Testreszabott JSP-eljárások

Hogyan egyszerűsítsünk összetett Java-kódot?

A Linuxvilág előző számaiban a kiszolgálóoldali Java-alkalmazásokkal ismerkedtünk. A vizsgálódást a servletekkel, ezekkel a különleges Java-osztályokkal kezdtük, melyek a servlettárolóból hajtódnak végre. A programozókat a servletek használata nyilván nem rettenti el, a grafikusok véleménye viszont e tekintetben megoszlik.

Ezt a gondot oldja meg a JavaServer Page-módszer (JSP-k) használata, amely a Javát a HTML-lel egyesíti, miközben a Microsoft Active Server Pages-hez (ASP) vagy a mod_perl-hez tartozó, nyílt forráskódú HTML::Mason-rendszer formai követelményeihez hasonló nyelvet használnak. Minden JSP valójában egy álcázott servlet; a lapot a JSP-motor servletté, majd pedig Java .class állománnyá alakítja.

A JSP-k Java-kódot is tartalmazhatnak, ami egyszerűbbé teszi az összetett műveletek elvégzését. Egy bizonyos ponton túl azonban a kód elnyomja a HTML-t, ezáltal a JSP fenntarthatatlanná válik.

A nem programozók sem kedvelik, ha nagy mennyiségű kódot találnak a JSP-ben, így viszont a JSP előnye erősen csökken az egyszerű servletekkel szemben.

Előző írásunkban az olyan módszerek egyikét vizsgáltuk meg, amivel elkerülhetjük a JSP-n belüli kódhasználatot, nevezetesen a JavaBeans alkalmazását. Egyszerű XML-alapú tagok használatával akár egy nem programozó is képes összerakni egy összetett viselkedésű JSP-t anélkül, hogy egyetlen sor kódot is írnia kellene. A JavaBeans valódi ereje nem magukban a babokban (JavaBeans magyarul „Java Babok”-at jelent) rejlik, hanem azokban a különleges tagokban, amelyek segítségével oly könnyen előhívhatjuk azokat.

Most azt vizsgáljuk meg, hogyan írjunk saját – ahogy mondani szokás – „testreszabott eljárást”, azaz olyan XML-alapú tagokat, amelyek lehetővé teszik, hogy Java-osztályokkal és eljárásokkal dolgozzunk anélkül, hogy magával a Javával egyáltalán találkoznánk. Példáinkat a nyílt forráskódú Jakarta-Tomcat servlet és JSP-megvalósításhoz terveztük. Mindazonáltal valószínűleg bármely olyan JSP-megvalósítással működnek, amelyek a testreszabott eljárásokat támogatják.

Több oka is lehet, amiért testreszabott eljárásokat érdemes használni. Az első, hogy csökkentik a JSP-be illesztendő Java-kód mennyiségét, ezáltal azok könnyebben olvashatóvá, érthetővé és karbantarthatóvá válnak. A testreszabott tagok ráadásul sokkal kevésbé bonyolultak, mint egy Java-kód, így azokat szélesebb felhasználói réteg használhatja. Végezetül minden testreszabott tagkönyvtár egyetlen központilag írt és fenntartott Java-osztályra mutat. Testreszabott eljárások használatával a honlap pontosan olyan tagok könyvtárát állítja elő, amelyek éppen szükségesek. Egyetlen Java-programozó számos tervezőgrafikus és JSP-felhasználó számára készíthet és adhat ki tagkönyvtárakat. Amint azt látni fogjuk, a testreszabott tag ugyan nem csodaszer, de nagyon hasznos; a magam részéről ezt tartom az egyik legfontosabb érvnek, és emiatt érdemesebb JSP-t használni a versenytárs módszerekkel szemben.

Melyek is azok a testreszabott eljárások?

A testreszabott eljárások JSP-ből rövidebb utat biztosítanak a Java-kódhoz. Bármit, amit testreszabott eljárással meg lehet tenni, az megoldható `<% %>` tagok közti Java-kóddal is. Hiszen – mint tudjuk – a JSP servletté alakul, mielőtt a végfelhasználó számára lefordítódik és végrehajtódik.

