Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

**INGINERIE SOFTWARE**

“ **Arhitectura aplicaţiilor WEB**”

**Grupa:** 443

**Studenţi:** Alexandra Agape

Andreea Mihaela Altiparmac

Andra Grigoraş

Anul universitar 2014-2015

      **CUPRINS:**

1. Arhitectura appleturilor – Andreea Mihaela Altiparmac

1.1 Arhitectura Applet-urilor si tehnologia Java Plug-in

1.2 Proprietatile Applet-urilor

1.3 Ciclul de viata al unui Applet

1.4 Avantaje si Dezavantaje Appleturi

2.Servleturi – Andra Grigoraş

2.1. O scurtă istorie a dezvoltării web

2.2. Rolul unui servlet

2.3. Ciclu de viață a unui servlet

2.4. Servleturile pentru World Wide Web

2.5. Scrierea codului pentru HttpServlet

2.6 Configurarea unui Servlet

3. JSPs – Alexandra Agape

3.1. Arhitectura JSP

3.1.1. Ciclul de viata a unei pagini JSP

3.2.Modele de acces JSP

3.3.Transmiterea cererilor

**1.Arhitectura appleturilor**

Un ***applet*** este un program Java ce este executat înauntrul unui browser Web, ca de exemplu Netscape Navigator sau Microsoft Internet Explorer.

Pentru ca applet-urile Java sunt programe care se obtin din Internet, se considera ca nu sunt de încredere si ca prezinta numeroase riscuri: virusarea fisierelor sistem ale unui calculator si extinderea virusilor si pe alte discuri, distrugerea sau divulgarea informatiilor existente în calculatorul în care se executa. Din motive de securitate, applet-urilor li se restrictioneaza operatiile pe care le pot efectua dupa cum urmeaza:

* Applet-urile nu pot citi sau scrie fisiere pe un disc al unui utilizator. Daca informatia trebuie salvata pe disc în timp ce un applet este în executie, memorarea informatiei trebuie facuta pe discul de pe care a fost încarcata pagina Web care contine applet-ul.
* Applet-urile nu pot realiza o conexiune pe retea spre un computer, altul decât cel de la care a fost încarcata pagina Web, cu exceptia îndreptarii browser-ului catre o noua locatie.
* Applet-urile nu pot folosi librariile dinamice sau partajate de la orice alt limbaj de programare. Desi Java poate folosi programe scrise în limbaje ca Visual C++, applet-urile nu se pot folosi de aceste caracteristici pentru ca nu exista nici o cale pentru o verificare adecvata a securitatii codului nonJava.

Java Plug-in, tehnologie inclusa în Java Runtime Environment ( JRE ) , permite unui applet Java rularea în browsere web pe desktop . Java Plug -in oferă proprietati puternice ale applet-urilor in cee ace priveste browser-ul web , si imbunatateste, în același timp, fiabilitatea generală și funcționalitatea applet-urilor. Java Plug -in executa una sau mai multe instante ale masinii virtuale Java prescurtat JVM, care se conectează înapoi la browser pentru interoperabilitate deplină cu paginile web. Această schimbare arhitecturala oferă multe avantaje și caracteristici cum ar fi urmatoarele:

* JVM-ul pe care ruleaza applet-urile este izolat de browserul web la nivelul sistemului de operare.Daca pe parcursul rularii appletu-ului apar erori, scopul Java Plug-in-ului este acela de a detecta si rezolva aceste erori, fara a face modificari la nivelul browserului web.
* Java Plug-in-ul are capabilitatea de a lansa applet-urile direct din fisierele JNLP (“Java Network Launch Protocol”).
* Legatura dintre JavaScript-ul din browserul web si limbajul de programare Java, este compatibila si ofera o oarecare fiabilitate, performanta si portabilitate pentru applet-uri.
* Spatiul de utilizare al applet-urilor este la fel de mare ca spatial ocupat de liniile de comanda din aplicatii.
* Fiecare applet in parte poate avea o versiune de JRE (“Java Runtime Environment”) diferita.
  1. **Arhitectura Applet-urilor si tehnologia Java Plug-in**

Prin tehnologia Java Plug-in, applet-urile nu sunt rulate in procesul JVM-ul din interiorul browserului web, ci sunt rulate in procese separate. Asadar, acelasi process JVM poate fi partajat intre mai multe applet-uri, sau applet-urile pot fi rulate in procese diferite. Applet-urile pot sugera o versiune specifica de JRE si mai pot sugera si executarea lor in procese JVM separate.

Arhitectura acestora prezinta anumite avantaje dupa cum urmeaza:

* Applet-urile care sugereaza o anumita versiune de JRE pot rula simultan (in acelasi timp)
* Prin applet-uri se pot specifica parametrii de initiere a JRE-ului, precum spatial de utilizare (“heap space”)
* Un applet poate rula intr-o versiune noua de JRE doar daca indeplineste conditiile : versiunea JRE sugerata de applet trebuie sa fie una corecta si parametrii de initializare a JRE-ului trebuie sa satisfaca cerintele applet-urilor.

In general, tehnologia Java Plug-in executa applet-urile intr-o versiune specifica de JRE existenta. Cand se genereaza o cerere de rulare a unui applet, acesta se va executa de la inceput in cea mai recenta versiune de JRE.

Java Plug-in-ul are capabilitatea de generare a mai multor fisiere (“threads”). Pentru fiecare applet, Java Plug-in creeaza un fisier separate, insa applet-urile in sine pot fi executate pe mai multe fisiere (“multi-threaded”).

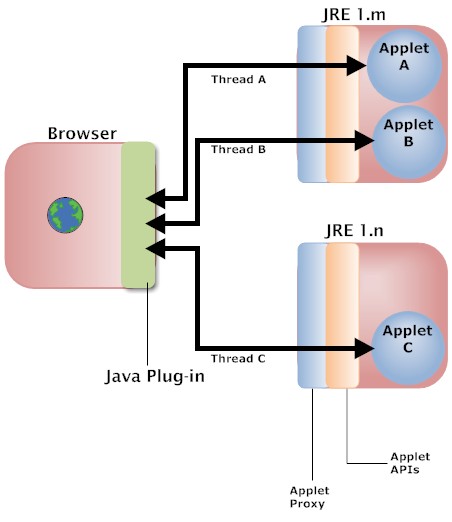


Figure 1 Multi threading

Figura de mai jos descrie comportamentul dintre Interpretorul JavaScript, Java Plug-in si appleturi:

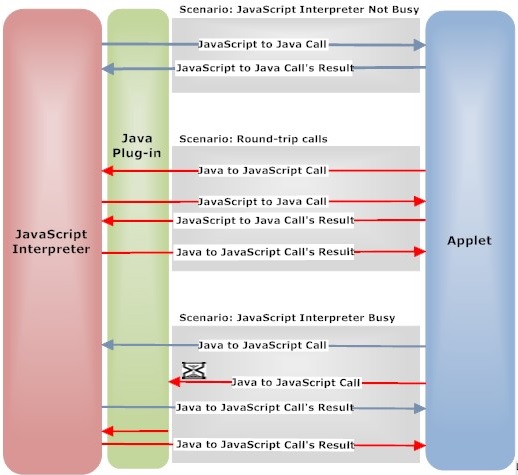


Figure 2 Legatura dintre JavaScript Interpreter, Java Plug-in si Applet

In cazul in care Interpretul JavaScript este inactive (pe modul “idle”), Java Plug-in-ul executa un script Java pentru apelat pe firul “ocupat” al applet-ului. Cand apelul Java – JavaScript este in executare si un apel JavaScript – Java este terminat (facut), executia in continuare se produce pe acelasi fir pe care se face apelul.

