Szakdolgozat

Rácz György

Debrecen

2009

Debreceni Egyetem
Informatika Kar

Java alapú web fejlesztő környezet ismertetése

Témavezető:

Dr. Almási Béla

Egyetemi docens

Készítette:

Rácz György

Programozó matematikus

Debrecen

2009

1. Köszönetnyilvánítás

Ezúton szeretném megköszönni a szakdolgozatom elkészítéséhez kapott anyagi és erkölcsi segítséget és támogatást a következő személyeknek és intézményeknek.

Legelőször is szeretném megköszönni az édesapámnak, keresztszüleimnek valamint apai nagyszüleimnek mindazon anyagi és erkölcsi támogatást, amivel hozzájárultak egyetemi tanulmányaimhoz! Tudom sokszor erejükön felüli mértékben támogattak ezekben az években.

Szakdolgozat vezetőmnek, Dr. Almási Béla Tanár Úrnak, aki értékes és hasznos megjegyzéseivel, észrevételeivel nagy mértékében a segítségemre volt a szakdolgozatom elkészítésében.

A budapesti GFTH Kft. alkalmazottainak, fejlesztői csapatának, akik nagyon pozitívan fogadtak, a munkába állásom kezdetén felmerülő kérdéseimre késséggel válaszoltak, segítettek a kezdeti nehézségek leküzdésében.

Végül, de nem utolsó sorban szeretném megköszönni a Debreceni Egyetem Természettudományi és Informatika Karának valamennyi oktatójának, dolgozójának azt a lelkiismeretes munkáját és erőfeszítését, amivel a hallgatók képzését, tanulását, a versenyképes tudás megszerzését támogatják.

2. Bevezetés

2.1. Szakdolgozatom témája

A Debreceni Egyetemen eltöltött évek során különböző programozási nyelvekkel – C, Assembler, Java – ismerkedtem meg és sajátítottam el alap szinten. A Java nyelvet – többek között az OO szemlélet és a platformfüggetlenség miatt – nagyon megkedveltem, és több Java programozással foglalkozó tárgyat is hallgattam. (Mobil programozás, Webalkalmazás készítés, Java esettanulmányok) Ezen tárgyak keretében ismerkedtem meg a Java Servlet–ekkel és a webalkalmazás készítésének alapjaival.

A budapesti GFTH Kft–nél 2008–ban kezdtem el dolgozni, mint Java fejlesztő. A cégnél egy már 1,5 éve folyó webalkalmazás fejlesztési projecthez csatlakoztam. A szakdolgozatomban azokat a Java eszközöket/technológiákat kívánom bemutatni, amelyekkel a munkám során kerültem kapcsolatba, és amelyek alapismerete elengedhetetlenül szükséges egy komplex webalkalmazás kifejlesztéséhez.

Azt várom, a szakdolgozat megírásától, hogy több és részletesebb ismeretekre teszek szert a Java alapú webes alkalmazásfejlesztéssel kapcsolatban, és ezek az ismeretek sokat segíthetnek a későbbiek során más hasonló technológiák elsajátításakor. Természetesen a szakdolgozat elkészítésével szeretném az egyes fejezetekkel kapcsolatban a saját ismereteimet is még inkább elmélyíteni és tovább bővíteni, valamint a téma kutatása során új ismeretekkel kiegészíteni.

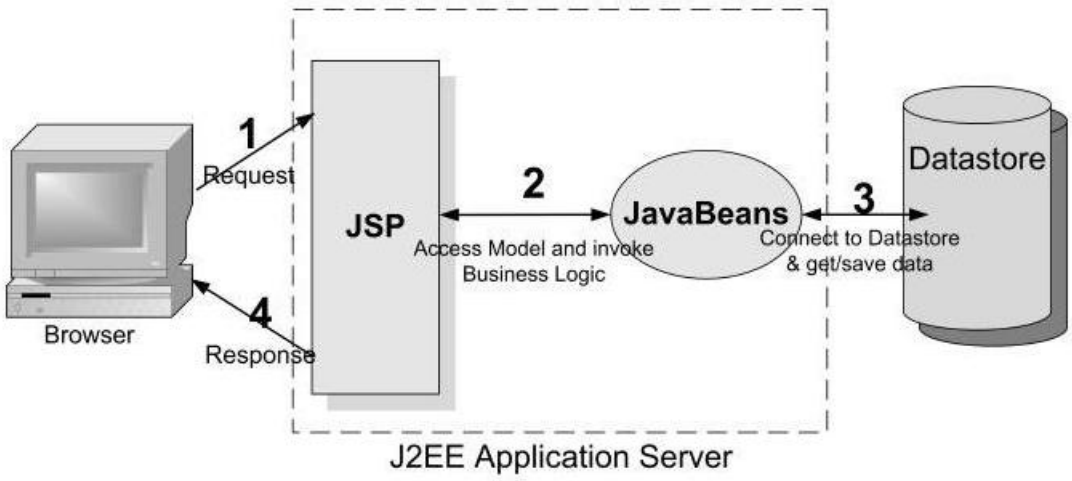
Azt remélem, hogy a szakdolgozatommal sikerül felkeltenem az olvasó érdeklődését a webalkalmazások készítése iránt. Ily módon dolgozatom akár kiindulópontot is jelenthet egy–egy olyan érdeklődő számára, aki a feldolgozott témával szeretne megismerkedni, esetleg a későbbiek folyamán a programozás ezen területén szeretne dolgozni.

3. Az MVC Design Pattern

Az MVC (Model View Controller) tervezési mintát 1979–ben[1] **Trygve Reenskaug** írta le, aki a Smalltalk–on dolgozott a Xerox kutatói labornál. Az eredeti megvalósítás részletesen az *Applications Programming in Smalltalk–80: How to use Model–View–Controller* című tanulmányban olvasható.[2] [4] (Wikipédia (magyar) 2009) [6] (Trygve/MVC 2009) [5] (How to use Model-View-Controller (MVC) 2009)

3.1. Modell 1 architektúra (MVC1)

A Modell 1 architektúra a legegyszerűbb tervezés minta, amely segítségével `JSP` alapú webalkalmazások készíthetőek. A Modell 1 szerint, ugyanis a böngésző közvetlenül hozzáfér a `JSP` oldalakhoz, és a felhasználótól érkező különböző kéréseket, maguk a `JSP` oldalak kezelik le. Tehát, ha van egy `HTML` oldalunk – amely például egy `JSP` oldalra hivatkozó linket tartalmaz – akkor, ha a felhasználó a linkre kattint, akkor ez a `JSP` oldal közvetlenül fog meghívódni. A `JSP` oldal a servlet container[3] által servletté fordul és végrehajtódik. A `JSP` oldal tartalmazhat beágyazott Java kódot (szkriptletet), hozzáférhet a `JavaBean`–hez és `JDBC` segítségével adatbázis műveleteket is végrehajthat. A megjelenítendő tartalom (`JavaBean`) és a megjelenítésért felelős rész (`JSP`) nem nagyon különül el egymástól.



A Modell 1 architektúra működési vázlata [10] (Shenoy 2004)

[1] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
[2] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html
[3] A Java Servlet és a JavaServer Pages specifikációkat támogató webszervereket szokás servlet container, servlet engine illetve web engine-nek is hívni.

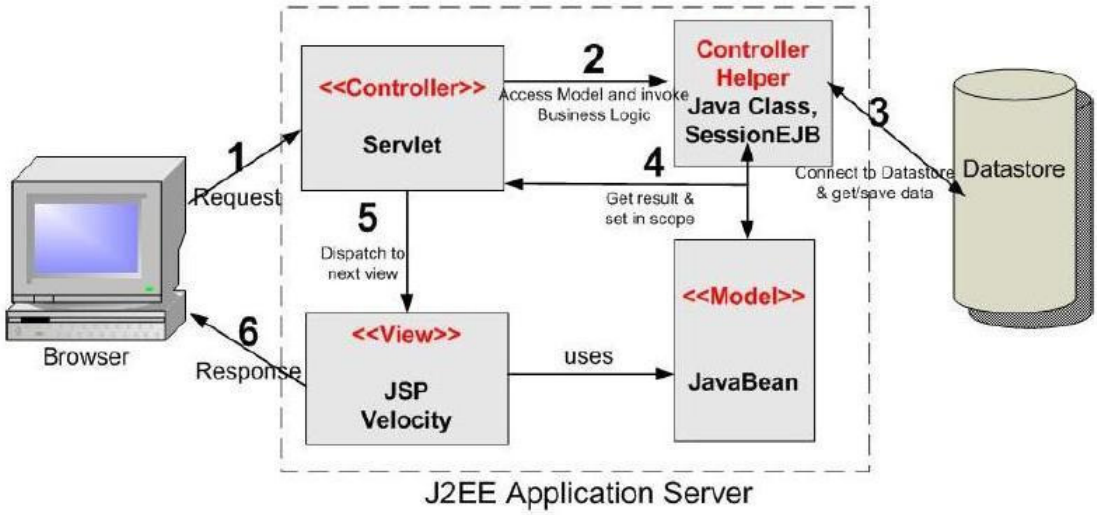
3.1.1. A Modell 1 architektúra problémái

Modell 1 architektúrában a Modell (`JavaBean`) és megjelenítés (`JSP`) nincs egymástól teljesen elválasztva. Ez a módszer elég jól használható kisebb alkalmazások esetében. Nagyobb alkalmazásoknál, azonban számos prezentációs logika van. A Modell 1–ben a prezentációs logika a `JSP` oldalban elhelyezett Java scriptletek segítségével valósul meg. Egy összetettebb oldal esetén ez nagyon bonyolult, nehezen olvasható kódot eredményez, amelynek a megértése – és az esetleges hibák kijavítása – még a tapasztalt Java fejlesztőknek is komoly nehézséget okozhat. Nagyobb alkalmazások fejlesztése esetén a `JSP` oldalakat nem is maguk a Java programozók készítik el, hanem ez már a dizájnerek feladata, mivel a `JSP` oldal elkészítéséhez nem szükséges programozási ismeret.

A Modell 1 architektúrában az alkalmazás vezérlése decentralizált, és ez sok esetben komoly problémákhoz vezethet.