Ahogy azt a múlt hónapban a JavaBean-tagok esetében láthattuk, a testreszabott eljárásokat HTML helyett egyszerű XML-tagokkal határozhatjuk meg. Ez elsőre némileg zavaró és elkeserítő lehet, különösen azoknak, akik hibás HTML-írási szokásokat vettek fel. A következő sor például helyesnek tűnhet:

```
<P><jsp:getProperty name="simple"
➥property="userID"></P>
```

Valójában azonban a fenti sor nem fog működni és kivételt, illetve veremkövetést (stack trace) vált ki a JSP-ben. Ez azért van így, mert XML-ben minden tagot le kell zárnunk valahogyan. Ha a `<tag>`-nak nincsen lezáró `</tag>` párja, akkor jelölni kell, hogy saját magát zárja le: `<tag/>`. A fenti sort tehát így kell helyesen írni:

```
<P><jsp:getProperty name="simple"
➥property="userID"/></P>
```

A testreszabott eljárás tulajdonképpen csak formaikövetelmény-máz a Java-tagfüggvényeken. Minden egyes tagkönyvtár egy-egy eljáráskészletet határoz meg, a jsp tagkönyvtár például három eljárást ad meg: `getProperty`, `setProperty` és `useBean`. Minden eljárást egy külön Java-osztály határoz meg, amit tagkezelőnek (tag handler) nevezünk.

A tagkönyvtár meghatározásához először egy XML fájlt készítünk, amelyet tagkönyvtár-leírónak (tag library descriptor, azaz TLD) neveznek. A TLD az eljárásokat a megfelelő tagkezelő osztályhoz rendeli, felsorolva a választható és kötelező értékeket, illetve a taghoz kötődő egyéb adatokat.

Ha JSP-ből szeretnénk használni a testreszabott eljárást, TLD-nk betöltésére különleges vezérlőjelet használunk. Ez segít a JSP-motornak, hogy a JSP-nkben talált testreszabott tagokat ellenőrizze, illetve a hozzájuk tartozó kezelőosztályt megtalálja.

Egyszerű testreszabott eljárások

Itt az ideje, hogy egy olyan egyszerű eljárást határozzunk meg, ami alapján érthetővé válik a tagkezelő osztályok, a TLD-k és a JSP-k működése mögött rejlő szerkezet. Testreszabott eljárásunk legyen a *hello* tag, amely kezeli az elhagyható *firstname* értéket. Ha a *firstname* létezik, a tag egyszerű üdvözlő üzenetet küld a megnevezett felhasználónak; ha pedig az érték hiányzik, a tag egy általános üzenetet készít.

Az első lépés egy egyszerű tagkezelő írása, amely ezt a feladatot megvalósítja. Ilyen tagkezelőt mutat be az *1. lista* a *HelloTag* osztály meghatározásával. A HelloTag.java forrásfájlt az összes JSP és servlettel kapcsolatos osztállyal együtt a `$TOMCAT_HOME/classes` könyvtárba helyeztem. Mivel a HelloTag.java az il.co.lerner csomagban található, és miután a `$TOMCAT_HOME` az én gépemen a /usr/java/jakarta-tomcat-3.2.1 könyvtár, a Java-forrás nálam a következő helyen található:

```
/usr/java/jakarta-tomcat-
3.2.1/classes/il/co/lerner/HelloTag.java
```