* 1. **Proprietatile Applet-urilor**

Un applet ruleaza intr-un “sandbox”, scopul acestuia fiind acela de a indeparta interactiunea applet-urilor cu sistemul userului, decat daca acesta este autorizat.Pentru a obtine autorizarea (accesul utilizatorului la applet-uri), acestea trebuie sa ceara anumite permisii cand sunt lansate si utilizatorul la randul lui, trebuie sa accepte rularea unui applet. Aceste permisii sunt utile pentru:

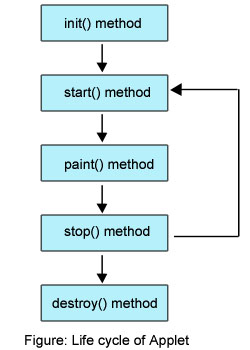
* Printare
* Accesarea sistemului
* Accesarea cookiurilor “cookies” ale browserului web
* Accesare unor alte resurse ale sistemului

Asadar, putem afirma faptul ca fara a avea permisie, un applet nu va putea avea nici un acces la sistemul utilizatorului, de aceea intotdeauna va avea nevoie de permisiuni.

* 1. **Ciclul de viata al unui applet**

Toate applet-urile au un set de metode prin care este descris mecanismul de baza al acestora. Acesta consta in controlarea executiei applet-urilor si in modul in care interactioneaza browserul cu applet-ul. Acest set de metode sunt urmatoarele:

* **Init()**
* **Start()**
* **Stop()**
* **Destroy()**
* **Paint()**



Se spune ca fiecare applet se afla intr-una din starile urmatoare:

1. Starea de nastere (“New Born state”)
2. Starea de functionare (“Running state”)
3. Starea de “idle” (“Idle state”)
4. Starea de moarte/ de distrugere (“Dead/Destroyed state”)

Figura de mai jos va descrie ciclul de viata al unui applet cu tot cu setul de metode:

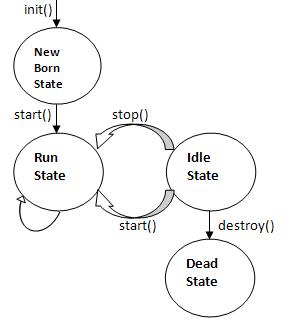


Figure Ciclul de viata al unui applet

1. Starea de nastere

-Se spune ca un applet intra in faza de initializare atunci cand acesta este pentru prima data incarcat in browser prin apelarea functiei init().

-Functia init() este apelata o singura data in ciclul de viata, adica la inceputul initializarii.

-In aceasta etapa, se pot crea obiecte (“Objects”) necesare applet-urilor, se seteaza valorile initiale si se pot incarca imagini/ fonturi.

2.Starea de functionare

-Dupa etapa de initializare, aceasta stare se va declansa automat prin apelarea functiei start().

-Starea de functionare poate fi atinsa din starea de idle atunci cand applet-ul este reincarcat.

-Metoda start() poate fi apelata de mai multe ori, atunci cand applet-ul trebuie pornit sau restartat.

3.Starea de pauza “idle”

-Starea de pauza va duce la oprirea temporara a executiei applet-ului.

-Applet-ul ajunge in starea aceasta atunci cand appletul executat current este minimizat sau cand -utilizatorul trece pe o alta pagina. La acest punct al etapei se va apela functia de stop().

-Applet-ul poate trece din “idle” in starea de executie/ de pornire.

-Functia stop() , ca si functia start(), poate fi apelata de mai multe ori in cadrul executiei.

4.Starea de distrugere/moarte

-Cand programul applet-ului se termina, functia de distrugere destroy() va fi apelata si prin -aceasta, applet-ul va fi condus in starea de moarte/distrugere.

-Functia destroy() va fi apelata o singura data, la finalul ciclului de viata al unui applet.

-In aceasta faza, applet-ul va fi scos/eliminat din memorie.

**Statusul unui Applet:**

|  |  |  |
| --- | --- | --- |
| **Status Applet** | **Valoare Status** | **Inteles** |
| LOADING | 1 | Applet-ul se executa… |
| READY | 2 | Applet-ul s-a incarcat total si este gata pentru a primii apeluri JavaScript |
| ERROR | 3 | Eroare la executie applet |

**1.4 Avantaje si Dezavantaje appleturi**

Applet-urile Java sunt încărcate pe un client atunci când utilizatorul vizitează o pagină ce conține un applet. Modelul de securitate din spatele applet-uri Java a fost conceput cu scopul de a proteja utilizatorul de applet-urile asa numite “malware-uri”.

Applet-urile sunt fie applet-uri “sandbox” sau applet-uri “privilegiate”. Cele “Sandbox” sunt permit doar un set de operațiuni, desfasurate în condiții de siguranță. Applet-urile privilegiate pot rula în afara sandbox-ului de securitate și au capacități extinse de acces client.

Applet-urile care nu sunt semnate sunt limitate in ceea ce priveste sandbox-ul de securitate, și poate rula doar dacă utilizatorul acceptă acest applet. Applet-urile care sunt semnate printr-un certificat eliberat de o autoritate de certificare recunoscuta, poate rula fie numai în sandbox, sau pot solicita permisiunea de a rula în afara sandbox-ului. În ambele cazuri, utilizatorul trebuie să accepte certificatul de securitate al applet-ului, în caz contrar applet-ul este blocat si nu va mai putea functiona.

**Applet-uri “Sandbox”**

Acestea sunt restrictionate de securitatea sandbox-ului si prezinta urmatoarele proprietati:

* Pot face conexiuni de rețea de la gazda si portul de la care provin. Protocoalele trebuie să se potrivească, iar în cazul în care un nume de domeniu este folosit pentru a încărca applet-ul, numele de domeniu trebuie să fie utilizat pentru a se conecta înapoi la gazdă, nu la adresa IP.
* Ele pot afișa cu ușurință documente HTML folosind metoda showDocument a clasei java.applet.AppletContext.
* Ele pot invoca metode publice ale altor applet-uri, pe aceeași pagină.
* Applet-urile care sunt încărcate de la sistemul de fișiere local (de la un director CLASSPATH-ul utilizatorului) nu prezinta niciuna dintre restricțiile pe care le au applet-urile încărcate în rețea.

