JavaBeans

A babok használata a programozók minden rétege számára leegyszerűsíti a JSP-oldalakkal való munkát.

A Kovácsműhely legutóbbi két írásában a Jakarta-Tomcat nyílt forráskódú servlettel és az Apache Software Foundation által támogatott JavaServer Pages (JSP) motorral foglalkoztunk. Mint láthattuk, JSP-hez servletet írni nem is olyan nehéz és nem is annyira időigényes feladat. Az egyetlen nehézséget talán a kiszolgálóoldali Java használatának elsajátítása okozhatja. A servletek ugyan a Java-programokból elérhető kifejezőerő és hatékonyság teljes kínálatát nyújtják, de arra kényszerítenek, hogy viszonylag alacsony szinten gondolkozzunk. Valahányszor csak egy HTML formátumú szöveget szeretnénk a felhasználó böngészőjére küldeni – ami, ugye, meglehetősen gyakran megesik –, a PrintWriter objektumot kell használnunk, ami a HTTP válasz-objektumhoz van rendelve:

```
PrintWriter out = response.getWriter();
out.println("<HTML><Body>This is illegal HTML
            </Body></HTML>");
```

Ebben az esetben a JSP-k siettek a segítségünkre – ezek feltételezik, hogy amit nem végrehajtható kódként jelöltünk, azt egy az egyben kell elküldeni a felhasználó böngészőjére. Ezáltal azonban rögvest új gond keletkezett: ha például a JSP egy relációs adatbázis-kezelőhöz szeretne csatlakozni, akkor Java-kódsorok tucatjait, esetleg százait szükséges beilleszteni.
A megoldás az, hogy a JSP-n kívüli kódcsomagokat készítünk, melynek tagfüggvényeit olyan formai követelményekkel hívhatjuk meg, amelyek inkább hasonlítanak a HTML-re, mint a Javára. Ezek a JavaBeansnek nevezett kódcsomagok megkönnyíthetik a JSP-kel való munkát – akár egy magasabb szinten dolgozó, tapasztalt programozó számára, akár egy tapasztalatlan programozónak, aki inkább az alkalmazhatóság előnyeit szeretné kihasználni.
Ebben a hónapban a JavaBeans birodalmában teszünk egy rövid körutat. Írunk néhány saját babot, beépítjük őket a JSP-be, és megvizsgálunk egy-két ehhez kapcsolódó hibát és csapdát.

Mi is az a bab?

A babok létrehozójának szemszögéből a JavaBeanek bizonyos feltételeknek megfelelő Java-osztályok. (Hamarosan megvizsgáljuk, mik is pontosan ezek a feltételek.)
A JSP-t írók számára viszont a babok olyan különleges tárolóosztályt jelentenek, amelyekben bizonyos adatokat tárolhatunk, illetve kaphatunk vissza. Az adatok egyes darabjait *tulajdonságoknak* nevezzük, amelyeket külön-külön beállíthatunk, illetve lekérdezhetünk. Ugyan nem minden tulajdonságot lehet lekérdezni vagy beállítani, a babok felhasználói felülete JSP-oldalról egységes és könnyen érthető.
Mivel a babok csak kötött műveletkészletet fogadnak el, létezik néhány különleges JSP-tag, amelyek segítségével használhatjuk őket. Ezeknek a tagoknak az alkalmazásával mérsékelhetjük a közvetlenül JSP-be helyezett Java-kód mennyiségét. Ez nem csupán azt jelenti, hogy csökken a zavar és átláthatóbbá válik a kód, hanem egyúttal a nem programozók számára is lehetővé teszi, hogy a babok erejét kihasználják anélkül, hogy meg kellene tanulniuk Javában programozni.

Nem szokatlan jelenség, hogy egy JSP-ben egyszerre több babot is felhasználnak, szükség szerint tárolva és visszaszedve a különféle tulajdonságokat. Például egy hálózati áruház bevásárlókosara használhat egy babot a leltár tárolásához, egy másikat a felhasználó kosarához, és még továbbiakat a felhasználó nyelvének követéséhez vagy a fizetés és kiszállítás beállításaihoz. Ezek a babok mind külön Java-osztályok, kezelésüket azonban különleges JSP-tagok végzik, amelyek a JSP-t író elől elrejtik sokrétűségüket.
Természetesen a babokat több honlapon (vagy egy honlap több részében) is fel lehet használni. Ha egyszer már létrehoztunk egy hasznos JavaBeant, amely érdekes képességekkel bír, mások is könnyedén behelyezhetik azt saját Java-osztály keresési utunkba (classpath), és máris JSP-ből élvezhetik ezek előnyeit.