1. lista HelloTag tagkezelő

```
package il.co.lerner;

import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HelloTag extends TagSupport {

  private String firstname = null;
  // null alapértékű mező
  // ----------------------------------------
  // Bean-stílusú "set" tagfüggvény

  public void setFirstname (String newFirstname)
                           {

     // Csak akkor állítjuk be a firstname
     // mezőt, ha nem üres.
     if (! newFirstname.equals(""))
         {
              firstname = newFirstname;
         }
  }

  // ----------------------------------------

  public int doEndTag() throws JspException {

     // Szúrjunk be némi szöveget a JSP-be,
     // vagy lépjünk ki!
     try {

          // Ha nem volt név megadva, általános
          // üzenetet adunk
          if (firstname == null)
               {
                  pageContext.getOut().println
                    ("Üdvözlöm!");
               }
          else
               {
                  pageContext.getOut().println
                    ("Üdv, " + firstname +
                     "!");
               }

     } catch (IOException e) {
       throw new JspException(e.getMessage());
     }

        // Haladjunk tovább, és értelmezzük
        // a JSP további részeit!
        return EVAL_PAGE;
  }

  // ----------------------------------------

  public void release()
  {
       firstname = null;
       super.release();
  }

}
```

A HelloTag.java HelloTag.class osztállyá történő fordítása után ezt a tagkezelőt egy vagy több más tagkönyvtárral akár egybe is lehet olvasztani. Minden tagkezelő osztálynak tartalmaznia kell a két különböző alapcsatolófelület (*Tag* vagy *BodyTag*) egyikét. (Utóbbi azoknál a testreszabott eljárásoknál használatos, amelyeknek nyitó és záró tagjuk között „testük” van, ellentétben azokkal, amelyeknek egyáltalán nincs testük, és ezekkel e hónapban ismerkedünk meg.)
A gyakorlatban ezeket a csatolófelületeket nemigen van értelme megvalósítani. Sokkal egyszerűbb és célszerűbb a *TagSupport* és *BodyTagSupport* osztályoktól örökölni őket, amelyek a megfelelő alapértelmezett csatolófelületeket tartalmazzák. A *TagSupport* alosztály-lehetőséget kihasználva megtakaríthatunk némi munkát, és csak azokat a tagfüggvényeket írjuk felül, amelyeket az alapértelmezettől eltérő módon szeretnénk felhasználni. Végezetül HelloTag-megoldásunk mindössze három tagfüggvényt tartalmaz: *setFirstname doEndTag* és *release*.
Az első tagfüggvény, a *setFirstname*, úgy néz ki és úgy is működik, mint egy JavaBean tulajdonságbeállító tagfüggvény, egyetlen értéket fogad el és *void*-ot ad vissza. A setFirstname önműködően hívódik meg, amikor a JSP-motor a *firstname* értéket észleli saját JSP-eljárásunkban. Az érték a tagban átadott adattal töltődik fel, és akárcsak a JavaBeansben, a *firstname*-et beállító tagfüggvény nevének *setFirstname*-nek kell lennie, ahol a nagy *F* betű kötelező.
Második tagfüggvényként a *doEndTag* akkor hívódik meg, amikor a JSP-motor eléri testreszabott eljárásunk bezáró tagját. A *doEndTag* tagfüggvény nem vár értéket és egy egész számot ad vissza. Mi azonban egész szám visszaadása helyett a számunkra biztosított egyik állandót adjuk vissza. Általános esetben `EVAL_PAGE`-et adunk vissza, amely értesíti a JSP-motort, hogy folytathatja annak a JSP-nek az értelmezését, amelyből testreszabott eljárásunk meghívódott. Ha a JSP-motor fájlértelmezését le szeretnénk állítani, akár azért, mert hibát találtunk, akár mert a felhasználót egy másik címre szeretnénk irányítani, egyszerűen `SKIP_PAGE`-et adjunk vissza.
A *doEndTag* belsejében bármilyen Java-kódot elhelyezhetünk. Az általunk készített változókon kívül magáról a JSP-ről szóló adatokhoz is hozzáférhetünk, ideértve a kapott HTTP-kérelmet és -választ.
A felhasználó böngészőjére úgy írhatunk adatokat, hogy a testreszabható tagot HTML-, XML- vagy egyszerű szöveggel helyettesítjük. (A testreszabott eljárások többnyire egyszerű szöveget adnak vissza és a szöveg pontos formázását a JSP szerzőjére bízzák.)
A *TagSupport* szuperosztályban bevezetett *PageContext* objektum használatával megkaphatjuk a kimeneti folyamot és adatokat is küldhetünk rá:

```
pageContext.getOut().println("Hi there!");
```