Applet-urile “sandbox” nu pot realiza urmatoarele:

* Ele nu pot accesa resurse client, cum ar fi sistemul de fișiere local, fișierele executabile, sistemul clipboard, și imprimantele.
* Nu se poate conecta la orice server tert si nici nu pot prelua resurse de la acesta.
* Nu pot incarca librarii native.
* Nu pot schimba “SecurityManager”.
* Nu pot crea o clasa “ClassLoader

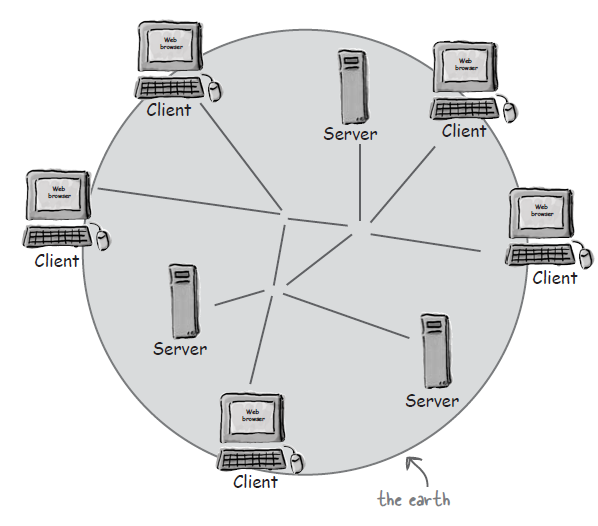
**Applet-uri privilegiate**

Applet-urile privilegiate nu au restricțiile de securitate pe le prezinta applet-urile “sandbox” și pot rula în afara sandbox-ului de securitate.

**2. Servleturi**

**2.1. O scurtă istorie a dezvoltării web**

În prezent, toată lumea vrea un site web .Web-ul este format din nenumăraţi clienţi (folosind browsere, cum ar fi Mozilla sau Safari) şi servere conectate prin cabluri şi reţele wireless. Un browser web permite utilizatorului să ceară o resursă. Serverul web primeşte cererea, găseşte resursă şi returnează ceva pentru utilizator. Când vorbim despre clienţi, ne referim ,de obicei, la utilizator uman şi la aplicaţia browser. Browser-ul este o piesă de software care ştie cum să comunice cu serverul şi să interpreteze codul HTML şi să transmită pagină web pentru utilizator.



*Figura 4*

Deoarece paginile simple web scrise în HTML sunt statice, iar utilizatorii din ziua de astăzi vor site-uri care să fie dinamice, interactice şi adaptive, dezvoltatorul web trebuie să folosească servleturi şi JSP-uri.

Pe scurt, servleturile şi JSP-urile ,,transformă” site-ul web într-o aplicatie web.

Servleturile sunt răspunsul tehnologiei Java la Common Gateway Interface (CGI). Acestea sunt programe care se execută pe un server Web, ele acţionează ca o etapă intermediară între o cerere care vine de la un browser Web şi alt client HTTP şi bazele de date sau aplicaţiile de pe serverul HTTP. Sarcina lor este:

1. Să citească toate datele trimise de către utilizator. Aceste date sunt, de obicei, introduse într-un form pe o pagină web, dar ar putea, de asemenea, să provină dintr-un applet Java sau un program client HTTP personalizat.

2. Să se uite la orice alte informaţii cu privire la cererea care este încorporată în cererea HTTP. Aceste informaţii includ detalii despre capacităţile browser-ului, cookie-uri, numele host-ului care este solicitat de client şi aşa mai departe.

3. Să genereze rezultatele. Acest proces poate necesita legătura cu o bază de date, executarea unei RMI(Remote Method Invocation) sau un apel CORBA, invocând o aplicaţie de calcul a răspunsul direct.

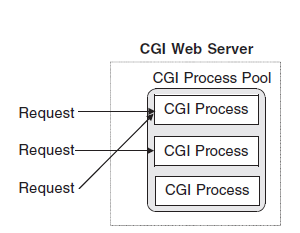
4. Să pună rezultatele în interiorul unui document. În cele mai multe cazuri, acest lucru implică încorporarea informaţiile privilegiate într-o pagină HTML.

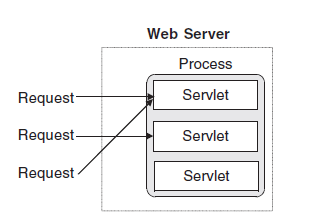
5.Să seteze corespunzător parametrilor de răspuns HTTP.

Această înseamnă a spune browser-ul ce tip de document este returnat (de exemplu, HTML), stabilind cookie-uri şi parametrii cache,şi alte astfel de sarcini.

6.Să trimită documentul înapoi la client. Acest document poate fi transmis în format text (HTML), format binar (Imagini GIF), sau chiar într-un format comprimat că gzip.

Mediile servlet şi JSP extind nevoia de suport de bază Java. Orice calculator care rulează JSP sau servlet-uri are nevoie de un container. Un container este o componentă software responsabilă cu încărcarea, executare şi descărcare servlet-urilor şi JSP-urilor.

 În lumea Java, servlet-urile sunt proiectate pentru a rezolva problemele CGI (Common Gateway Interface) şi a pentru a crea un mediu robust server-side pentru dezvoltatorii web. Asemănător cu CGI, servlet-urile permit că o cerere să fie procesată de către un program şi lasă acelaşi program să producă un răspuns dinamic. Servlet-urile au definit ,în plus, un ciclu eficient de viaţă, care a inclus posibilitatea de a utiliza un singur proces pentru gestionarea tuturor cererilor. Acest lucru a eliminat efectul overhead multi-proces a CGI şi a permis procesului principal să partajeze resursele între mai multe servlet-uri şi mai multe cereri.

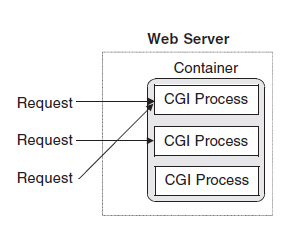


*Fig.5Diagrama unui server web cu suport servlet Fig.6Diagrama unui server web cu suport CGI*

Diagrama servlet este similară cu cea a celei de a două generaţie CGI, dar se observă că toate servlet-urile sunt administrate de un proces principal sau de un program părinte. Aceasta este una dintre soluţiile pentru eficientizarea servlet-urilor. Cu un design eficient şi suport Java prin platforme ce pot fi utilizate pe diferite tipuri de calculatoare sau cu diferite pachete software, servletul a rezolvat reclamţiile ale lui CGI şi a devenit rapid o soluţie populară pentru funcţionalitatea pe partea de server dinamic.