3.2. Modell 2 architektúra – MVC2 (Model View Controller)

Összetettebb alkalmazások fejlesztésénél gyakori, hogy az alkalmazást több rétegre bontjuk: megjelenítés (felhasználói felület), alkalmazás logika és adatelérés részekre. Az MVC2 tehát egy olyan háromrétegű modell, amely különválasztja a **megjelenítést**, a **vezérlést** és az **adatbáziskezelést**. A továbbiakban a megjelenítést Nézetnek (View), a vezérlést Vezérlőnek (Controller) és az adatbázis kezelést Modellnek (Model) nevezzük. Az egyes rétegek feladatait a következő alfejezetek írják le. [1] (Ford 2004) [2] (Elte 2009)



A Modell 2 architektúra működési vázlata [10] (Shenoy 2004)

3.2.1. Modell (Model)

A Modell réteg felel az adatok kezeléséért, az ebbe rétegbe tartozó osztályok írják le az üzleti logikát, definiálják azokat az adatszerkezeteket, amelyeket használni fogunk az alkalmazásban, valamint meghatározzák azokat a szabályokat, amelyek alapján elérhetjük azokat. Ha valamilyen szerkezeti módosítást végzünk az adatbázisban, akkor jó esetben csak a Modell megfelelő osztályát kell megváltoztatnunk.

3.2.2. Nézet (View)

A Nézet réteg, mint a nevéből is következik a megjelenítéssel/prezentációval kapcsolatos osztályokat írja le, a megjelenítés rendszerint `JSP` oldalak segítségével történik. Feladata, hogy a Modell–t – vagy annak egy részét – a megfelelő alakban jelenítse meg a felhasználó számára. Különböző esetekhez/eszközökhöz (például számítógép vagy mobiltelefon) ugyanazon Modell esetében is más–más Nézet tartozhat. Minden támogatni kívánt eszközhöz elkészítjük a hozzá tartozó Nézetet, a megjelenítéshez használt adatok azonban minden Nézet esetében ugyanabból a Modell–ből származnak.

A konzisztens működéshez elengedhetetlenül szükséges a következő alapszabály betartása: **a Modell–ben lévő adatokat csak a Modell osztályainak példányain keresztül érhetjük el és módosíthatjuk!** [3] (Janka János 2009)

3.2.3. Vezérlő (Controller)

A Modell–ben lévő adatok és a megjelenítésért felelős Nézet közötti kapcsolatot a Vezérlő réteg valósítja meg. Egy felhasználótól érkező eseménykor (egérkattintás, billentyűleütés) a Vezérlő réteg továbbítja a kérést a megfelelő Modell osztály megfelelő metódusának, illetve azt követően a megjelenítésért felelős Nézetnek.

A Modell 1 és a Modell 2 között az alapvető különbség a felhasználói kérésekhez kapcsolódó eseménykezelésben van. A felhasználótól beérkező kérések kezelését nem egy `JSP` oldal, hanem egy eseménykezelő (Vezérlő) szervlet végzi. Egy MVC2 tervezési minta alapján elkészített webalkalmazás esetén a felhasználó által kiváltott esemény hatására a következő lépések hajtódnak végre [7] (Wikipédia (angol) 2009):

1. A felhasználó által kiváltott esemény (pld: egérrel történő kattintás, billentyűzet leütés) kezelését a Vezérlő kapja meg.
2. A Vezérlő a kérésben szereplő paraméterek alapján példányosítja a megfelelő `JavaBean`–t.
3. Ezt követően közvetlenül vagy különböző – az adatbázis elérését, és az azzal való kommunikációt segítő – segéd osztályok segítségével lekérdezi az adatbázisból a szükséges adatokat.
4. A Vezérlő az adatbázisból kiolvasott adatokkal feltölti a már létező `JavaBean`–t, vagy ha szükséges, akkor létrehoz egy újat, és ezt elhelyezi a valamelyik hatókörben (`scope`): `request`, `session` vagy `application`.
5. A Vezérlő ezt követően a következő – a kérésben szereplő `URL`–nek megfelelő – Nézet–hez továbbítja a kérést.
6. A Nézet megjelenítése a 4. pontban létrehozott `JavaBean`–ben lévő adatokat alapján történik.

Mivel nincs prezentációs logika a `JSP`–ben, ezért szkriptletek sincsenek, illetve nincs rájuk szükség. Az MVC2 ugyanakkor nem tiltja meg, hogy szkriptleteket helyezzünk el a `JSP` oldalakban, ezek használatát azonban lehetőség szerint kerüljük. Az MVC2 használata azt a célt szolgálja, hogy ne kelljen a `JSP` oldalakba szkriptleteket írni, túl sok szkriptlet használata általában valamilyen tervezési hiányosságra vezethető vissza.

3.2.4. Az MVC2 előnyei

Az MVC2 nagy előnye tehát az, hogy az üzleti logika elválik az interakciós logikától, az alkalmazás egyes részei jól elkülönülnek egymástól, ezzel is támogatva az újrafelhasználhatóságot. Az egyes Nézetek nincsenek kapcsolatban egymással, nincsenek direkt hivatkozások. Lehetővé teszi, hogy az igazi nagy projektek fejlesztése csapatmunkában történjen – hiszen a feladatok és a felelősségek is jól körülhatárolhatóak, az egymásra utaltság minimális, a feladatok jól koordinálhatóak így – mindenki a saját feladatára összpontosíthat.

3.2.5. Az MVC2 hátrányai

Az MVC2 hátránya, hogy az alkalmazás tervezésére több időt kell fordítani, – kezdve a megfelelő MVC2 framework kiválasztásával[4] – az egyes osztályokat jól meg kell tervezni, ez a befektetés azonban a későbbiek során mindenképpen megtérül, hiszen egy jól strukturált könnyen átlátható, és a későbbiekben is jól menedzselhető kódot fog eredményezni.

3.3. MVC alapú webalkalmazás frameworkok

Néhány a gyakorlatban használt MVC alapú webalkalmazás framework:

- Aranea
- Cocoon
- Grails
- Google Web Toolkit
- Oracle Application Framework
- Spring MVC Framework
- **Struts**
- Stripes
- Tapestry
- WebObjects
- WebWork
- Wicket
- JSF

Mint a felsorolásból is látszik számos framework létezik, ezek között vannak ingyenes (open source) és fizetős megoldások is. A következő fejezetben a felsorolt framework–ök közül az ingyenesen használható Struts framework használatáról lesz szó.

[4] http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

4. Struts

4.1. Bevezető

A Struts az Apache által fejlesztett nyílt forráskódú keretrendszer, amely megkönnyíti a Java alapú webalkalmazások készítését. A korábbi kizárólag JSP technológiát használó webalkalmazásokban gyakran keveredtek az adatbázis kezeléssel, megjelenítéssel és a vezérléssel kapcsolatos kódok, így ezek a rendszerek átláthatatlanok, nehezen karbantarthatók és ezért költségesek voltak.

A Struts teljes mértékben megvalósítja az MVC 2 architektúra modell–t, így az egyes funkciók jól elkülönülnek egymástól. A megjelenítésre (View) JSP oldalak és Tag Library–k szolgálnak, a Vezérlés (Controller) XML alapú konfigurációs fájl és a feldolgozó osztályok úgynevezett Action–ök segítségével valósul meg. A Struts keretrendszer a Modell résszel nem foglalkozik. Az MVC modell–t megvalósító rendszerek karbantartása, későbbi módosítása, további fejlesztése sokkal költségkímélőbb, mint egy nem MVC alapú rendszeré. A Struts nagyságrendileg 300 osztályból és interface–ből áll, amelyek 12 csomagba vannak szervezve. [15] (Struts Tutorial 2009) [13] (Struts Wiki 2009)

A Struts előnyei:

- Az alkalmazás vezérlését a Vezérlő (Controller) végzi, ezt nem kell programozni elég csak konfigurálni. A Vezérlő mindent megvalósít, ami a feladatkörébe tartozhat: akciók leképezése, navigáció, továbbítások
- Az Action–ök könnyen elérik a Modell adatait, de a folyamatot a Vezérlő (Controller) továbbítja.
- A Modell (Model) és a Nézet (View) független egymástól, a Nézet–et általában JSP segítségével kerül megvalósításra.
- Az irányítási gráfot egyetlen állomány a struts-config.xml tartalmazza, amely könnyen kezelhető.
- Struts Tag Library–ik hatékony eszközök, bár egy részük a Java Standard Tag Library (JSTL) miatt feleslegessé vált.
- Kiegészítő szolgáltatások: I18N, L10N, JavaBean kezelés, XML feldolgozás

A Struts hátrányai:

- Az XML fájl elég nagyra is megnőhet, ilyenkor fejlesztőeszköz nélkül már nehéz lehet átlátni.

- Régi nézőpontot képvisel, versenytársak: Tapestry (más komponens modell), Java Server Faces

4.2. Telepítés

A Struts legfrissebb verzióját – ez jelenleg a 2.0.12 – a http://struts.apache.org/ címről tudjuk letölteni. A letöltés után tömörítsük ki a `.zip` fájlt, ekkor automatikusan létrejön egy `struts-2.0.12` nevű könyvtár. Ebben a könyvtárban található a `lib` alkönyvtár, amely a `struts.jar`–t – amely az alapkönyvtár – és további `jar` fájlokat tartalmaz. A `lib` könyvtár tartalmát a webalkalmazásunk `WEB-INF/lib` könyvtárába kell bemásolni. [11] (Sincock 2003)

4.3. A Struts Framework

4.3.1. Modell (Model) komponensek

A Modell komponensek általában Java osztályok, nincs előre definiált Modell komponens, ezeket a Java osztályokat más projectekben is fel tudjuk használni. A Modell teljes mértékben az ügyfél igényeinek megfelelő üzleti logikát írja le.

4.3.2. Nézet (View) komponensek

A Nézet komponensek felelősek a felhasználóval való kapcsolattartásért, fogadják a felhasználótól érkező adatokat, és megjelenítik a kéréseknek megfelelő Modell adatokat. Az adatok megjelenítésére leggyakrabban `JSP` oldalak használunk, amelyeket szükség szerint `JavaScript`–el illetve `Custom Tag`–ekkel is kiegészíthetünk.