Babok készítése

A bab létrehozásához először írnunk kell egy java.io.Serializable csatolófelületet megvalósító Java-osztályt, azaz olyan babot készítünk, amely képes lemezre írni és visszaállítani önmagát. Ha osztályunk mezői egyszerű Java-típusok, például egészek és karakterláncok, akkor a Serializable megvalósítás nem igazán fontos.
Az *1. lista* egy egyszerű babmegvalósítást mutat be. Ez a bab egyetlen példányváltozót (`userID`) és két tagfüggvényt tartalmaz. A `getUserID` tagfüggvény a `userID` pillanatnyi értékét adja vissza, míg a `setUserID` tagfüggvény e mező értékét állítja be. Mivel ezek a tagfüggvények a SSP-kben használatos formátumoknak megfelelnek, JSP-inkből felhasználhatjuk őket.
A saját Apache projekt Jakarta-Tomcat servlet, illetve JSP-rendszer 3.2-es változatát futtató rendszeremen Java-osztályaimat a $TOMCAT_HOME/classes könyvtárba kellett elhelyeznem. (A $TOMCAT_HOME egy környezeti változó, amely a Tomcat-telepítés helyére mutat; saját rendszeremen értéke /usr/java/jakarta-tomcat-3.2.1/). Ha ez a „classes” könyvtár létezik, akkor hozzáadódik a Tomcat `CLASSPATH` környezeti változójához, így az új osztályok számára kényelmesen használható helyet hoz létre.
Az osztály maga igen egyszerű, a különböző típusú tagfüggvények bemutatásához a következőket kell elkészítenünk:

1. A bab létrehozóját (constructor), amely nem kap egyetlen értéket sem, és egy vagy több mezőt állít be. A mi példánkban a SimpleBean létrehozó a `userID` változót állítja az alapértelmezés szerinti nullára.
2. Tulajdonságlekérdező tagfüggvényt, amely értéket ad vissza a hívónak. Akárcsak a bab létrehozója, a tulajdonságlekérdező tagfüggvény sem vár értéket.
3. A tulajdonságbeállító tagfüggvényt, amely egyetlen értéket vár (az új értéket), a hívónak azonban nem ad vissza semmit.

Ne feledjük, a babok ugyanolyan osztályok, mint a legtöbb Java-osztály, azaz mielőtt használnánk, újra kell fordítani őket. Továbbá a Tomcat servlettároló az újrafordított osztályokat nem tölti be önműködően. A legbiztosabb, ha minden babosztály újrafordítása után újraindítjuk a Tomcatet is.

1. lista SimpleBean.java

```
package il.co.lerner;

import java.util.*;
import java.lang.*;
import java.sql.*;

public class SimpleBean
             implements java.io.Serializable {

     private int userID;

     // ----------------------------------------
     // Constructor

     public SimpleBean () {
          // Beállítjuk a mezőnket
          userID = 0;
     }

     // ----------------------------------------
     // userID tulajdonság lekérdezése

     public int getUserID()
     {
          return userID;
     }

     // ----------------------------------------
     // userID tulajdonság megváltoztatása

     public void setUserID (int newID)
     {
          userID = newID;
          return;
     }
}
```