Performanţa Servlet-urilor poate fi atribuită direct containerelor servlet. Un container servlet este o componentă de software care gestionează ciclul de viaţă al unui servlet.Software-ul unui container este responsabil cu interacţiunea cu un server web pentru a procesa o cerere şi cu trecerea la un servlet pentru a primi răspuns. Definiţia oficială a unui container este descrisă complet de specificaţiile JSP şi Servlet. Spre deosebire de cele mai multe tehnologii, specificaţiile servlet definesc numai un standard pentru funcţionalitatea unui container. Acest lucru permit dezvoltatorilor software diferite opţiuni de implementare.

Cu cele prezentate mai sus, diagramă anterioară a unui servlet poate fi mai bine desenată folosind un container pentru a reprezenta procesul unic care creează, gestionează şi distruge fire de execuţie cu care rulează servlet-urile pe un server web. Trebuie reţinut faptul că acest lucru poate sau nu poate fi un proces fizic separat.



*Fig.7 Server web folosind container servlet*

Ceea ce în figura 7 nu se observă este faptul că, în unele cazuri, un container acţionează mai degrabă că un serverul web decât un modul la un server existent. În aceste cazuri, serverul web şi containerul din figura 6 sunt în esenţă acelaşi lucru.

**2.2. Rolul unui servlet**

Java Servlets sunt o soluţie eficientă şi puternică pentru creerea de conţinut dinamic pentru Web. De-a lungul ultimilor ani servletul a devenit componenta fundamentală Java server-side. Puterea din spatele servleturilor vine de la folosirea java ca o platformă de interacţiune cu un container de servlet. Platforma Java oferă un dezvoltator servlet cu un API robust, orientate obiect pro- program, neutralitate platformă, tipuri stricte, precum şi toate caracteristici de securitate ale JVM. Însumând acestea, un container servlet prevede managementul ciclului de viaţă , ca un singur proces de a împărtăşi şi de a gestiona resurse, aplicarea pe scară largă şi interacţiune cu un server Web. Împreună, această funcţie face Servlets o tehnologie dorită de dezvoltatorii Java server-side.

Servleturile fac întotdeauna parte dintr-un proiect mai amplu numit aplicaţie web. O aplicaţie web este o colecţie completă de resurse pentru un site web. Nimic nu opreşte o aplicaţie web să conţină unu, mai multe sau niciun servlet. Aplicaţii web şi fişierele de configurare pentru acesta sunt găsite în specificaţiile servlet.

Scopul principal al specificaţiilor servlet este de a defini un mecanism solid pentru trimiterea conţinutului unui client astfel cum este definită de modelul client / server. Servleturile sunt cele mai populare, ele sunt folosit pentru a genera conţinut dinamic pe web şi au suport nativ pentru HTTP.

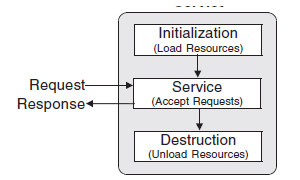
Filtrele au fost introduse oficial în specificaţiile Servlet 2. Un filtru oferă o metodă abstractă de manipulare a unui client înainte de cerere şi / sau un răspuns de a ajunge de fapt obiectivul cererii. Filtrele completează foarte mult servleturile şi sunt utilizate în mod obişnuit pentru lucruri, cum ar fi autentificarea, compresia conţinutului şi conectarea.

Servleturile utilizează deja caracteristicile de securitate oferite de Java Virtual Machine, dar specificaţiile servlet defineşte un mecanism de control al accesului la resurse într-o aplicaţie web.

Una dintre cele mai bune caracteristici ale unui servlet este abilitatea de a dezvolta conţinut pentru aproape orice limbaj. O mare parte din această funcţionalitate vine direct de la platforma Java pentru internaţionalizare şi localizare. Servlet API păstrează această funcţionalitate şi poate fi uşor de utilizat pentru a crea un conţinut în cele mai multe dintre limbile existente.

**2.3. Ciclu de viață a unui servlet**

Pentru a înţelege funcţionalitatea minimă a unui servlet este necesar să se cunoască ciclu de viaţă a acestuia.

.

*Fig.8 Ciclul de viață a unui servlet*

Ciclu de viaţă reglementează mediul multi-thread pe care servleturile le administrează şi oferă o perspectivă la unele dintre mecanismele disponibile unui dezvoltator pentru partajarea resurselor serverelor.

Ciclul de viaţă a unui servlet este cauza principală prin care a depăşit CGI-urile tradiţionale. Spre deosebire de CGI care au un ciclu de viaţă de unică folosinţă, servleturile au o viaţă cu trei faze: iniţializare, servire şi distrugere. De obicei iniţializarea şi distrugerea sunt efectuate o dată şi serviciile sunt efectuate de mai multe ori.

Iniţializarea este prima fază a ciclului de viaţă a unui servlet şi reprezintă crearea şi initializarea resurselor ce pot fi necesare cererilor de servicii. Toate servleturile trebuie să implementeze interfaţă javax.servlet.Servlet. Această interfaţă defineşte metoda init () pentru a se potrivi cu faza de iniţializare a unui ciclu de viaţă. Atunci când un container încarcă un servlet, el invocă metoda init () înainte de a servi orice solicitare.

Faza de servire a ciclului de viaţă servlet reprezintă totalitatea interacţiunilor cu cererile până când servletul este distrus. Interfaţă servlet corespunzătoare fazei de servire a ciclului de viaţă este metoda service(). Metoda service () a unui servlet este invocată o dată cu o cerere şi este responsabilă pentru generarea răspunsului la aceea cerere. Cerinţă unui servlet pentru metoda serviciu () este că metoda să aibă doi parametri: un javax.servlet.ServletRequest şi un obiect javax.servlet.ServletResponse. Aceste două obiecte reprezintă o cerere a clienţilor pentru resursele dinamice şi răspunsul servletului pentru client. În mod implicit un servlet este mulţi-thread, ceea ce înseamnă că de obicei doar o instanţa a unui servlet este încărcată de un container JSP la un moment dat. Iniţializarea se face o dată, şi fiecare solicitare ,după ce este tratată concomitent de thred-urile, execută metoda service ().

Faza de distrugere din ciclului de viaţă a unui servlet constă în faptul că un servlet este scos din uz de un container. Interfaţa servlet defineşte metodă destroy () care îi corespunde fazei ciclului de terminare a vieţii. De fiecare dată când un servlet este pe cale să fie scos din uz, un container apelează metoda destroy() , care permite servletului să îşi termine treaba şi să şteargă orice resursă care a fost creată de el.