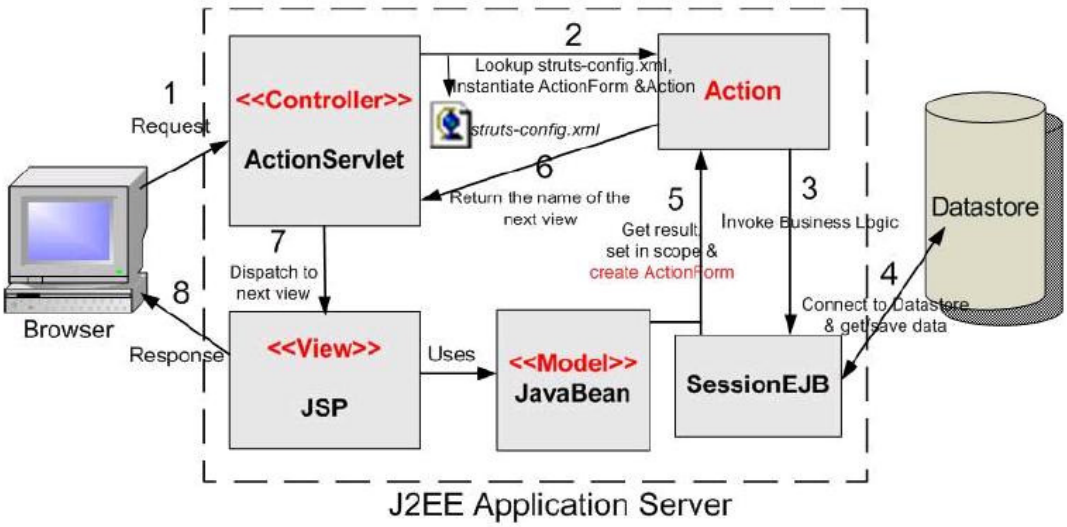
4.3.3. Vezérlő (Controller) komponensek

A felhasználótól érkező valamilyen kérés a Struts `ActionServlet`–nek kerül továbbításra, amely ezt a kérést az `URL` alapján a konfigurációs fájlban megadott `Action`–nek továbbítja kezelésre. Az `Action` egy vezérlő komponens, amely képes a Modell réteggel való kommunikációra.

4.4. A Struts Vezérlő (Controller) osztályai

A Struts vezérlő osztályai – `ActionServlet`, `RequestProcessor`, `ActionForm`, `Action`, `ActionMapping`, `ActionForward` – az

`org.apache.struts.action` csomagban vannak és a Struts konfigurációs fájljában (`struts-config.xml`) vannak leírva. [9] (Hightower 2004) [12] (Spielman 2003)



A Struts framework működési vázlata [10] (Shenoy 2004)

4.4.1. ActionServlet

A Struts Vezérlő (Controller) központi komponense a `javax.servlet.HttpServlet` osztályt kiterjesztő `ActionServlet` osztály, amelynek két feladata van:

1. Az alkalmazás indulásakor az `init()` metódus segítségével betölti a memóriába a Struts konfigurációs állományát.
2. A `doGet()` és a `doPost()` metódusokban elfogja a beérkező `HTTP` kéréseket, és megfelelő módon kezeli őket.

A Struts konfigurációs állományt – csupán konvenciós okokból – `struts-config.xml` néven a webalkalmazás `WEB-INF` alkönyvtárában szokás elhelyezni. A konfigurációs állomány neve tehát tetszőleges lehet, és a `WEB-INF` könyvtár tetszőleges alkönyvtárában elhelyezhető. A Struts konfigurációs állomány nevét a `web.xml`–ben kell megadni, az `ActionServlet` ezt az állományt dolgozza fel először, a Struts konfigurációs állomány feldolgozása csak ezt követően történik meg.

```
<servlet>
      <servlet-name>action</servlet-name>
      <servlet-class>org.apache.struts.action.ActionServlet</servlet-
      class>
      <init-param>
```

```
            <param-name>config</param-name>
            <param-value>/WEB-INF/config/myconfig.xml</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
</servlet>
```

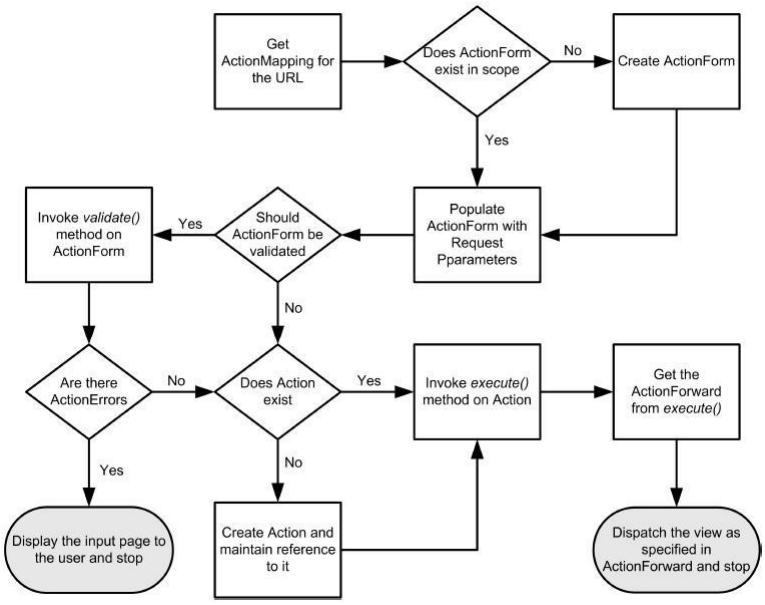
Az ActionServlet a Struts konfigurációs állomány nevét a web.xml <init-param> részéből olvassa ki. A webalkalmazás indulásakor az ActionServlet az init() metódusában feldolgozza a Struts konfigurációs állományt és a memóriában létrehozza a szükséges konfigurációs objektumokat (adatstruktúrákat).

Az első HTTP kérés beérkezésekor – a többi szervlethez hasonlóan – az ActionServlet–nek is meghívódik az init() metódusa, azonban a Struts konfigurációs állomány memóriába történő beolvasása nagyon időigényes feladat. Ha a konfigurációs állomány beolvasása az első kérés beérkezésekor történik meg, akkor az első felhasználó esetében nagy lesz a válaszidő. Egy lehetséges alternatív megoldás az lehet, ha a web.xml–ben megadjuk és 1–re állítjuk a <load-on-startup> tag–et, ekkor a servlet container a webalkalmazás elindításakor azonnal meghívja az init() metódust.

Az ActionServlet másik feladata, hogy az URL minta alapján megfelelő módon kezelje a beérkezett HTTP kéréseket. Az URL minta egy útvonal vagy egy kiterjesztés lehet, ezt a web.xml–ben lévő <servlet–mapping> rész írja le.

```
<servlet-mapping>
      <servlet-name>action</servlet-name>
      <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Amikor beérkezik egy HTTP kérés, akkor az ActionServlet továbbítja a kérést a RequestProcessor osztálynak és meghívásra az osztálynak a process() metódusa. Az alábbi folyamatábra részletesen megmutatja, hogy hogyan kezel egy HTTP kérést a RequestProccess osztály process() metódusa.



4.4.2. RequestProcessor és ActionMapping

A `RequestProcessor process()` metódusa beolvassa a `struts-config.xml`–ből az adott `URL`–hez tartozó `XML` blokkot. A Struts terminológiában az `ActionMapping` alatt ezt az `XML` blokkot értjük.

Az `ActionMapping` osztály az `org.apache.struts.action` csomag egy osztálya, és mint a neve is mutatja, ez teremti meg a kapcsolatot az `URL` és az `Action` között. Az `Action` definíciókat az `<action-mappings>` tag tartalmazza, amelyben minden egyes `URL`–hez definiálunk egy–egy `<action>` elemet, egy `<action>` három fő paramétere a következő: **`path`**, **`type`** és **`name`**.

A `path` az `Action`–höz tartozó relatív útvonalat adja meg, a `type` attribútum mondja meg a Struts–nak, hogy melyik `Action` osztályt kell példányosítania, a `name` attribútum pedig a form–hoz tartozó `JavaBean` neve, amit egy `<form-bean>` elemben kell definiálnunk. Az `XML` blokkban további paraméterek is szerepelhetnek, ezek együtt alkotják az adott útvonalhoz tartozó `ActionMapping` példány `JavaBean` tulajdonságait.

Minden egyes `HTTP` kérés útvonalának különböznie kell a többitől, mivel egy útvonalhoz csak egy `ActionMapping` hozzárendelés tartozhat. Ellenkező esetben a

korábbi ActionMapping összerendelés a legutolsóval kerül felülírásra. Egy rövid ActionMapping példa:

```
<struts-config
 <action-mappings>
  <action path=”/submitDetailForm”
  type=”mybank.example.CustomerAction” name=”CustomerForm”
  scope=”request” validate=”true”
  input=”CustomerDetailForm.jsp”>
   <forward
         name=”success”
         path=”ThankYou.jsp”
         redirect=”true”/>
   <forward
         name=”failure”
         path=”Failure.jsp” />
  </action>
  .
  .
 </action-mappings>
</struts-config>
```

4.4.3. ActionForm

Az ActionForm–ok olyan JavaBean osztályok, amelyek az org.apache.struts.action.ActionForm osztály leszármazottai, **semmilyen üzleti logikát nem tartalmaznak**, csak getXXX() és setXXX() – ezeken kívül esetleg még validate() és reset() – metódusaik vannak. Az ActionForm osztályok a Modell és a Nézet rétegek közötti adatcserét szolgálják. [9] (Hightower 2004)

Az ActionForm logikai neve az ActionMapping name attribútumában van megadva, ezt a RequestProcessor tölti fel értékekkel. Miután a RequestProcessor kiválasztotta a megfelelő ActionMapping–et, példányosítja a hozzá tartozó ActionForm–ot, ehhez azonban tudnia kell az ActionForm teljes nevét. Ezt a struts-config.xml–ben a következő módon lehet deklarálni:

```
<form-bean name=”CustomerForm” type=”mybank.example.CustomerForm”/>
```

A <form-bean> deklaráció a logikai névhez (CustomerForm) egy konkrét osztályt (mybank.example.CustomerForm) rendel hozzá. A RequestProcessor példányosítja az osztályt és kérés vagy session hatókörben

helyezi el. A hatókört a RequestProcessor az ActionMapping scope attribútuma alapján határozza meg. Ha az ActionBean–t session hatókörbe hozzuk létre, akkor fontos a reset() metódus implementálása, ugyanis a form különböző elemeit (például: checkbox, combobox, select, stb) minden használat előtt a megfelelő értékkel inicializálni kell.

A RequestProcessor a HTTP kérésben szereplő paraméterek nevei alapján – a megfelelő setXXX() metódusok segítségével – feltölti a form megegyező nevű paramétereit. Ezt követően a RequestProcessor ellenőrzi az ActionMapping validate attribútumát, és ha az értéke igaz, akkor a RequestProcessor meghívja a példányosított form osztály validate() metódusát.

A validate() metódussal lehetőség van az űrlapról beérkező adatok helyességének ellenőrzésére, a metódus szintaktikája:

```
public ActionErrors validate(ActionMapping mapping,HttpServletRequest
request)
```

Ez a metódus a JavaBean tulajdonságainak beállítását követően, de még az Action osztály execute() metódusának meghívása előtt fog lefutni. A metódus visszatérési értéke egy ActionErrors objektum, amely az űrlapon hibásan kitöltött mezőkkel kapcsolatos hibaüzenetek tartalmazza. A konkrét hibaüzentek a hibáknak megfelelő kulcsok alapján a MessageResource–ből kerülnek kiolvasásra. A vezérlés ezt követően visszakerül arra az oldalra, ahol a hiba történt, és <html:errors> segítségével lehetőség van a hibaüzentek megjelenítésre. Ha a form ellenőrzésekor nem történt hiba, akkor a metódus null–lal vagy egy üres ActionErrors objektummal tér vissza.

4.4.4. Action

Az Action–ök olyan Java osztályok, amelyek az org.apache.struts.action.Action absztrakt osztály leszármazottai, és implementálják annak execute() metódusát. A RequestProcessor az ActionMapping alapján példányosítja a megfelelő Action osztályt és meghívja annak execute() metódusát. Az execute() metódus szintaktikája a következő:

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws
Exception
```

A `RequestProcessor` minden `Action` osztályból csak egy példányt hoz létre az alkalmazásban. Az `Action` osztályoknak ezért szál biztosaknak kell lenniük, az attribútumaikat csak az `execute()` metóduson keresztül lehet megváltoztatni.