Végül meghatározzuk a *release* tagfüggvényt, amely semmilyen értéket nem fogad és *void*-ot ad vissza. A `release()` a testreszabott eljárás végrehajtásának végén hívódik meg, ezáltal lehetőséget ad a tagkezelő osztálynak, hogy kitakarítson maga után. Ez általában annyit jelent, hogy az összes belső változót *null*-ra állítja, de adatbáziskapcsolat lezárása vagy valamilyen adat hibanaplóba küldése is állhatna itt. A HelloTag.java-ban egyszerűen csak *null*-ra állítjuk a *firstname*-et,

majd arra kérjük a szuperosztályt, hogy saját változóit nullázza.
Most hogy megértettük, hogyan működnek az egyes HelloTag tagfüggvények, már csak az a kérdés: miképpen működnek együtt? Ha a JSP (az alább ismertetett TLD-n keresztül) osztályunkhoz rendelt testreszabott eljárást tartalmaz, az eljárás összes értéke meghívja osztályunk megfelelő *set* tagfüggvényét. Ha valaki például a `firstname="foo"` értéket adja át, akkor tulajdonképpen a *setFirstname("foo")* tagfüggvényt hívja meg.
Mivel azt szeretnénk, hogy a *firstname* érték elhagyható legyen, amikor először elkészítjük, `null` alapértéket adunk neki. Amikor a JSP-motor a testreszabott eljárás értelmezését befejezi, meghívja a *doEndTag*-et és megvizsgálja a firstname értékét. Amennyiben a *firstname* `null`, akkor az általános *(Üdvözlet)* üzenetet küldi a végfelhasználónak. Ha viszont nem `null`, akkor az értéket a *doEndTag* egy valamivel személyesebb üzenethez használja fel. Amikor a testreszabott eljárás végrehajtása befejeződik, a JSP-motor meghívja a `release()`-t és alaphelyzetbe állítja a *firstname* változót, valamint néhány egyéb objektumot.

TLD írása

Ha az osztály megírásával elkészültünk, TLD-t hozhatunk létre, amely leírja azt a JSP-motornak. Néhány esetben kedvezőbb a fordított irány, ha a TLD-t használjuk leírásnak, amiből a JSP-szerzők és a tagkezelőírók párhuzamosan tudnak dolgozni. Én jobban szeretem előbb megírni a testreszabott eljárást és menetközben módosítgatni a TLD-t, annak ellenére, hogy ez nyilvánvalóan nem éppen a legbiztonságosabb és nem is a legelegánsabb munkamódszer.
A TLD, amint az a 2. listában látható (15. CD, Magazin/JSP könyvtár), egy viszonylag rövid XML fájl, amely az eljárásneveket a megvalósítást végző osztályokhoz rendeli. A TLD egyetlen eljárást egyetlen osztályhoz vagy akár ezernyi eljárást ezernyi különféle osztályhoz rendelhet. Mivel minden osztály elkülönítve létezik, még az is lehetséges (bár nem tűnik túl jó ötletnek), hogy az osztályt egyszerre több TLD-ben is felhasználjuk.
A TLD az első hivatkozás után servlettárolónkba töltődik. Ez azt is jelenti, hogy ha egy már betöltődött, testreszabott eljáráshoz tartozó TLD-t változtatunk meg, akkor sajnos újra kell indítanunk a Tomcatet (és az Apache-t, ha az Apache mod_jk-t használjuk a Tomcat kiszolgálóhoz). A JSP-motor ilyenkor tudja meg, hogy tagkönyvtárunk melyik változatot és szabványt támogatja. Ezáltal lehetővé válik, hogy a JSP-motor kitalálja, szükséges-e az adott könyvtárat frissíteni vagy sem, hogy a jelenlegi szabvánnyal összeegyeztethető legyen.
A TLD egy legfelsőbb szintű `<taglib>` tagból áll, amely legalább négy alrészt tartalmaz:

- a `<tlibversion>` tartalmazza, hogy ez a könyvtár mely változatú tagkönyvtár-szabványnak felel meg;
- a `<jspversion>` azon JSP-előírások változatszámát adja meg, amely szerint ez a könyvtár készült;
- a `<shortname>` nevet ad az adott tagkönyvtárnak, amelyet néhány JSP-motor ki is használ; végül minden, a tagkönyvtárban tárolni kívánt tagkezelőnél egyszer szerepel a `<tag>` tag. Minden tag saját nevet kap: a meghívandó eljárás nevét. Így ha egy tagkönyvtárat az *abc*-előtaggal importálunk, a *hello* nevű tag *abc:hello*-ként hívódik majd meg. A `<tagclass>` alrész a tagnevet az eljárást végző tagkezelő osztályhoz rendeli; ennek az osztálynak nyilvánvaló módon a kiszolgáló `CLASSPATH`-jában kell lennie. Az `<info>` alrész módot ad arra, hogy néhány, a taggal kapcsolatos alapadatot és sorközi leírást adjunk közre.

Végül az összes paramétert elnevezzük, amit testreszabott eljárásunk elfogad. Minden értéknek saját `<name>` tagja és egy jele van, ami megmutatja, elhagyható-e az adott érték.

Testreszabott eljárások használata JSP-kben

Most, hogy elkészítettük a TLD-t és a tagkezelőt, ezeket bármely JSP-nkben felhasználhatjuk. A tagkönyvtárat a `taglib` nevű egyedi JSP-utasítással importálhatjuk:

```
<%@ taglib uri="/WEB-INF/hello.tld"
➥prefix="hello" %>
```

Figyeljük meg, hogy a `taglib` utasítás két értéket vár, egy *uri* és egy *prefix* nevűt. Az *uri*-rész határozza meg a pillanatnyilag készített TLD fájl nevét. Ha a TLD-ket a `WEB-INF` könyvtárban szeretnénk elhelyezni, akkor a fenti formai követelmény teljes mértékben helyes. A *prefix* érték tulajdonképpen névtér- (namespace) meghatározás, ami tudatja a JSP-motorral, hogy a tagkönyvtárból importált eljárásoknál milyen előtagot fogunk használni. Azáltal, hogy a JSP-nek adtuk meg ezt a megkülönböztető előtagot (ahelyett, hogy a tagkönyvtárba fordítottuk volna), lehetővé válik, hogy egyszerre több tagkönyvtárat is importáljunk anélkül, hogy a névütközések miatt aggódnunk kellene.
Mivel TLD-nk mindössze egyetlen *hello*-tagot tartalmaz, és mivel a tagkönyvtárat a *hello*-előtaggal importáltuk, *HelloTag* tagfüggvényünket a következő formátumban hívhatjuk meg: `<hello:hello/>`. A *3. lista* a tag használatát bemutató teljes JSP-t tartalmazza (test-tag.jsp).
Ne feledjük el a testreszabott eljárások meghívásánál a lezáró perjelet kitenni! Ha erről megfeledkezünk, a Tomcat JSP-motor (más néven Jasper) a következő hibaüzenetet fogja visszaadni:

```
Unterminated user-defined tag:
ending tag &lt;/hello:hello&gt; not found or
incorrectly nested
```

A TLD szerint a *firstname* érték elhagyható. Amennyiben nem adjuk át a *firstname* értéket, a következő kimenetet kapjuk a böngészőn:

```
Testreszabott eljárásteszt.
Üdvözlöm!
```

A *firstname* értéket azonban át is adhatjuk:

```
<hello:hello firstname="Reuven"/>
```

Ha a fenti sort a JSP-be helyezzük, a böngészőre a következő kimenet kerül:

```
Testreszabott eljárásteszt.
Üdv, Reuven!
```