Prin utilizarea corectă a fazelor de iniţializare, servire, precum şi de distrugere, un servlet poate gestiona eficient resursele aplicaţiei. În timpul iniţializării servletul încarcă tot ce are nevoie să utilizeze pentru a servi cererile. Resursele sunt apoi disponibile pentru utilizare imediată în timpul fazei de servire şi pot fi şterse în faza de distrugere. Aceste trei evenimente formează ciclul de viaţă servlet, dar, în practică, există mai multe metode pentru care un dezvoltator web trebuie să îşi facă griji. Conţinutul de pe web este accesat în principal prin protocolul HyperText Transfer (HTTP). Un servlet de bază nu ştie nimic despre HTTP, dar pentru aceasta există o implementare specială, care este proiectat special: servlet javax.http.HttpServlet.

**2.4. Servleturile pentru World Wide Web**

Când termenul servlet este menţionat, acesta implică aproape întotdeauna faptul că un servlet este o instanţa de HttpServlet . Explicaţia este simplă: Hypertext Transfer Protocol (HTTP) este utilizat pentru majoritate tranzacţiilor de pe World Wide Web - fiecare pagină web pe care o vizitaţi este transmisă prin HTTP, prin urmare http: // prefix. HTTP nu este cel mai bun protocol existent, dar HTTP funcţionează şi este deja utilizat pe scară largă. Suport Servlet pentru tranzacţii HTTP vine sub formă de clasă javax .servlet. http .HttpServlet.

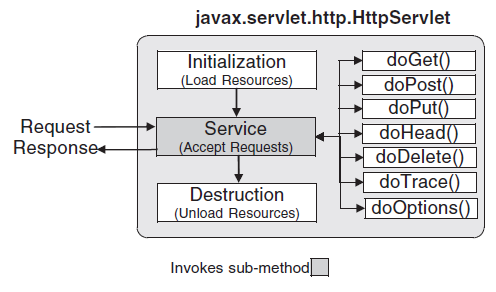
Mulţi dezvoltatori nu înţeleg complet HTTP, care este esenţial pentru a înţelege pe deplin HttpServlet. HTTP este un protocol simplu. Protocolul se bazează pe un client, de obicei, un browser Web, pentru a face o cerere şi un server pentru a trimite un răspuns. Conexiunile durează timpul necesar pentru o singură tranzacţie. O tranzacţie poate constă în una sau mai multe perechi cerere / răspuns. De exemplu, un browser va trimite o cerere de pagină HTML urmată de mai multe solicitări pentru fiecare imagine de pe acea pagină. Toate aceste cereri şi răspunsuri se vor face în aceeaşi conexiune. Conexiunea va fi apoi distribuită la sfârşitul ultimului răspunsului. Întregul proces este relativ simplu şi are loc de fiecare dată un browser solicită o resursă de la un server HTTP.

Prima parte a unei tranzacţii HTTP este atunci când un client HTTP creează şi trimite o solicitare la un server. O cerere HTTP în formă sa cea mai simplă este nimic mai mult decât o linie de text în care se specifică ce resurse ar dori un client să preia. Linia de text este împărţită în trei părţi: tipul de acţiune, sau metodă, pe care clientul ar dori să o facă; resursă pe care clientul ar dori să o acceseze; şi versiunea protocolului HTTP care este utilizat. De exemplu: GET /index.html HTTP / 1.0. Exemplul este o cerere HTTP complet validă. Primul cuvânt, GET, este o metodă definită prin HTTP pentru a cere unui server o resursă specifică; / index. html este resursă care se solicită de la server; HTTP/ 1. 0 este versiunea de HTTP care este utilizată. Când orice dispozitiv prin HTTP vrea să obţină o resursă de la un server, s-ar folosi ceva similar cu linia de mai sus.

HTTP foloseşte antete să trimită meta-informaţii cu o cerere sau cu un răspuns. Un antet este o pereche de forma nume: valoare . Anteturile tipice de răspuns includ: descrieri despre conţinut, lungimea conţinutului, o time-stamp, informaţii despre server, şi dată la care conţinutul a fost ultima oară schimbat. Aceste informaţii ajută clientul să îşi de-a seama ce este trimis, cât de mare este, iar dacă datele sunt mai noi decât un răspuns văzut anterior. O cerere HTTP va conţine întotdeauna câteva antete. Anteturile pentru cereri standard constau în detaliile utilizatorului şi formatele sale preferate, limbile, şi codificarea de conţinut pentru a primi. Aceste headere transmit serverului ce clientul este şi modul în care acestea ar preferă să primească informaţii.

**2.5. Scrierea codului pentru HttpServlet**

Anterior, am menţionat că servleturi au un ciclu de viaţă divizat în trei părţi: iniţializare, servire şi distrugere. Un obiect Httpservlet are acelaşi ciclu de viaţă, dar cu câteva modificări pentru protocolul HTTP. Implementarea metodei de serviciu a obiectul Httpservlet, care este apelată în timpul fiecărei cerere de servire, solicită una dintre cele şapte metode ajutătoare diferite. Aceste şapte metode corespund direct cu cele şapte metodele HTTP şi sunt denumite după cum urmează: doGet(), doPost(), doPut(), doHead(), doOptions(), doDelete () şi doTrace (). Metodă ajutoare adecvată este invocată pentru a se potrivi tipului de metodă a unei cereri HTTP. Ciclul de viaţă HttpServlet poate fi ilustrată după cum se arată în figura de mai jos.



*Fig.9 Ciclul de viață a unui HttpServlet*

Scrierea codului unui HttpServlet este simplă. Clasa javax.servlet.http.HttpServlet are grijă de administrarea părţilor redundante a cererilor şi a răspunsurilor HTTP, dar are nevoie de un programator pentru a înlocui metodele care trebuiesc personalizate. Manipularea unei anumite cereri şi răspunsul se face prin două obiecte: javax.servlet.http.HttpServletRequest şi javax.servlet.htt.HttpServletResponse. Ambele obiecte sunt transmise că parametri când se apelează metodele de servire HTTP.

Servletul nu este o aplicaţie completă Java. Servleturile fac parte dintr-o aplicaţie Web care conţine un container administrator. Folosind un servlet pentru a genera răspunsuri dinamice implică atât crearea unui servlet şi implementarea să pentru utilizare în aplicaţia Web. Implementarea unui servlet nu este atât de uşoară, dar nici intuitivă cum s-ar putea crede. Spre deosebire de o resursă statică, un servlet nu este pur şi simplu plasată în directorul rădăcina al aplicaţiei web. Fişierul clasei servlet se slaveaza în directorul aplicaţiei /WEB-INF/classes cu toate celelalte clase Java. Pentru că un client să acceseze un servlet, o adresa unică URL sau set de adrese URL, trebuie că servletul să fie declarat în Web Application Deployment Descriptor. Descrierea web.xml se bazează pe elemente noi că : servlet şi servlet-mapping ce au trebuit introduse pentru a fi utilizate în web.xml Elementul servlet este utilizat pentru a defini un Servlet care ar trebui să fie încărcate de o aplicaţie Web . Element servlet-mapping este folosit pentru a mapă un Servlet la o anumită adresa URL sau la un set de adrese URL. Ambele elemente au sub-elemente utilizate pentru a le descrie.