Az `Action` osztályok feladata, hogy az `execute()` metódusukban az űrlapról beérkező adatokat feldolgozzák (ezek általában adatbázis műveletek). A feldolgozással kapcsolatos kódrészleteket – bár kerülhetnek közvetlenül az `execute()` metódusba is – célszerű az üzleti logikát leíró osztályokban elhelyezni. Az `execute()` metódusba azért nem szerencsés üzleti logikát írni, mivel ez megakadályozná az üzleti logika későbbi újrafelhasználását! (Például ha valamilyen ok miatt a Struts framework–ről egy másikra térünk át, akkor az üzleti logikát ismételten implementálni kell.)

Az `execute()` metódus visszatérési érteke egy `ActionForward` objektum, amely azt adja meg, hogy a kérés feldolgozását követően a vezérlés hova kerüljön átadásra. (`JSP` oldal, Tile definició, Velocity sablon vagy esetleg egy másik `Action`).

4.4.5. ActionForward

Az `execute()` metódus egy `ActionForward` objektummal tér vissza, amely azt határozza meg, hogy melyik lesz a következő megjelenítendő `JSP` oldal a felhasználó számára. Az `ActionForward` a `JSP` oldal konkrét neve helyett csupán a logikai nevet tartalmazza. A logikai nevek és a konkrét fizikai állománynevek közötti kapcsolatot a `struts-config.xml` írja le. Két féle `ActionForward` van: **lokális** és **globális**. A globálisként deklarált `forward`–ok minden `ActionMapping`–ből hozzáférhetőek. [9] (Hightower 2004)

Az `ActionMapping`–nek három paramétere van: **`name`**, **`path`** és **`redirect`**. A `name` attribútum segítségével a `JSP` oldal logikai nevét lehet megadni, a hozzá tartozó fizikai állományt a `path` attribútum tartalmazza. Az `ActionMapping`

attribútumait csak és kizárólag az Action osztályok execute() metódusában lehet elérni. Az execute() visszatérési értékét – az ActionForward objektumot – az ActionMapping findForward() metódusa a következő módon adja meg, például:

```
ActionForward forward = mapping.findForward(“success”);
```

A findForward() metódus paraméterül a megjelenítésre kerülő oldal logikai nevét kapja meg, amelyet először az ActionMapping–ben, majd a globális forward deklarációban keres. Az execute() végrehajtása után a RequestProcessor már a megjelenítésre kerülő konkrét fizikai állomány nevét kapja meg.

4.4.6. ActionErrors és ActionError

Előfordulhat, hogy a felhasználó hiányosan vagy nem megfelelő adatokkal tölti ki a form–ot, ezért az adatok feldolgozása során különböző hibák jelentkezhetnek. (Például hiányzó értékek az adatbázis INSERT vagy UPDATE műveleteihez.)

A Struts ActionForm osztálya rendelkezik egy validate() metódussal, amely lehetővé teszi, hogy a form–ról beérkező adatok helyességét ellenőrizzük. A validate() metódus visszatérési értéke egy ActionErrors objektum példány, amely kulcs–érték párokat tartalmaz.
Példa az ActionErrors objektum létrehozására:

```
ActionErrors errors = super.validate(mapping, request);
```

Az egyes hibákhoz tartozó ActionError objektumokat az ActionErrors add() metódusának segítségével helyezhetjük el a Map–ben. A kulcs általában annak a mezőnek a neve, ahol a hiba bekövetkezett vagy általános hiba esetén GLOBAL_ERRORS. A kulcshoz tartozó érték egy ActionError objektum, amelynek a konstruktorában a hibához tartozó hibaüzenet kulcsát kell megadni. Két hibaüzenet hozzáadása az ActionErrors objektumhoz:

```
errors.add("firstName",new ActionError("error.firstName.null"));
errors.add(GLOBAL_ERROR,new ActionError("error.custform"));
```

A kulcshoz tartozó hibaüzenet – az adott területi és nyelvi beállításoknak megfelelő – `.properties` állományból kerül kiolvasásra. A Java a `java.util.ResourceBundle` osztály segítségével lehetőséget biztosít arra, hogy egy alkalmazás különböző területi és nyelvi beállításokat támogasson, erről a lehetőségről a 4.6 fejezetben részletesebben is lesz szó.

Abban az esetben, ha hiba következik be a `RequestProcessor` leállítja az adatok további feldolgozását, és az `ActionMapping` alapján dönt arról, hogy melyik legyen a következő megjelenítendő `JSP` oldal. Az `ActionMapping` input attribútuma tartalmazza annak az állománynak a fizikai nevét, amelyet hiba esetén meg kell jeleníteni. Általában ez az oldal megegyezik azzal az oldallal, ahol a felhasználó kitöltötte a form–ot, mivel azt az adatok javítását, pontosítását követően az adatokat így újra el tudja küldeni.

4.4.7. struts–config.xml

Az előző fejezetek alapján kijelenthetjük, hogy a `struts-config.xml` konfigurációs állományt tekinthetjük a Struts technológia lelkének, hiszen ebben a fájlban deklaráljuk, hogy egy–egy űrlap elküldésekor vagy egy linkre történő kattintáskor milyen `Action` eseménykezelő hajtódjon végre, és hogy ehhez az `Action`–höz milyen `ActionForm` tartozik. A Struts framework konfigurációs állománya (például a Struts 1.1 esetében a `struts-config_1_1.dtd`) tartalmazza az adott Struts verzióban használható összes lehetséges elemet, ezek tulajdonságait és leírását. A `struts-config.xml`–ben használt öt legfontosabb attribútum a következő:

1. Form bean deklarációs rész `<form-beans/>`
2. Globális forward deklarációs rész `<global-forward/>`
3. ActionMapping deklarációs rész `<action-mappings/>`
4. Vezérlő (Controller) deklarációs rész `<controller/>`
5. Alkalmazás által használt erőforrás állomány (Application Resources) deklarációk `<message-resources/>`

Az ActionForm–okat a <form-beans/> tagek között lehet megadni, minden form–hoz egy egyedi logikai név tartozik, az ActionForm–al a 4.4.3–as fejezet foglalkozik részletesen.

A <global-forward/> tagek között megadott form deklarációk minden ActionMapping–ből elérhetőek, részletesebb leírás a 4.4.5–ös fejezetben található.

Az <action-mappings/> tag segítségével lehet megadni, hogy az egyes útvonalakhoz milyen Action osztályok illetve ActionForm–ok tartoznak, valamint itt adhatjuk meg a lokális forward deklarációkat. Az ActionMapping használatával a 4.4.2–es fejezet foglalkozik.

A <controller/> megadása nem kötelező, mivel van alapértelmezett Vezérlő (Controller) osztály (org.apache.struts.action.RequestProcessor). Ezen tag megadására például JSP sablon oldalak használata esetén lehet szükség, ekkor ugyanis a RequestProcessor helyett a TilesRequestProcessor–t kell használni.

A <message-resources/> tag segítségével adhatjuk meg az alkalmazáshoz tartozó egy vagy több ApplicationResources.properties állományt. Ez az állomány kulcs–érték párokat tartalmaz, az állomány használatával kapcsolatos részletesebb leírás 4.6–os fejezetben található. Egy teljes struts-config.xml állomány:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
      <form-beans>
            <form-bean name="CustomerForm"
            type="mybank.example.CustomerForm"/>
            <form-bean name="LogonForm"
            type="mybank.example.LogonForm"/>
      </form-beans>
      <global-forwards>
            <forward name="logon" path="/logon.jsp"/>
            <forward name="logoff" path="/logoff.do"/>
      </global-forwards>
      <action-mappings>
            <action path="/submitDetailForm"
                  type="mybank.example.CustomerAction"
                  name="CustomerForm"
                  scope="request"
                  validate="true"
                  input="/CustomerDetailForm.jsp">
```

```
                    <forward name="success"
                          path="/ThankYou.jsp"
                          redirect=”true” />
                    <forward name="failure"
                          path="/Failure.jsp" />
              </action>
              <action path=”/logoff” parameter=”/logoff.jsp”
              type=”org.apache.struts.action.ForwardAction” />
        </action-mappings>
        <controller
        processorClass="org.apache.struts.action.RequestProcessor" />
        <message-resources parameter="mybank.ApplicationResources"/>
</struts-config>
```

4.5. Struts 1.1 Tag–ek

4.5.1. HTML Tag–ek

A Struts rendelkezik egy HTML tag library–vel, amely megkönnyíti a HTML form–ok létrehozását, ezek a tag–ek mindig egy `<html:form>` elem belsejében szerepelnek. Ahhoz, hogy a HTML tag library–t használni tudjuk a `JSP` oldalunkban, abban szerepelnie kell a következő sornak [14] (RoseIndia 2009):

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
```

A Struts HTML tag library–jének leggyakrabban használt tag–jei a következők:

Struts HTML Tags	Leírás
`<html:message key="thekey"/>`	A megadott kulcshoz tartozó üzenetet jeleníti meg.
`<html:password property="prop" size="10"/>`	Egy jelszó mezőt hoz létre, a beírt jelszó a `Java Bean` azon változójában kerül eltárolásra, amely a `property` paraméterben került megadásra.
`<html:text property="text1" size="5"/>`	A `size` paraméterben megadott karakter hosszúságú szövegbeviteli mezőt hoz létre. A beírt sztring a `Java Bean` azon változójában kerül eltárolásra, amely a `property` paraméterben került megadásra.

`<html:submit>`
`Submit`
`</html:submit>`
	Egy SUBMIT gombot hoz létre, a gomb szövegét a nyitó és a záró tag–ek között lehet megadni.

