatke filtriranja

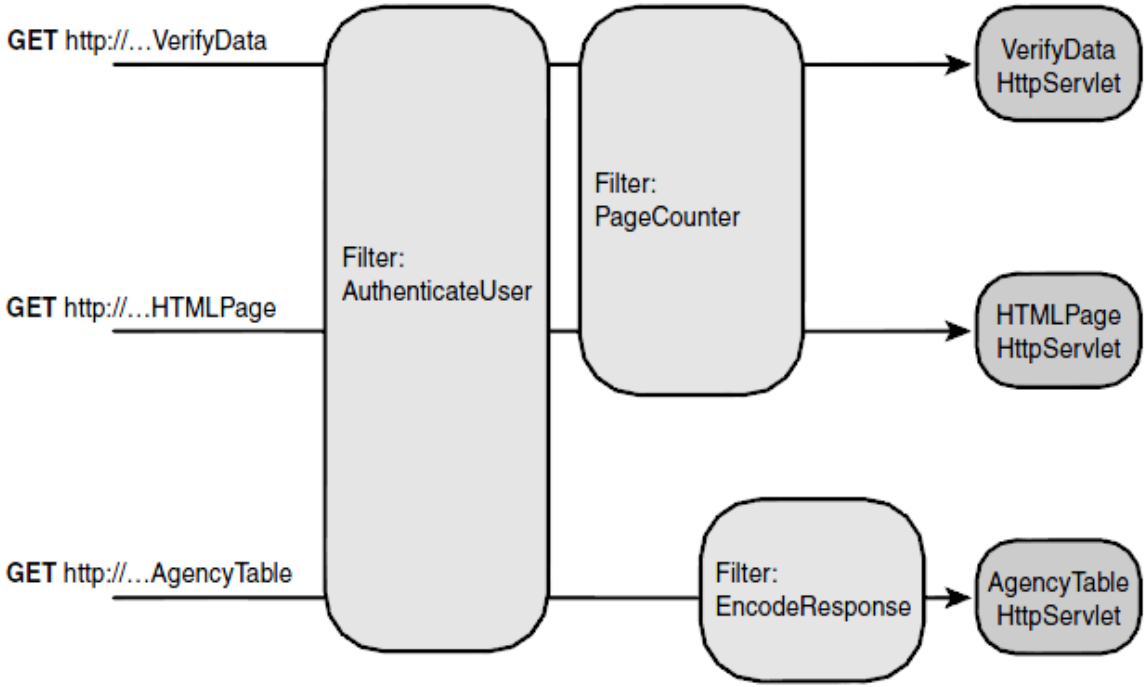
Filtri

- Kada filter obavi svoj zadatak, mora da pozove metod doFilter() sledećeg filtra u lancu
- Ako se radi o poslednjem filtru u lancu, servlet kontejner će pozvati resurs na kraju lanca, tj. običan servlet

```
public void doFilter(ServletRequest req, ServletResponse res,
➥FilterChain chain) throws IOException, ServletException {
    ...  // filter code
    chain.doFilter(req, res);  // call the next filter in the chain
}
```

Filtri

- Pošto se filtri dodaju aplikaciji u deploy fazi, jedan filter je moguće primeniti na jedan servlet ili više njih



Deploy filtara

- U web.xml moraju se dodati odgovarajući tagovi

```
<filter>
    <filter-name>AuditFilter</filter-name>
    <display-name>AuditFilter</display-name>
    <description></description>
    <filter-class>AuditFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>AuditFilter</filter-name>
    <servlet-name>HTMLPage</servlet-name>
  </filter-mapping>
```