**2.6 Configurarea unui Servlet**

Uneori este necesar să se prevadă informaţii iniţiale pentru configurarea unui servlet. Informaţiile pentru configurarea unui servlet constau dintr-un şir sau un set de valori string incluse declararea web.xml a servletului. Această funcţionalitate permite unui Servlet să aibă parametri iniţiali specificaţi în afară codului compilat şi aceştia să poată fi schimbaţi fără a fi nevoie să se recompileze servletul. Fiecare servlet are un obiect asociat numit ServletConfig. Acest obiectiv este creat de container şi implementează interfaţă javax.servlet.ServletConfig. Tot el conţine parametrii de iniţializare. O referire la acest obiect se poate face prin apelarea metodei getServletConfig(). Obiectul ServletConfig oferă următoarele metode pentru a accesa parametrii iniţiali:

*1.getInitParameter (nume string)*

Metodă getInitParameter() returnează un obiect string care conţine valoarea parametrului de initializare sau valoarea nul dacă parametrul nu există.

*2.getlnitParameterNames ()*

Metodă getInitParameterNames() returnează numele parametrilor de iniţializare a unui Servlet ca o enumerare a obiectelor string sau o enumerare null dacă servletul nu are parametri de iniţializare.

Definirea parametrilor iniţiali pentru un servlet necesită utilizarea elementelor init-param, param-name şi param-value din .xml web. Fiecare element init-param defineşte un parametru iniţială şi trebuie să conţină numele parametrului şi valorile specificate de param-name şi param-value. Un servlet poate avea mai mulţi parametri iniţiali după cum este necesar. Informaţiile parametru iniţial pentru un anumit servlet ar trebui specificat în elementul servlet.

**3. Java Server Pages (JSP)**

JSP este o extensie java standard utilizat pentru a simplifica crearea și administrarea paginilor web dinamice. JSP ne permite să separam conținutul dinamic de pagini web de conținutul static de prezentare . O pagină JSP constă în tag-uri HTML și tag-uri JSP. Tag-uri HTML sunt folosite pentru a crea conținutul static al paginii și tag-urile JSP sunt folosite pentru a adăuga conținut dinamic la paginile web. Pagini JSP sunt compilate într-un Java Servlet de un traducător JSP.

**De ce sa folosim paginile JSP?**

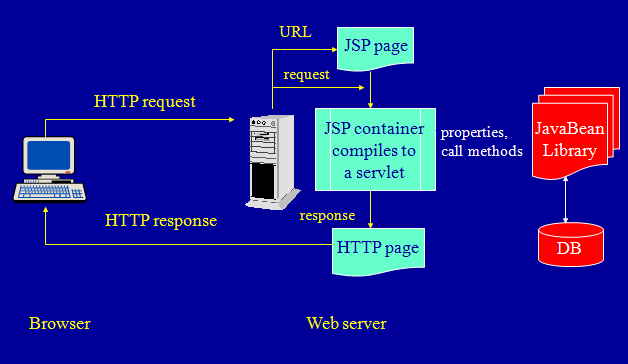
Construirea paginilor web dinamice folosind pagini JSP ofera urmatoarele beneficii:

* Paginile web create folosind pagini JSP sunt portabile si pot fi folosite pe mai multe platform, fara a face nicio modificare;
* Programarea in JSP este mai usoara ca cea in Servlet-uri;
* Paginile JSP sunt compilate automat de servere. Dezvoltatorii au nevoie sa compileze paginile JSP cand codul sursa al paginilor JSP este modificat;
* Cel mai important beneficiu al paginilor JSP este diferenta intre logica afacerii si logica prezentarii. Javabeans contine logica afacerii, in timp ce paginile JSP contin logica prezentarii. Aceasta diferenta face ca o aplicatie realizata in pagini JSP sa fie mai sigura si reutilizabila;

**Avantajele paginilor JSP**

* Paginile JSP combina usor template-uri statice, cum ar fi fragmente HTML sau XML;
* Inainte de executare, paginile JSP sunt compilate in servlet-uri, astfel incat putem actualize cu usurinta codul de prezentare;
* Daca pagina JSP este modificata, atunci nu este nevoie de recompilarea si realocarea proiectului;
* Este usor sa separam logica afacerii de logica prezentarii. Astfel, paginile JSP pot fi usor gestionate;
* Este usor de invatat si usor de implementat;
* Mediu JSP prevede compilare de pagini în mod automat;
* Servlet-urile nu pot fi accesate direct, in timp ce o pagina JSP poate fi accesata direct ca o simpla pagina HTML;

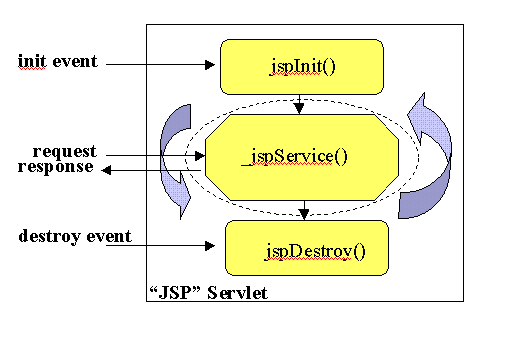
**3.1. Arhitectura JSP**



*Fig.10 Arhitectura JSP*

Așa cum se poate vedea în arhitectura de mai sus, browser web acceseaza direct pagina JSP. Paginile JSP interacționează cu JavaBeans , care reprezintă modelul de cerere. Când clientul trimite cererea de la o pagină JSP, răspunsul este trimis înapoi la client, în funcție de cererile care sunt invocate de cererea clientului. Dacă răspunsul necesită accesarea bazei de date, pagina JSP utilizează JavaBeans pentru a obține date din baza de date.

**3.1.1. Ciclul de viata a unei pagini JSP**

****

*Fig.11 Ciclu de viata*

Paginile JSP urmaresc fazele urmatoare:

* Traduc pagina JSP intr-un cod servlet;
* Compileaza servlet-ul cu codul byte;
* Incarca clasa servlet-ului;
* Creaza instanta servlet-ului;
* Apeleaza metoda jspInit;
* Apeleaza metoda \_jspService;
* Apeleaza metoda jspDestroy;

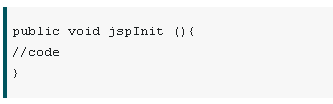
**Traducerea paginii JSP:**  în prima etapă, containerul web traduce fișierul JSP într-un fișier sursă Java care conține o definiție de clasă servlet. Containerul web valideaza corectitudinea paginilor JSP și fișierelor tag.