Összetettebb testreszabott eljárások

A fenti példa meglehetősen egyszerű testreszabott eljárás volt. A testreszabott eljárás-tagok sokkal többre is képesek, mint egyszerűen neveket írni a képernyőre. Például az objektumok adatbázisokkal léphetnek kapcsolatba, ott adatokat kérhetnek le (vagy tárolhatnak) anélkül, hogy közvetlen Java-hívásokat helyeznénk a JSP-be. Sőt, a testreszabott eljárásokat ismétlőelemekként vagy feltételes végrehajtásra is használhatjuk.
Hogy ezeket az összetettebb eljárásokat megvalósíthassuk, kihasználjuk, hogy a tagkezelő osztály a testreszabott eljárás testében is tud keresni; azaz abban a szövegben, ami az eljárás nyitó- és zárótagja közt található. Ezzel a szöveggel bármit megtehetünk: többszöröz-hetjük, feltételes elágazást készíthetünk vagy akár meg is kérhetjük a JSP-motort, hogy értelmezze a tartalmat, mielőtt átadná a tagkezelőnek. Még a tagok egymásba ágyazása is lehetséges, ahol az egyik eljárás hatékonyan adhat át értékeket a másiknak.

3. lista test-tag.jsp

```
<%@ taglib uri="/WEB-INF/hello.tld"
prefix="hello" %>

<HTML>
<Head>
    <Title>Testreszabott eljárásteszt</Title>
</Head>

<Body>
    <P>Testreszabott eljárásteszt.</P>

    <P><hello:hello/></P>

</Body>

</HTML>
```

Számos nyílt forrású tagkönyvtár létezik, ideértve azt is, amelyet a Jakarta Project támogat, és amely ügyesen használja ezeket a lehetőségeket, számos taggal növelve a lehetőségek tárházát.

Mire jók a testreszabott eljárások?

A testreszabott eljárások félelmetesen hatékony eszközök. A JSP-be helyezett közvetlen Java-kóddal szemben előnyök széles skáláját biztosítják: összetett feladatokat zárnak könnyen megjegyezhető tagokba, ami a nem programozók számára is lehetővé teszi, hogy adatbázisokkal vagy hasonló nem egyértelmű rendszerekkel dolgozzanak.

A testreszabott eljárásoknak azonban árnyoldaluk is van, melynek nyitja pontosan a „testreszabott” szóban rejlik. Az, hogy a JSP-k belsejében saját tagokat adhatunk meg, ügyes és kifinomult eszköz, és a honlap minden fejlesztője számára előnyökkel jár. Csakhogy a Web egyik legnagyszerűbb tulajdonsága éppen az, hogy meglehetősen szabványosított. A testreszabott eljárások segítségével ráadásul akár egy teljes, önálló nyelvet is készíthetünk Javában és tagkezelő osztályoknak feleltethetjük meg. *Hans Bergsten*, akinek JavaServer Pages című könyve kitűnő forrásmunka a JSP-k utasításai terén, ezt az alapötletet egészen a végletekig követi – tulajdonképpen teljes egészében feleslegessé téve a Java-használatot a JSP-kben. Engem viszont aggodalommal tölt el, ha egy ilyen viszonylag megbízható és jól ismert nyelvet, mint a Java, egy új, sokkal kevésbé ismert és sokkal kevésbé harcedzett nyelvvel (saját testreszabott könyvtárral) helyettesítenek.

Ha egy nagy cégnél dolgoznék, amely a Java, a servletek és a JSP felhasználását komolyabban tervezi, a testreszabott eljárások használatát valószínűleg nagyon kényelmesnek találnám. Egy ilyen cég megteheti, hogy elkészíti a saját tagkönyvtárát, amelyet aztán a honlap teljes élettartama alatt felhasználhat, valamint hogy saját szabványt alkot dolgai működtetésére.

Azok számára azonban, akik nem nagy cégeknél dolgoznak, vagy akik számos különböző ügyféllel állnak kapcsolatban, az átalakíthatóság a legfontosabb szempont. Ha összes ügyfelem saját honlapjához más és más testreszabott eljáráskészletet szeretne meghatározni, igencsak bajban lennék, hisz emlékeznem kellene, hogy hol melyik tagot és értéket kell használnom a ciklusokhoz, az adatbázisokhoz vagy az elágazásokhoz.