<html:reset> Reset </html:reset>	Egy RESET gombot hoz létre, a gomb szövegét a nyitó és a záró tag–ek között lehet megadni.
<html:errors/>	Az oldalon lévő hibák megjelenítésére szolgál.
<html:file property="fileSelectionBox"/>	Állományok feltöltését teszi lehetővé a form–on, a property–nek egy org.apache.struts.upload.FormFile típust kell megadni.
<html:checkbox property="myCheckBox"/>	Egy checkbox–ot a hoz létre a form–on.
<html:hidden property="hiddenfield"/>	Egy rejtett – nem megjelenő – html elemet hoz létre a form–on.
<html:radio value="abc" property="myCheckBox"/>	Egy radiobox–ot hoz létre a form–on.
<html:select multiple="true" property="selectBox">	Egy több lehetőség kiválasztására alkalmas lista dobozt hoz létre a form–on. A property egy olyan tömb, amely a kiválasztható adatokat tartalmazza, ezeket az elemeket a <html:options> használatával adhatjuk meg.
<html:textarea property="myTextArea" value="Hello Struts"/>	Egy többsoros szövegbeviteli mezőt hoz létre a form–on.
<html:form action="/Address" method="post">	Egy HTML form elemet hoz létre, amely az adatokat post metódussal küldi át a szervernek.
<html:base/>	Egy <BASE> tag–et hoz létre, a későbbi linkek URL–je az itt megadotthoz viszonyított relatív címek.
<html:html/>	Egy <HTML> taget hoz létre.

4.5.2. Bean Tag–ek

A Struts Bean Tag–jei	
<bean:cookie>	Ez a tag beolvassa a megadott cookie által tartalmazott adatot/adatokat és – beolvasott adatok számától függően – elhelyezi az oldal hatókörében egy Cookie vagy

	`Cookie[]` típusú objektumban.
`<bean:define>`	Ez a tag egy új attribútumot és ennek megfelelően egy új script változót hoz létre a hatókörben az `id` paraméterben megadott névvel.
`<bean:header>`	Ez a tag kiolvassa a kérésből (`request`) a fejléc elmet (ez egy vagy több érték is lehet) és azt egy `String` vagy `String[]` objektumként elhelyezi azt oldal hatókörben (`scope`). Ha a megadott nevű paraméter nem található és a tag–nek nincs megadva alapértelmezett érték, akkor kivétel váltódik ki.
`<bean:include>`	Ez a tag hasonlóan működik, mint a standard `<jsp:include>` tag, azzal a különbséggel, hogy a válasz adatok az oldal hatókörben vannak tárolva.
`<bean:message>`	A nyelvi és területi beállításoknak megfelelő üzenetet jeleníti meg a kimeneten. Az üzenet szövegében {} jelek között paraméter is megadható (maximum 5), amelyek a kiírás előtt behelyettesítésre kerülnek.
`<bean:page>`	Ez a tag beolvassa az oldal egy meghatározott elemét és a beolvasott értéket az oldal hatókörben egy script változóban tárolja el.
`<bean:parameter>`	Ez a tag kiolvassa a kérésből (`request`) a megadott paraméter értékét és azt egy `String` vagy `String[]` objektumként elhelyezi azt oldal hatókörben (`scope`). Ha a megadott nevű paraméter nem található és a tag–nek nincs megadva alapértelmezett érték, akkor kivétel váltódik ki.
`<bean:resource>`	Ez a tag beolvassa a webalkalmazás megadott erőforrás fájlját és `String`–ként vagy `InputStream`–ként elérhetővé teszi.
`<bean:size>`	A tag hivatkozik egy tömbre, egy `Collection`–re vagy egy Map–re és létrehoz egy Integer típusú bean–t amely értékül a hivatkozott gyűjtemény (`Collection`) elemszámát kapja.
`<bean:struts>`	Ez a tag beolvassa a megadott Struts konfigurációs objektumot és elhelyezi az aktuális oldal névterében.
`<bean:write>`	A megadott `Bean` mezőjének értékét írja ki az aktuális `JspWriter` objektumba.

4.5.3. Logic Tag–ek

A Struts Logic Tag–ek segítségével kiválthatjuk a JSP oldalakban lévő scriptlet–eket. [14] (RoseIndia 2009)

A Struts Logic Tag–jei	
<logic:iterate>	A megadott Collection, Map vagy tömb minden elemére végrehajtódik egyszer a tag törzse. A megadott adatszerkezetnek implementálnia kell az Iterator interface–t.
<logic:empty>	A tag törzse csak akkor hajtódik végre, ha a form vizsgált mezőjének értéke nem létezik (null) vagy üres String.
<logic:notEmpty>	A tag törzse csak akkor hajtódik végre, ha a form vizsgált mezőjének értéke létezik, tehát nem null és nem üres String.
<logic:equal>	A tag törzse csak akkor hajtódik végre, ha a form vizsgált mezőjének értéke megegyezik az előre megadott értékkel.
<logic:notEqual>	A tag törzse csak akkor hajtódik végre, ha a form vizsgált mezőjének értéke nem egyezik meg az előre megadott értékkel.
<logic:greaterEqual>	Ez a tag a form vizsgált mezőjének értéket egy konstans értékkel hasonlítja össze. A tag törzse csak akkor hajtódik végre, ha a változó értéke nagyobb vagy egyenlő a konstans értékkel.
<logic:greaterThan>	Ez a tag a form vizsgált mezőjének értéket egy konstans értékkel hasonlítja össze. A tag törzse csak akkor hajtódik végre, ha a változó értéke nagyobb, mint a konstans érték.
<logic:lessEqual>	Ez a tag a form vizsgált mezőjének értéket egy konstans értékkel hasonlítja össze. A tag törzse csak akkor hajtódik végre, ha a változó értéke kisebb vagy egyenlő a konstans értékkel.
<logic:lessThan>	Ez a tag a form vizsgált mezőjének értéket egy konstans értékkel hasonlítja össze. A tag törzse csak akkor hajtódik végre, ha a változó értéke kisebb, mint a konstans érték.

<logic:match>	A tag törzse csak akkor hajtódik végre, ha a form vizsgált mezője tartalmazza a megadott konstans String–et.
<logic:notMatch>	A tag törzse csak akkor hajtódik végre, ha a form vizsgált mezője nem tartalmazza a megadott konstans String–et.
<logic:present>	A tag törzse csak akkor hajtódik végre, ha a kérésben szereplő mező neve megegyezik a vizsgált form mezőnevével.
<logic:notPresent>	A tag törzse csak akkor hajtódik végre, ha a kérésben szereplő mező neve nem egyezik meg a vizsgált form mezőnevével.
<logic:redirect>	Ez a tag a HTTP kérés átirányítására szolgál.

Példa: Egy vásárlóra vonatkozó feltétel (John Doe 28 éves) scriptlet segítségével:

```
<%  if (customer.firstName == "John" && customer.lastName == "Doe" &&
customer.age == 28) {
                  <%-- do something --%>
    }
%>
```

és ugyanez a feltétel Strutc logic tag–ek használatával:

```
<logic:equal name="customer" property="firstName" value="John">
      <logic:equal name="customer" property="lastName" value="Doe">
            <logic:equal name="customer" property="age" value="28">
                  //do something….
            </logic:equal>
      </logic:equal>
</logic:equal>
```

A Struts Logic Tag–ekkel csak ÉS (AND) műveleteket tudunk végrehajtani, nincs lehetőség VAGY (OR) műveletekre. Ha VAGY (OR) műveletre van szükségünk, akkor a JSTL Expression Language (EL) –t kell igénybe vennünk.

4.5.4. Template Tag–ek

Ez a tag könyvtár három elemet tartalmaz: **put**, **get** és **insert** elemeket. A put tag elhelyezi azokat a tartalmakat a kérés hatókörében, amelyeket a különböző

JSP oldalak (sablonok) a get metódussal kérhetnek le. Egy sablont az incude tag–el lehet beilleszteni egy oldalba.

A Struts 1.1 verziójától a template tag használata nem javasolt (deprecated), a Struts 1.2 verziójában pedig már nem is szerepel, helyette a tiles tag használható.

A Struts Template Tag–jei	
<template:insert>	Sablon beszúrására szolgál, amely egy paraméterezhető tartalommal rendelkező JSP oldal. Az oldal tartalmát a put tag tartalmazza.
<template:put>	A sablon tartalmát elhelyezi a kérés hatókörében.
<template:get>	A korábban a put tag–el elhelyezet tartalmat veszi ki a kérés hatóköréből.

4.6. Internationalization(I18N)

A legtöbb alkalmazás esetében – és ez webalkalmazásokra különösen igaz – fontos alapkövetelmény, hogy a program fel legyen készülve több nyelv támogatására is, képes legyen a dátum, idő, pénznem, stb. adatok nyelvfüggő megjelenítésére. Az alkalmazásokat, már a tervezési fázisban – az Internationalization (I18N) segítségével – fel kell készíteni arra, hogy képesek legyenek több nyelv támogatására is, így nem kell az alkalmazás kódját módosítani és újrafordítani egy újabb nyelv támogatásához. [1] (Ford 2004)

A Struts keretrendszer az alkalmazásban használt szövegeket, üzeneteket alapértelmezés szerint a \WEB-INF\classes\resources könyvtárban található MyApplication.properties[5] állományból olvassa ki. Minden – a webalkalmazás által támogatott – nyelvhez tartozik egy MyApplication_xx.properties nevű állomány, ahol az xx karakterek helyén, az adott nyelv ISO kódja szerepel. A különböző nyelvekhez így elkészített állományokat a struts-config.xml–ben fel kell sorolni.

Példa: Egy programhoz tartozó négy nyelv (francia, spanyol, angol és német) erőforrás fájljainak a megadása:

[5] Ahol a MyApplication természetesen a webalkalmazásunk neve.

```
<!-- Message Resources Definitions -->
<message-resources parameter="com.mycompany.mypackage.MyApplicationResources_fr"/>
<message-resources parameter="com.mycompany.mypackage.MyApplicationResources_es"/>
<message-resources parameter="com.mycompany.mypackage.MyApplicationResources_en"/>
<message-resources parameter="com.mycompany.mypackage.MyApplicationResources_de"/>
```

A Struts a lokalizálást a Java standard lokalizáló osztályainak segítségével valósítja meg [8] (Cavaness 2004):

- `java.util.Locale`: Minden `Locale` objektum egy országot és nyelvet azonosít, valamint az ehhez kapcsolódó szám, dátum, stb. formázásokat.
- `java.util.ResourceBundle`: Ez az osztály támogatja a különböző nyelvű erőforrásfájlokat.
- `java.util.PropertyResourceBoundle`: A `ResourceBundle` egy standard implementációja, amely az üzeneteket java property fájlokban kulcs–érték (`name=value`) formában tárolja.