**Compilarea paginii JSP:** in cea de-a doua etapa, container-ul web compileaza codul sursa al servlet-ului intr-un fisier de clasa Java.

**Inacarc area clasei paginii JSP:** in a 3-a etapa, codul byte al clasai servlet-ului este incarcat in software-ul JVM al container-ului web folosind class loader.

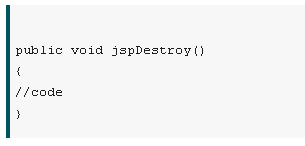
**Instanta servlet-ului:** in cea de-a 4-a etapa, container-ul web creaza o instanta a clasei servlet-ului.

**Initializarea paginii JSP:**  in cea de-a 5-a etapa, container-ul web initializeaza servlet-ul apeland metoda jspInit(). Metoda este apelata imediat dupa ce instanta ste creata. Este apelata o singura data in timpul ciclului de viata al paginii JSP. Pentru a efectua initializarea trebuie sa trecem peste metoda jspInit():



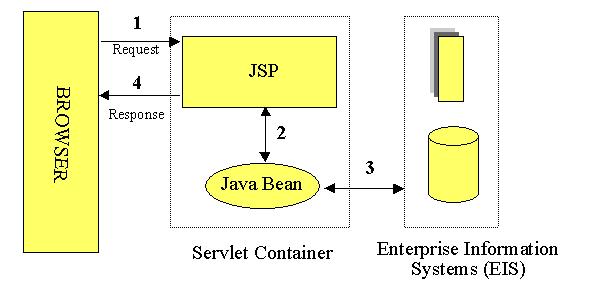
**Serviciul paginii JSP:** servlet-ul initializat poate deservi cererile. Container-ul web apeleaza metoda \_jspService() pentru a procesa cererea user-ului. Aceasta metoda este apelata in timpul ciclului de viata al paginii JSP. Aceasta metoda foloseste parametrii HttpServletRequest si HttpServletResponse.

**Distrugerea paginii JSP:**  cand container-ul web elimina instanta servlet-ului JSP, apeleaza metoda jspDestroy() pentru a permite paginii JSP sa poata curata resursele. Acesta este sfarsitul ciclului de viata al paginii JSP. Aceasta metoda poate fi scrisa astfel:



**3.2.Modele de acces JSP**

Specificatiile JSP sustin doua abordari filosofice, cunoscute ca arhitectura modelului 1 si a modelului 2, pentru aplicarea tehnologiei JSP. Aceste abordari difera prin locatia in care a fost realizata cea mai mare parte a prelucrarii cererii. Consideram arhitctura modelului 1:

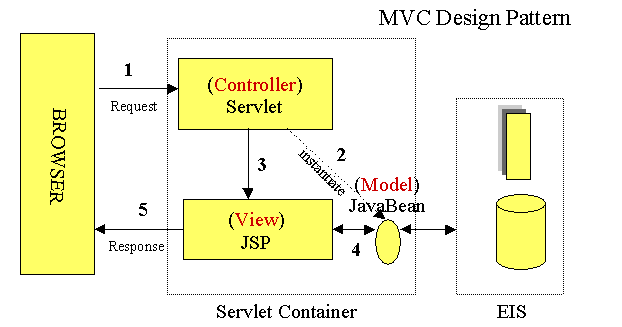


*Fig.12*

In arhitetcura modelului 1, cererea de intrare de pe un browser web este trimisa direct la pagina JSP, care este responsabila pentru procesarea ei. Nu există încă separarea prezentarii de conținut, deoarece toate datele de acces se realizează folosind beans.

Chiar daca arhitectura modelului 1 este potrivita pentru aplicatii simple, aceasta poate sa nu mearga pentru aplicatii complexe. Utilizarea fara discernamant a acestei arhitetecturi, de obicei duce la o cantitate semnificativa de scriplet-uri sau cod Java incorporata in pagina JSP, mai ales in cazul in acre exista o canttate semnificativa de procese de prelucrare care urmeaza sa fie efectuate.

Deși acest lucru nu poate părea a fi o problema prea mare pentru dezvoltatorii Java, acesta este cu siguranță o problemă dacă paginile JSP sunt create și menținute de designeri - care este, de obicei, norma pe proiecte mari. Un alt dezavantaj al acestei arhitecturi este faptul că fiecare dintre paginile JSP trebuie să fie responsabile individual pentru verificarea autentificari și securitate.



*Fig.13*

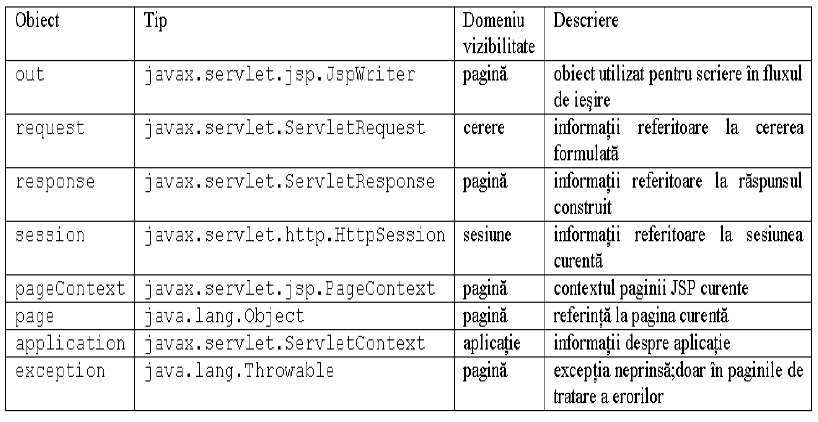
In arhitetcura modelului 2, prezentat mai sus, este o implementare “server-side” a modelului de design Model\View\Controller. Aici, prelucrarea este impartita intre componentele prezentarii si componentele frontale. Componentele prezentarii sunt paginile JSP care genereaza raspunsul HTML/XML care determina interfata cu utilizatorul atunci cand este transmisa de browser. Componentele frontale (cunoscute, de asemenea, ca si controlere) nu se ocupa de probleme de prezentare, ci mai degraba proceseaza toate cererile HTTP. Aici, ele sunt respinsabile de crearea oricaror obiecte sau a unor “beans” folosite de componentele prezentarii, precum si de a decide, in functie de actiunile utilizatorului,la care component putem trimite cererea. Componentele frontale pot fi implementate fie ca un servlet, fie ca o pagina JSP.