Továbbá, mint azt már korábban említettem, a nem programozók miatt is aggódom, akik már eleve sokat küszködtek, mire a HTML-lapokba ágyazott Javát megtanulták használni – és most, hogy két különféle ciklust is meg kell tanulniuk (egy javásat, és egy másikat, amit a testreszabott eljárás biztosít), szinte biztosan összezavarodnak.

Viszonylagos megoldást nyújthat egy nagy, szabványos testreszabott eljáráskészlet alkalmazása, amely a JSP-szabvány része lenne, valahogy úgy, ahogyan az a JavaBean-tagokkal is történt. A Bergsten-féle JavaServer Pages című könyvben szereplő könyvtár jó kiinduló alap lehetne, de ez csak egy a számos elérhető könyvtár közül. Jó volna látni, hogy a JSP-közösség ebben a témában összefog, mielőtt tucatnyi hasonló, de egymásnak még véletlenül sem megfelelő könyvtárral találkoznánk, melyek közül jó néhány kétségtelenül jogvédett lesz.

Összegzés

A JSP-kkel való munka hatékony és gyors módszer a kiszolgálóoldali Java-alkalmazások készítésében, különösen az olyan nem programozók számára, akik semmilyen nyelvet nem szeretnének megtanulni. A testreszabott eljárások – különösen JavaBean-elemekkel együttműködve – lehetővé teszik, hogy összetett feladatokat kevés kóddal oldjunk meg. Egy kis megfontoltsággal a honlap akár úgy is elkészülhet, hogy a JSP-kbe egyetlen Java-sor sem kerül, mert azok teljes egészükben testreszabott eljárásokon és tagkönyvtárakon alapulnak. A webhelyek (és a testreszabott eljárásokat használó tanácsadók) azonban nem árt, ha figyelmet fordítanak rá, hogy a tagkönyvtárak kényelmét és hatékonyságát élvezve esetleg egy új programozási nyelvet alkotnak. Ha óvatlanok vagyunk, a testreszabott tagok megoszthatják az ügyféloldali Java-közösséget, különböző összeférhetetlen könyvtárakat használó alközösségekre tördelve azt szét.

Reuven M. Lerner (reuven@lerner.co.il) kisebb webes és internetes módszerekkel foglalkozó tanácsadó cég tulajdonosa és vezetője. A cikk megjelenésének időpontjában valószínűleg már végleg elkészült Core Perl című könyvével, melyet idén jelentet meg a Prentice-Hall. Az ATF honlapon érhető el (➲ http://www.lerner.co.il/atf/).

Kapcsolódó címek

Az Apache Programalapítvány Jakarta Projectje, mely számos Java-vonatkozású fejlesztésnek ad otthont, a ➲ http://jakarta.apache.org/ címen található.

A Tomcat – a Jakarta Project része – a Sun servlet és JSP-előírásnak megfelelő nyílt forráskódú megoldáson dolgozik. A Jakarta-honlap mindig a Tomcat legújabb forrás-, bináris és fejlesztői állományait nyújtja az érdeklődőknek.

JSP-témában kitűnő forrás az O'Reilly kiadásában nemrégiben megjelent JavaServer Pages című mű *Hans Bergstentől*. Véleményem szerint a könyv túlságosan is a testreszabott eljárások alkalmazása felé hajlik, de ha ilyenfajta megoldást szeretnénk alkalmazni, vagy egyszerűen csak arra vagyunk kíváncsiak, hogyan lehet JSP-ket írni és használni, kezdetnek mindenképpen kitűnő.

Számos, a testreszabott tagokról és a JSP-kről szóló hálózati (online) útmutató is létezik. A Jakarta-Tomcat honlap szintén rendelkezik egy ilyen ismertetővel, ➲ http://jakarta.apache.org/taglibs/tutorial.html. Ebben a témában egy másik tájékoztató egyenesen a Suntól is letölthető a ➲ http://java.sun.com/products/jsp/tutorial/TagLibrariesTOC.html címről.