  Példa: Egy program üzenetei az angol és a magyar nyelvű .properties fájlokban:

language_en_EN.properties	**language_hu_HU.properties**
`message.confirm.proceed.text=You have unsaved data. Are you sure you want to proceed?`	`message.confirm.proceed.text=Önnek nem mentett adatai vannak. Biztos hogy folytatja?`
`message.confirm.switch.screen.text=You have unsaved data. Are you sure you want to switch screens?`	`message.confirm.switch.screen.text=Önnek nem mentett adatai vannak. Biztos hogy elhagyja ezt a képernyőt?`
`message.confirm.offline.mode.not.avaliable.text=Offline mode not available for this screen.`	`message.confirm.offline.mode.not.avaliable.text=Kapcsolat nélküli üzemmódban ez a képernyő nem elérhető.`

- `java.text.MessageFormat`: Lehetőség van arra, hogy a szöveges üzenetekben paramétereket helyezzünk el, és ezek értékeit futási időben adjuk meg. A paramétereket `{index}` alakban kell megadni, ahol az index 0–val kezdődik és maximális értéke 5 lehet.
- `org.apache.struts.util.MessageResources`: Egy konkrét üzenet egy adott nyelven történő megjelenítésére szolgál.

Ha a webalkalmazásunkat már eleve úgy készítettük el, hogy képes legyen több nyelvet is támogatni[6], akkor a beállítások helyes elvégzése után a programunk mindig az aktuális nyelvi beállításoknak megfelelően fog futni. Például: ha a böngészőprogramban a

[6] A menüpontok, hibaüzenetek, szövege nincs „beégetve” a programba, hanem ezek az adatok a megfelelő – éppen aktuális – állományból kerülnek beolvasásra.

nyelvet – egy az alkalmazás által támogatott nyelvre – változtatjuk meg, akkor ezt követően a webalkalmazásba való ismételt belépéskor már az új nyelvi beállításoknak megfelelő nyelven fog futni a programunk

5. Log4J

5.1. Bevezető

A naplózásra szinte minden program írásakor szükség van, kezdve a legegyszerűbbtől egészen a komplex összetett alkalmazásokig.

A Log4J az Apache Software Foundation Apache Logging Services Project keretében létrejött ingyenesen használható naplózó rendszer Java–hoz. A Log4J a `JDK 1.4` előtt időből származik – pontosan a Java 1.1 óta létezik – amikor még nem volt naplózás a Java–ban. Segítségével egyszerűsíthetjük, egységesíthetjük a programjaink visszajelzéseit, amely hibakereséskor nagymértékben megkönnyítheti a munkánkat. [17] (pcjuzer 2009)

A Log4J népszerűségét jól jelzi, hogy szinte minden framework és alkalmazás szerver használja, illetve támogatja a használatát. (Pld. JBoss, Spring, Hibernate, Struts) Előnye, hogy rugalmas számos `appender` (csatorna lásd később) van hozzá, sokféleképpen lehet konfigurálni (pld: xml, property file, stb), illetve ha véletlenül a naplózást hibásan konfiguráltuk, akkor a programunk attól még futni fog, nem száll el. A naplózás ki– és bekapcsolását konfigurációs fájl segítségével tudjuk állítani, így a forráskódban semmilyen változtatásra nincs szükség.

A Log4J fejlesztésekor különösen nagy hangsúlyt fektettek arra is, hogy – mivel a Java alapvetően nem rendelkezik előfordítóval és a log üzenetek pedig a forráskódban szerepelnek – ne legyen észrevehető sebességkülönbség egy naplózott és egy naplózás nélkül futó kód között. A napló üzenetek nagyon hasznosak, de figyeljünk arra, hogy ezek – ha túl sűrűn szerepelnek a kódunkban – ronthatják a forráskód olvashatóságát, ezért fontos hogy megtaláljuk a helyes arányokat és csak a működés nyomon követéséhez elengedhetetlenül szükséges adatokat írassuk ki.

5.2. Telepítés

A Log4J legfrissebb verzióját – ez jelenleg az 1.2.15 – a http://logging.apache.org/log4j/1.2/download.html címről tudjuk letölteni A `.zip` állomány letöltése után tömörítsük ki, ekkor automatikusan létrejön egy `apache-log4j-1.2.15` könyvtár, amely tartalmazza a `log4j-1.2.15.jar-t`. A Log4J

használatához ennek a .jar fájlnak az elérés útvonalát kell megadni a CLASSPATH–ban. Fontos megjegyezni, hogy a CLASSPATH–ban csak egy Log4J elérési útvonal szerepelhet, mivel ezek statikusan kapcsolódnak, ezért bármelyik osztályban is hozzuk létre a loggert, az az összes többivel kapcsolatban lesz.

5.3. Konfigurálás

5.3.1. Log4J hozzáadása egy project–hez

A Log4J először megnézi, hogy a project–ünk gyökér könyvtárában (src) létezik–e a log4j.xml vagy a log4j.properties nevű konfigurációs állomány. A konfigurációs állomány, a naplózással kapcsolatos különböző beállításokat tartalmazza.

5.3.1.1.log4j.xml

A log4j.xml használatához szükség van a log4j.dtd fájlra, amelyet szintén az src könyvtárban kell elhelyezni, valamint a dom4j.jar fájlra, amelyet a régebbi Java verziók még nem tartalmaztak.
Példa egy lehetséges log4j.xml állományra:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" >
<log4j:configuration>
<appender name="stdout" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.SimpleLayout"></layout>
</appender>
<root>
    <priority value="debug"></priority>
    <appender-ref ref="stdout"/>
</root>
</log4j:configuration>
```

5.3.1.2.log4j.properties

A log4j.properties konfigurációs fájl kevésbé informatív, és az log4j.xml–el szemben nem támogatja Szűrőket (Filters), egyéni hibakezeléseket (Custom ErrorHandlers) és a speciális Appender–eket (AsyncAppender). A .properties fájlból nagyon hiányzik ez a tartományszűrő funkció, amely lehetővé tenné, hogy például csak az INFO és WARN közti események kerüljenek naplózásra. A Szűrők segítségével lehetőség van

például arra, hogy a különböző szinteknek megfelelő napló üzenetek különböző állományokba kerüljenek kiírásra.

Példa egy lehetséges `log4j.properties` állományra:

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p
%c{1}:%L - %m%n
log4j.rootLogger=debug, stdout
```

5.3.2. Log4J inicializálása Tomcat–ben

Alapértelmezés szerint a Tomcat a `java.util.logging` csomagot használja naplózásra, de a megfelelő beállítások után lehetőség van más naplózási eszközök használatára is, mint például a Log4J.

5.3.2.1.Webalkalmazásonkénti Log4J beállítások

A `log4j.xml` vagy a `log4j.properties` állományt a webalkalmazásunk `WEB-INF/classes` könyvtárában kell elhelyezni, ekkor ez a konfigurációs állomány csak erre a webalkalmazásra lesz érvényes. A Log4J ebben a könyvtárban fogja keresni a konfigurációs fájlt és az abban szereplő beállítások alapján fogja inicializálni magát. Mind a két féle konfigurációs fájl használatához (`.xml` és `.properties`) be kell állítani a `CATALINA_OPTS` környezeti változót (Tomcat 3.X verziókban a `TOMCAT_OPTS` környezeti változót).

```
set CATALINA_OPTS="-Dlog4j.configuration=config.properties"
```

A parancs hatására a Log4J a `config.properties`–t használja alapértelmezett konfigurációs fájlnak, a fájl feldolgozását a `ProperyConfigrator` végzi.

```
set CATALINA_OPTS="-Dlog4j.debug -Dlog4j.configuration=config.xml"
```

A parancs hatására a Log4J a `config.xml`–t használja alapértelmezett konfigurációs fájlnak, a fájl feldolgozását a `DOMConfigrator` végzi.

5.3.2.2. Általános, minden webalkalmazásra vonatkozó beállítások

Lehetőség van arra is, hogy ne külön–külön webalkalmazásonként adjuk meg a konfigurációs állományokat, hanem az összes Tomcat–en futó webalkalmazásunk egy közös konfigurációs állományt használjon.

```
set CATALINA_OPTS=-Dlog4j.configuration=file:/c:/config.lcf
```

A parancs hatására a Log4J a `c:\config.xml`–t használja alapértelmezett konfigurációs fájlnak, a konfigurációs állományt teljes elérési útvonallal kell megadni.

Ez a megoldás azonban kerülendő, mivel szinkronizálás nélkül az egyes webalkalmazásokhoz tartozó Log4J példányok összeakadhatnak. Ha több `appender`–t használunk egyszerre, és ezek egy közös fájlba írnak, akkor ez nem biztonságos működéshez vezethet, mivel nincs szinkronizáció az `appender`–ek között még akkor sem, ha ezek ugyabban a `VM`–ben futnak. Ilyen eset lehet például, ha két webalkalmazás szeretne egy `FileAppender`–en keresztül ugyanabba a fájlba írni, az `appender`–ek nem tudnak osztozni, ugyanazon az rendszererőforráson. [16] (Gülcü 2002)

5.3.3. Log4J inicializálása Servletel

A webalkalmazásunkban elég a Log4J–t egyszer konfigurálni, erre legegyszerűbb a `BasicConfigurator.configure()` metódusát használni. Nagyon sokan ugyanakkor elkövetik azt a hibát, hogy ezt újra minden osztályhoz beállítják.

Egy alternatív megoldás lehet, az hogy a Log4J inicializálását egy szervletre bízzuk, amely nem csinál mást, mint az `init()` metódusában beolvassa a megfelelő konfigurációs fájlt. Ezt a szervletet a webalkalmazást leíró `web.xml` állományban kell megadni, ahol a `<load-on-startup>` paraméterrel állíthatjuk be, hogy ez az inicializáló szervlet csak egyszer induláskor fusson le. [19] (Viczián István 2009)

A Log4J konfigurálását végző szervlet:

```
import org.apache.log4j.PropertyConfigurator;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;
import java.io.IOException;
public class Log4jInitServlet extends HttpServlet {
      public void init() {
            String prefix = getServletContext().getRealPath("/");
            String file = getInitParameter("log4j-init-file");
            if(file != null) {
                  PropertyConfigurator.configure(prefix+file);
            }
      }
      public void doGet(HttpServletRequest req, HttpServletResponse
res) {}

}
```

Egy webalkalmazás `web.xml`–jében a `Log4JInitServlet` beállítása:

```
<servlet>
      <servlet-name>log4j-init</servlet-name>
      <servlet-class>geo.Log4jInitServlet</servlet-class>
      <init-param>
            <param-name>log4j-init-file</param-name>
            <param-value>WEB-INF/classes/log4j.properties</param-
            value>
      </init-param>
      <load-on-startup>1</load-on-startup>
</servlet>
```

5.3.4. Naplózás az osztályokban

Minden olyan osztályban, ahol naplózni szeretnénk az eseményeket, szükségünk van egy `Logger` osztály példányra, amelyet a `Logger.getLogger()` metódus segítségével kaphatunk meg. A `Logger.getLogger()` metódusnak paraméterül a naplózásra kerülő osztály nevét kell megadni, ezt a `Logger` objektumot általában `private static final`–ként deklaráljuk. A `Logger.getLogger()` metódusnak több változata is van, az alábbi három metódus egyenértékű [16] (Gülcü 2002):

```
Logger.getLogger("com.wombat.X");
Logger.getLogger(X.class.getName());
Logger.getLogger(X.class);
```

Példa a `Logger` objektum elkérésére a `RadixUtil.java` osztályban:

```
private static final Logger log = Logger.getLogger(RadixUtil.class);
```

Ezt követően a `RadixUtil` osztály eseményeinek naplózására a log különböző kiíró metódusait – `debug()`, `info()`, `warn()`, `error()`, `fatal()` – használhatjuk.