Avantajul aceste arhitecturi este ca nu exista nicio logica de procesare in cadrul componentei de prezentare, este pur si simplu responsabila pentru preluarea oricarui obiect sau “beans”, care poate au fost create in prealabil de catre operator, precum si pentru extragerea continutului dinamic. Prin urmare aceasta separare a prezentarii de continut duce la delimitare clara a rolurilor si reponsabilitatilor dezvoltatorilor si designerilor de pagini. Un alt avantaj al acestei abordari este ca aceste componente frontale prezinta un singur punct de intrare in aplicatie.

Pentru formularea raspunsului pentru navigatoarele Web se face apel la o serie de obiecte. Acestea sunt create in cadrul paginii JSP sau sunt predefinite si pot avea urmtoarele domenii de vizibilitte:

* Pagina(page) – obiectele sunt vizibile doar in cadrul paginii curente;
* Cerere(request) – obiectele pot fi accesate pe tot parcursul rezolvarii cererii:in pagina curenta, in paginile care sunt incluse si in paginile la care s-a realizat redirectarea;
* Sesiune(session) – obiectele vor fi disponibile de-a lungul intregii sesiuni;
* Aplicatie(application) – obiectele sunt vizibile in toata aplicatia si pentru toate sesiunile;

Tehnologia JSP pune la dispozitie programatorilor o serie de obiecte implicite:

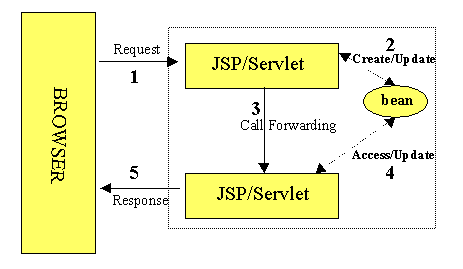


**3.3.Transmiterea cererilor**

Cu tag-ul <jsp:forward> puteți redirecționa cererea oricăreia JSP, servlet, sau pagina HTML statica în același context ca și pagina de invocarea. Acesta opreste efectiv prelucrare paginii curente în punctul în care are loc redirecționarea, deși toata procesarea de până la acel moment are încă loc:

<jsp:forward page="OricePagina.jsp" />

Pagina de invocare poate trece, de asemenea, parametrii resursei țintă „bean”prin plasarea lor în cerere, așa cum se arată în diagrama:



Un tag <jsp:forward> poate, de asemenea, avea subelemente jsp:param

care poate oferi valori pentru unele elemente din cerere utilizate în transmitere:

<jsp:forward page="<%= OricePagina %>" >

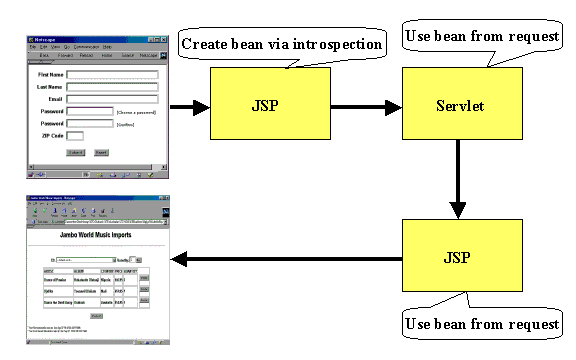
<jsp:param name="name1" value="value1" />

<jsp:param name="name2" value="value2" />

</jsp:forward>

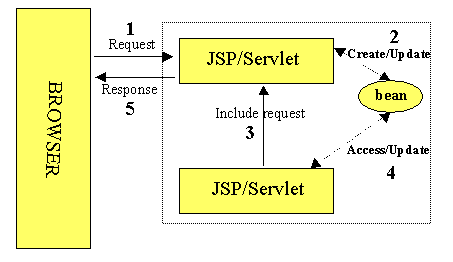
**Cererea de inlantuire**

Cerere de înlănțuire este o caracteristică puternica și poate fi utilizată pentru a introduce eficient pagini JSP și servlet în forme de prelucrare HTML, așa cum se arată în figura de mai jos:

****

**Cereri de includere**

Tag-ul <jsp:include> poate fi folosit pentru a redirecționa solicitarea la orice resursă statica sau dinamica, care este în același context ca și pagina de asteptare JSP. Pagina de asteptare poate trece, de asemenea, parametrii resursei țintă „bean” prin plasarea lor în cerere, așa cum se arată în diagrama:



De exemplu:

<jsp:include page="Orice.jsp" flush="true"/> nu numai ca permite paginii orice.jsp de a accesa orice “bean” plasat in cerere folosind un tag <jsp:useBean> , dar este intreodus in pagina de asteptare continutul dinamic produs de aceasta in punctual in care tag-ul <jsp:include> se realizeaza. Resusa inclusa nu poate stabili niciun antet HTTP care o opreste din a face lucruri cum ar fi setarea cookie-urilor, altfel o exceptie are loc.

Bibliografie:

* „Core Servlets and JavaServer Pages, Vol. 1”de Marty Hall, Larry Brown, publicată pe 01.01.2000
* „Head First Servlets and JSP, Second Edition”de Bryan Basham, Kathy Sierra, Bert Bates, publicată Martie 2008
* „Servlets and JavaServer Pages” de Jayson Falkner, Kevin Jones publicată Septembrie 2004
* <http://en.wikipedia.org/wiki/JavaServer_Pages>
* <http://java.cnam.fr/iagl/biblio/Serlvets%20&%20JSP%20-%20Falkner%20Jones.pdf>
* [http://www.dcc.fc.up.pt/~zp/aulas/0708/asw/geral/bibliografia/O%27Reilly%20-%20JavaServer%20Pages\_2nd%20Edition.pdf](http://www.dcc.fc.up.pt/%7Ezp/aulas/0708/asw/geral/bibliografia/O%27Reilly%20-%20JavaServer%20Pages_2nd%20Edition.pdf)
* <http://en.wikipedia.org/wiki/Applet>
* <http://en.wikipedia.org/wiki/Java_applet>
* <http://www.smartclass.co/2011/02/life-cycle-of-applet.html>
* <http://docs.oracle.com/javase/8/docs/technotes/guides/deploy/applet_dev_guide.html>
* “Java, a Beginner’s Guide, 5th Edition”

Figuri :

* Figurile 1,2 de pe google: applet architecture
* Figurile 3 si 4 din cartea „Java, a Beginner’s Guide, 5th Edition”
* Figura 4 din cartea „Head First Servlets and JSP, Second Edition”de Bryan Basham, Kathy Sierra, Bert Bates, publicată Martie 2008
* Figurile 5, 6, 7, 8, 9 din cartea „Servlets and JavaServer Pages” de Jayson Falkner, Kevin Jones publicată Septembrie 2004