5.4. A Log4J komponensei

5.4.1. Loggerek

5.4.1.1.Szintek

A naplózás a `Logger` osztályban található kiíró metódusok segítéségével történik, ezek: `debug()`, `info()`, `warn()`, `error()`, `fatal()` és a `log()`. Mindegyik metódushoz – kivéve a `log()` generikus metódust – egy beépített szint tartozik. A szintek nevei az `org.apache.log4j.Level` osztályban vannak deklarálva és ezek a neveik megegyeznek a kiíró metódusok neveivel: `DEBUG`, `INFO`, `WARN`, `ERROR` és `FATAL`. A felsorolás sorrendje egyben a szintek sorrendjét is mutatja. A `DEBUG` az összes szintet tartalmazza, míg a `FATAL` nem tartalmaz egyetlen másik szintet sem. A `root logger`–nek mindig meg kell adni egy szintet, míg a többi `logger` örökölheti azt a szülőjétől.

Egy kiíró metódusnak akkor lesz csak hatása, ha a `logger`–nek megadott szint ugyanaz, vagy magában foglalja a kiíró metódushoz tartozó szintet. Azaz, ha `debug()` metódust használjuk, akkor az csak akkor jelenik meg a kimeneten, ha a `logger` szintje is `DEBUG`. Egy `warn()`–nal történő kiírás csak akkor jelenik meg, ha a `logger` szintje `WARN`, `ERROR` vagy `FATAL`. A `fatal()` metódussal történő kiírás mindig megjelenik.

Példa a különböző naplózási szintekre:

```
//naplózás szintjének beállítása
logger.setLevel((Level) Level.WARN);

//különböző szintű naplóüzenetek kiírása
logger.debug("Here is some DEBUG");
logger.info("Here is some INFO");
logger.warn("Here is some WARN");
logger.error("Here is some ERROR");
logger.fatal("Here is some FATAL");
```

Mivel a setLevel()-ben a WARN szintet adtuk meg, ezért csak a WARN illetve a magasabb szintű naplóüzenetek kerülnek naplózásra. A program futásának eredménye:

```
0 [main] WARN mathutil.htmlandwrite  - Here is some WARN
0 [main] ERROR mathutil.htmlandwrite  - Here is some ERROR
0 [main] FATAL mathutil.htmlandwrite  - Here is some FATAL
```

Az egyes naplózási szintek jelentése:

FATAL	Olyan súlyos hibák, amelyek a program azonnali befejeződéséhez vezetnek, ezek azonnal megjelennek a konzol kimeneten.
ERROR	Egyéb futási hibák vagy előre nem várt események, ezek rendszerint Java kivételt kiváltó események, amelyek azonban nem vezetnek feltétlenül a program befejeződéséhez, ezek azonnal megjelennek a konzol kimenten.
WARN	Olyan kisebb hibák, mint például a hiányzó vagy nem megfelelően megadott bemeneti paraméterek.
INFO	Hasznos futás közbeni események (például: indítás/leállítás), ezek olyan alapvető információk, amelyek az alkalmazás normális működésével kapcsolatos információkat jelenítik meg a konzol kimeneten.
DEBUG	Kisebb, gyakran előforduló, de egyébként szokásos események. Annak eldöntése, hogy egy esemény jelentős–e, vagy sem sok mindentől függhet, például az időtől, az alkalmazás fejlesztési fázisától, a fejlesztő személyes ízlésétől stb. Általában azonban az események gyakorisága és hasznossága alapján különbséget kell tenni az INFO és a DEBUG szintek között.
TRACE	Részletes információk, amelyek csak a log–ban jelennek meg.
ALL	Ebben az esetben minden szint naplózásra kerül.
OFF	A naplózás kikapcsolása.

Az ALL és az OFF szintek csak menedzselési célokat szolgálnak, nem rendelkeznek a Logger osztályban kiíró metódusaival. [16] (Gülcü 2002)

5.4.1.2.Hierarchia

Az logger–ek és az appender–ek egy statikus hierarchiát alkotnak a Java objektumok mögött, amelyek a gyökere a log4j csomag. Az appender–ek

gyökere a `log4j.appender` csomag, a `logger`–eké pedig a `log4j.logger`. A hierarchia gyökér eleme a `root logger`, amely egy speciális objektum, minden `logger` őse, tehát ő minden log bejegyzést megkap, és feldolgoz. A `root logger`–t három dolog különböztet meg a többitől:

- mindig létezik
- a szintje nem lehet `null`
- nem lehet név alapján elkérni

A legelterjedtebb stratégia a `logger`–ek elnevezésére az osztályhierarchia pontos követése, és a `logger` deklarálása statikus változóként. Minden `appender`–hez megadhatjuk, hogy milyen szintű log bejegyzéseket írhatunk bele, így konfigurálhatunk egy adott osztályhoz egy általános `appender`–t, amely mindent megjelenít, illetve például egyet külön, ami az **`ERROR`** és **`FATAL`** hibákat elküldi egy **`SYSLOG`** szervernek, vagy beírja az alkalmazás adatbázisába.

5.4.2. Appenderek

A Log4J a log bejegyzések megjelenítésére számos kimeneti csatornát támogat, ezeket a csatornákat **`appender`**–eknek hívja. Az API számos `appender`–t biztosít, így a naplót írhatjuk például [20] (Wikipédia (angol) 2009) [18] (pcjuzer 2009):

- konzol kimenetre (`ConsoleAppender`)
- állományba (`FileAppender`) vagy syslog–ba (`SyslogAppender`, `NTEventLogAppender`)
- adatbázis táblába (`JDBCAppender`)
- elküldhetjük e–mailben (`SMTPAppender`)
- JMS keresztül (`JMSAppender`)
- `(/dev/null)` (`NullAppender`)

A Log4J megengedi, hogy egy `logger`–hez több `appender`–t is hozzárendeljünk, ekkor az üzenet valamennyi `appender`–re elküldésre kerül. A `logger`–hez bármikor hozzáadhatunk és elvehetünk `appender`–eket, ugyanakkor egy `logger` csak egy szintet használhat. Ha esetleg egy olyan eszközre akarunk

naplózni, amit az API nem támogat, akkor az Interneten minden bizonnyal találunk megfelelő appender–t, illetve ennek hiányában vagy speciális igény esetén írhatunk sajátot is.

5.4.2.1.Saját Appender készítése

Ha saját appender–t szeretnénk készíteni, akkor ehhez az AppenderSkeleton osztályt kell kiterjesztenünk, amely már rendelkezik a legtöbb appender által használt funkciókkal (például layout–ok kezelése). A leszármaztatott osztályban csak az append(LoggingEvent), a close() és a requiresLayout() metódusokat kell implementálni. Az általunk készített appender–t is ugyan úgy konfigurálni kell, mint a beépített appender–eket, ezt egy .xml konfigurációs állomány segítségével tehetjük meg.

5.4.3. Layoutok

A leggyakrabban nem csak a naplózás kimeneti csatornáját szeretnénk meghatározni, hanem a naplózásra kerülő adatokat formázni is szeretnénk, erre szolgálnak a layout–ok. A csatorna (appender) gondoskodik arról, hogy a naplózás a megfelelő kimenetre kerüljön, míg a formázásért a layout felel. A legtöbb layout nem kerül megosztásra az csatornák között, minden csatornának saját layout–ja van.

5.4.3.1.PatternLayout

A Log4J–ben van egy PatternLayout nevű általános layout, amely – a C nyelv printf függvényéhez hasonló módon – különböző konverziós karakterek segítségével lehetővé teszi a kimenetre történő kiírás formázását. A konverziós karakterek használata során a karakterek előtt a % jelnek kell szerepelnie.

Konverziós karakterek és hatásuk:

Konverziós karakterek	Hatás
c	a naplózási esemény kategóriája

C	az osztály neve, alapértelmezésként az osztály teljes nevét kiírja, ha például az osztályunk neve: „`org.apache.pm.HelloLog4J`” és a következő mintát használjuk %C{1} akkor a csak a „`HelloLog4J`” kerül kiírásra
d	az aktuális dátum, lehetséges formázási beállítások: %d{HH:mm:ss,SSS} vagy %d{dd MMM yyyy HH:mm:ss}, alapértelmezés szerint a formázás ISO8601 szerint történik
F	az állomány neve
l	a naplózási eseménnyel kapcsolatos információk kiírása a `JVM`–ből, ez rendszerint tartalmazza a hívó metódus teljes nevét, az aktuális sor sorszámát, stb.
L	az aktuális sor sorszáma a forrás fájlban
m	a naplóbejegyzés szövege
M	annak a metódusnak a neve, amelyikből a naplózási kérés származik
n	kiírja a platformfüggő sorelválasztó karakter(eke)t ("\n" vagy "\r\n") többsoros kiíratásnál ajánlott a sorelválasztáshoz ezt a formázási módot használni, mert így minden esetben helyesen kerül értelmezésre a sorelválasztó karakter
p	a naplózási esemény prioritása/szintje (a naplózás mélysége)
r	az alkalmazás indításától a naplózási esemény bekövetkezéséig eltelt milliszekundumok száma
t	az aktuális szál neve
x	a naplózási eseménnyel kapcsolatos szálhoz tartozó `NDC (nested diagnostic context)` kiírása
X	a naplózási eseménnyel kapcsolatos szálhoz tartozó `MDC (mapped diagnostic context)` kiírása, az X konverziós karakter után {} zárójelek között meg lehet adni egy kulcsot pld: %X{clientNumber} az `MDC` megfelelő értéke kerül a kimenetre
%	% % hatására egyetlen % jel jelenik meg a kimeneten

Megjegyzés: A hívó osztály információinak lekérdezése nagyon lassú, ezért ezeknek a konverziós karakterek a használatát (C, F, l, L, és M) lehetőleg kerüljük el.

Az első opcionális formátummódosító karakter a – jel, amely segítségével a kiírás balra igazítását tudjuk beállítani. A második egy egész szám, amely segítségével a minimálisan kiírásra kerülő karakterek számát adhatjuk meg, rövidebb karaktersorozat esetén balról szóközzel tölti fel a karaktereket, hosszabb karaktersorozat esetén az egész kiírásra kerül, nem történik levágás. Természetesen megadható a karakterek maximális száma is, ekkor abban az esetben, ha kiírásra kerülő karaktersorozat hosszabb, mint a megadott maximális érték, akkor a szöveg elejéről kerülnek levágásra a karakterek. Például: ha a karakterek maximális száma nyolc és a kiírandó karaktersorozat tíz karakter hosszú, akkor az első két karakter kerül levágásra és a többit – a harmadik karaktertől a végéig – fogja kiírni a kimenetre. [16] (Gülcü 2002)

Néhány példa:

Formátum módisító	Balra igazítás	Minimum szélesség	Maximum szélesség	Magyarázat
%20c	hamis	20	nincs	Balról szóközökkel egészíti ki a szöveget, ha a `logger` neve kevesebb, mint 20 karakter.
%–20c	igaz	20	nincs	Jobbról szóközökkel egészíti ki a szöveget, ha a `logger` neve kevesebb, mint 20 karakter.
%.30c	NA	nincs	30	Levágja a szöveg elejét, ha a hossza több mint 30 karakter.
%20.30c	hamis	20	30	Ha a `logger` neve rövidebb, mint 20 karakter, akkor azt balról szóközökkel egészíti ki. Ha hosszabb, mint 30 karakter, akkor az csak az utolsó 30 karaktert jeleníti meg.

5.4.3.2.XMLLayout

Az `XMLLayout` a naplóeseményeket egy rögzített formában tárolja, pontosabban az `XMLLayout` kimenete `<log4j:event>` elemek sorozatából áll,

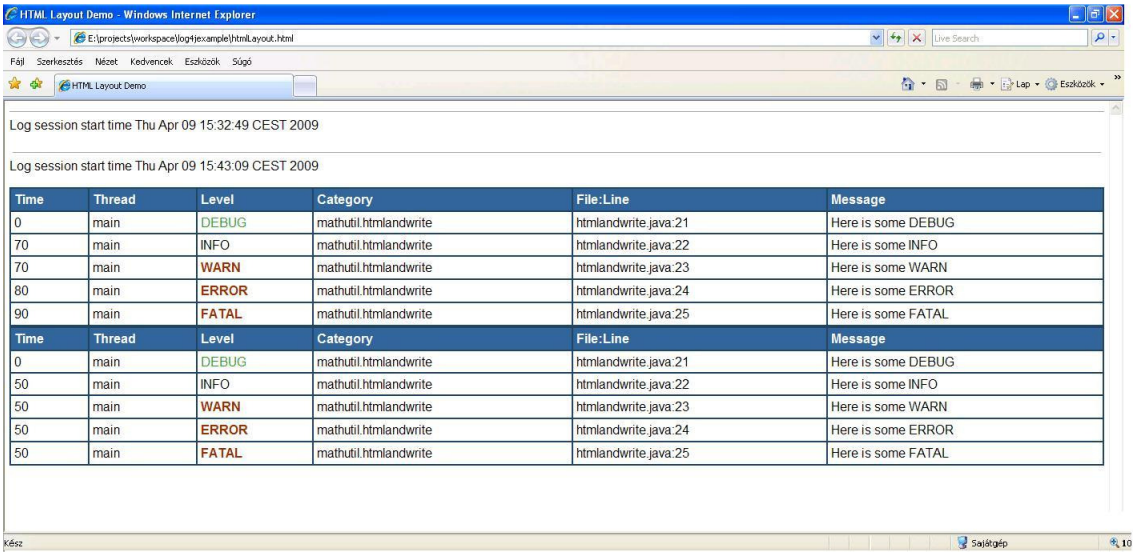
amely a `log4.dtd` állományban van definiálva. Lehetőség van arra is, hogy különböző konverziókat követően a naplózás eredménye egy `Swing` táblázatban jelenjen meg. Az `XMLLayout`–nak egy opciós beállítási lehetősége van:

Opció neve	Típus	Leírás
LocationInfo	boolean	Ha igaz, akkor a kimenet tartalmazza a hívó adatait is egy „File:Line” nevű oszlopban. Alapértelmezésben a kimenet nem tartalmazza ezeket az információkat.

5.4.3.3.HTMLLayout

A `HTMLLayout` kimenete egy rögzített formátumú táblázat, amelynek öt oszlopa van: **idő**, **szál**, **szint**, **kategória** és **üzenet**. A táblázat minden sora egy napló eseménynek felel meg. A `HTMLLayout`–nak két opciónális beállítási lehetősége van, ezek:

Opció neve	Típus	Leírás
LocationInfo	boolean	Ha igaz, akkor a kimenet tartalmazza a hívó adatait is egy „File:Line” nevű oszlopban.
Title	String	A generált HTML oldal címe.



HTMLLayout LocationInfo és Title opciók használatával

5.4.3.4.Saját Layout készítése

Ha saját `layout`–ot szeretnénk készíteni, akkor ehhez a `Layout` osztályt kell kiterjesztenünk, amely implementálja az `OptionHandler interface`–t. Az általunk készített `layout`–ot is ugyan úgy konfigurálni kell, mint a beépített `layout`–okat, ezt egy `.xml` konfigurációs állomány segítségével tehetjük meg.

6. Összefoglalás

A szakdolgozat elkezdése előtt, azt a célt tűztem ki magam elé, hogy a Java alapú webalkalmazás fejlesztéssel kapcsolatos ismereteimet elmélyítsem, az eddig megszerzett tudásomat kiegészítsem.

Az MVC tervezési sablon részletes megismerése kapcsán – hasonlóan az OO alapú programozáshoz – elmondhatom, hogy tapasztalatom szerint az a többletmunka, amelyet az alkalmazás tervezésénél be kell fektetni, az a tényleges fejlesztés megkezdésekor, illetve a későbbi karbantartás során mindenképpen megtérül. Az MVC ugyanolyan gondolkozásmódot igényel, mint az OO programozás, az elméletben megszerzett tudást a gyakorlatba is át kell tudni ültetni, amihez minél több webalkalmazás készítésével lehet megszerezni a megfelelő rutint.

A Struts a legismertebb MVC alapú framework – kvázi szabványnak is tekinthető – ezért is esett erre a framework–re a választásom. Bár a munkám során már részben sikerült megismerkednem a Struts működésével, mégis ez, és az MVC fejezet segített abban, hogy a technológiát teljes mértékben megismerjem. Különösen hasznos volt, a Struts komponensek és a konfigurációs állomány felépítésének részletes megismerése. Bár időről időre újabb framework–ök jelennek meg, úgy érzem, hogy a Struts megismerése ezen új technológiák elsajátításakor nagy segítségemre lesz.

A Log4J–vel kapcsolatban nagyon sok olyan új lehetőséget ismertem meg, amelyekről korábban nem tudtam, illetve a munkahelyem nem került alkalmazásra. Sikerült olyan új ismeretre is szert tennem (a különböző naplózási szinteknek megfelelő üzenetek eltérő állományokba történő írása), amelynek alkalmazását a jelenleg is futó fejlesztés során a project vezetőjének javasoltam. A jelenleg alkalmazott naplózás során ugyanis minden naplózásra kerülő esemény egy állományba kerül kiírásra, amely az én teszt környezetem esetében jelenleg 1,5 GB!

Úgy gondolom, hogy a Szakdolgozatom elején kitűzött célt többé–kevésbé sikerült elérnem, hiszen az egyes fejezetek feldolgozása során nagyon sok hasznos ismeretre tettem szert. A későbbiek során szeretném a webalkalmazás fejlesztéssel kapcsolatos ismereteimet tovább bővíteni újabb technológiákkal (például JSF, Hibernate) is megismerkedni.

7. Irodalomjegyzék

MVC

Szakkönyvek:

[1] Ford, Neal. *Art of Java Web Developement.* Manning, 2004.

Magyar nyelvű oldalak:

[2] *Elte.* 2009. http://ikon.inf.elte.hu/wiki/index.php?title=MVC.
[3] *Janka János.* 2009. http://jankajanos.spaces.live.com/Blog/cns!C3E2695FC6F7B0A4!394.entry.
[4] *Wikipedia (magyar).* 2009. http://hu.wikipedia.org/wiki/Modell–n%C3%A9zet–vez%C3%A9rl%C5%91#cite_note–0.

Angol nyelvű oldalak:

[5] *How to use Model-View-Controller (MVC).* 2009. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.
[6] *Trygve/MVC.* http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.
[7] *Wikipédia (angol).* 2009. http://en.wikipedia.org/wiki/Model–view–controller.

Struts

Szakkönyvek:

[8] Cavaness, Chuck. *Programming Jakarta Struts 2nd Edition* . O'Reilly, 2004, 2004.
[1] Ford, Neal. *Art of Java Web Developement.* Manning, 2004.
[9] Hightower, Richard. *Jakarta–Struts Live* . SourceBeat, 2004.
[10] Shenoy, Srikanth. *Struts for J2EE Developers* . ObjectSource, 2004.
[11] Sincock, Austin. *Enterprise Java for SAP* . Apress, 2003.
[12] Spielman, Sue. *The Struts Framework: Practical Guide for Java Programmers.* Morgan Kaufmann Publishers, 2003.

Angol nyelvű oldalak:

[13] *Struts Wiki.* 2009. http://wiki.apache.org/struts/StrutsTutorials.
[14] *RoseIndia.* 2009. http://www.roseindia.net/struts/.
[15] *Struts Tutorial.* 2009. http://www.visualbuilder.com/jsp/struts/tutorial/.

Log4J

Szakkönyvek:

[16] Gülcü, Ceki. *The Complete log4j Manual.* QOS, 2002.

Magyar nyelvű oldalak:

[17] *pcjuzer.* 2009. http://pcjuzer.blogspot.com/2007/10/krkp.html.
[18] *pcjuzer.* 2009. http://pcjuzer.blogspot.com/2007/10/log4j-jdbcappender.html.
[19] *Viczián István.* 2009. http://www.codexonline.hu/CodeX6/alap/java/ViczianIstvan/log4j.htm.

Angol nyelvű oldalak:

[20] *Wikipédia (angol).* 2009. http://en.wikipedia.org/wiki/Java_logging_framework.