A bab használata

Most, hogy egyszerű babosztályunkat már elkészítettük, lássuk, miképpen használhatnánk a JSP-ből! A JSP három különleges, egyedi tagot ismer fel, amelyek mindig „jsp:" kezdetűek. Mielőtt folytatnánk, egy figyelmeztetés: ezek az egyedi tagok, melyek lehetővé teszik, hogy a JSP-ből JavaBeanekkel dolgozzunk, nem HTML-ben, hanem XML-ben íródtak! Míg a HTML lazán meghatározott szabvány, ahol nem mindig kötelező lezárni a tagokat, az XML sokkal szigorúbb. Minden nyitó `<tag>`-ot le kell zárni egy megfelelő `</tag>`-gal. Lezáró `</tag>` nélkül az XML értelmező-hibával kilép. A tag azonban saját magát is lezárhatja, ha a végére perjelet helyezünk a következők szerint: `<tag/>`.
Ez azt jelenti, hogy a JavaBeanst használó a JSP-ben – amely HTML-kimenetet készít – két kissé eltérő írásmódra kell figyelni. Egy idő után a két formai követelmény közti váltás teljesen természetessé válik. Ráadásul a JSP értelmező hibaüzeneteiből viszonylag könnyű meghatározni, hol felejtettük el kitenni a bezáró perjelet valamelyik JavaBean-tagra. Ennek ellenére az írásmódok ilyen keverése a kezdők számára elég őrjítő lehet, és elsajátításuk során jópár nehézségre van kilátásuk.
A JavaBean-osztályokhoz különleges, `<jsp:useBean/>` formátumú tagokat használunk. Ez a tag arról tájékoztatja a JSP-t, hogy keressen meg és töltsön be egy adott babot, majd a JSP-nkben készítsen

2. lista use-simple.jsp

```
<HTML> <Head><Title>SimpleBean
                    használata</Title></Head>
<jsp:useBean id="simple"
             class="il.co.lerner.SimpleBean"/>
<Body> <H1>SimpleBean használata</H1>
<P>Eredeti érték: <jsp:getProperty name="simple"
                   property="userID"/></P>
<jsp:setProperty name="simple" property="userID"
                 value="300"/>
<P>Új érték:<jsp:getProperty name="simple"
property="userID"/></P>
</Body> </HTML>
```

belőle egy példányt. A `<jsp:useBean/>` tag arra is lehetőséget nyújt, hogy a példánynak nevet adjunk, amit a későbbiek során felhasználhatunk. Lássuk például, miképpen tölthetnénk be imént készített SimpleBean-osztályunkat a JSP-ből:

```
<jsp:useBean id="simple"
             class="il.co.lerner.SimpleBean"/>
```

Amint látható, a tag perjellel (/) végződik – követve az XML írásmódját. Előfordulhat olyan eset is, amikor jobb szétválasztani a `<jsp:useBean/>` tagot nyitó `<jsp:useBean>` és záró `</jsp:useBean>` részre; ugyanis ami ilyenkor található a két tag között, csakis akkor hajtódik végre, amikor a bab először töltődik a memóriába. Természetesen az egyszerűsített forma is igen gyakori.
A `<jsp:useBean/>` tag két kötelező értékkel bír. A `class` érték azt a csomagot és osztályt nevezi meg, ahol a babunk található. Az `id` érték a JSP-n belül egyedi nevet rendel a babhoz. Akárcsak a változónevek esetében, nem árt, ha a babokkal való munkához egyértelmű azonosítókat választunk. Minél inkább magától értetődő a név, annál könnyebb lesz később nyomon követni a JSP-t.
A babpéldányunkból adatokat elővarázsolni és beállítani a `<jsp:setProperty/>` és `<jsp:getProperty/>` tagokkal tudunk. Mindkét tag egy `name` értéket vár, amelynek azonosnak kell lennie a korábban beállított `id`-vel (biztos vagyok benne, hogy megvan az oka, amiért a `<jsp:useBean/>`-ben `id` tulajdonságot használunk, míg a `<jsp:setProperty/>` tagban `name` tulajdonságot, de azért mindenképpen zavarónak tartom). A következő taggal tehát visszakaphatjuk az `userID` tulajdonság értékét:

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

Az előbbi sor egyszerű babunkból letölti a `userID` tulajdonságot, és a JSP-be helyezi. Érdemes megjegyezni, hogy nemcsak egyszerűen letölti az értéket, hanem egyszersmind láthatóvá is teszi a felhasználó számára. Azt is érdemes megfigyelni, hogy tulajdonságnevünk kis- és nagybetűi kicsit megváltoztak. A babunkban található `getUserID` tagfüggvény eléréséhez, azaz a `userID` tulajdonsághoz a `<jsp:getProperty/>` tagot kell használnunk. Ez senkit ne tévesszen meg, a tulajdonságnevek nem kis- és nagybetűfüggetlenek; az átalakítás egyszerűen az olvashatóság kedvéért történik.
A tulajdonság értékének megváltoztatásához a `<jsp:setProperty/>` tagot használjuk. Ez a tag semmit sem ad vissza, elfogad viszont egy `value` tulajdonságot, amelynek értéke aztán a babosztály megfelelő tagfüggvényéhez kerül:

4. lista calculator.jsp

```
<HTML>
<Head><Title>Calculator</Title></Head>

<jsp:useBean id="calculator"
 class="il.co.lerner.Calculate"/>

<Body>
<H1>Egyszerű számológép</H1>

<jsp:setProperty name="calculator"
property="*"/>

<P>arg1 legyen <jsp:getProperty
   name="calculator" property="arg1"/>.</P>
<P>arg2 legyen <jsp:getProperty
   name="calculator" property="arg2"/>.</P>

<P>Összeg: <jsp:getProperty name="calculator"
            property="sum"/></P>

<P>Hányados:
     <% try { %>
          <jsp:getProperty name="calculator"
           property="quotient"/>
     <% } catch (Exception e) { %>
          <B>Hiba! Osztás nullával </B>
     <% } %>
</P>

<P>Különbség: <jsp:getProperty
              name="calculator"
              property="difference"/></P>

</Body>
</HTML>
```

```
<jsp:setProperty name="simple" property="userID"
                 value="300"/>
```

A *2. lista* egy teljes a JSP-t tartalmaz, amely bemutatja, miként használhatjuk JSP-ből a SimpleBean-osztályt. Először megjeleníti a `userID` tulajdonság alapértelmezett értékét, majd új értéket rendel a tulajdonsághoz, végül ezt az újat szintén megjeleníti.

Értékek és tulajdonságok

Nyilvánvaló, hogy általános gyakorlat a tulajdonságokat a bab példányváltozóiban tárolni, ahogyan azt mi is tettük a SimpleBean-osztály esetében. Ilyenkor a `<jsp:setProperty/>` meghívása tulajdonképpen a mező értékét állítja be, míg a `<jsp:getProperty/>` a pillanatnyi értékét adja vissza. Természetesen a tulajdonságoknak nem kötelező mezőkhöz kötődniük, ugyanígy tárolhatók és lekérdezhetők egy relációs adatbázis-kezelőből. Amikor egy tulajdonságot lekérdezünk, az is megoldható, hogy a visszatérési értéket valós időben számítsuk ki, ahelyett hogy egyszerűen csak kiolvasnánk egy példányváltozóból. Képzeljük el, például milyen könnyedén készíthetnénk egyszerű matematikai műveleteket végző babot! Beállíthatnánk két írható, illetve olvasható tulajdonságot (nevezzük őket arg1-nek és arg2-nek), majd létrehozhatnánk számos csak olvasható tulajdonságot, amelyek a korábban megadott paraméterekből számított értékeket adnák vissza. A 3. lista (mely az ftp://ftp.ssc.com/pub/lj/listings/issue86 címen is elérhető) egy ilyen, a fentieket bemutató egyszerű babot tartalmaz Calculate.java néven.

Mivel `setSum` tulajdonság nem létezik, a JSP-motor a `sum` tulajdonságra nem fogja engedélyezni a `<jsp:setProperty/>` meghívását. Ugyanakkor engedélyezi az arg1 és arg2 beállítását, illetve azt is, hogy bármelyik tulajdonságot lekérdezzük.

Most hogy van egy működő babunk, akár fel is használhatjuk a JSP alól. A calculator.jsp forrását a *4. lista* tartalmazza, ahol néhány alapműveletet végzünk az imént készített JavaBeannel.

A calculator.jsp első és legérdekesebb része az a mód, ahogy a tulajdonságokat beállítja:

```
<jsp:setProperty name="calculator" property="*"/>
```

Megszokott helyzetben fognánk az egyik GET vagy POST tagfüggvénnyel kapott értéket, és a következő jelölésmóddal valamelyik tulajdonsághoz rendelnénk:

```
<jsp:setProperty name="calculator"
                 parameter="foo" property="arg1"/>
```

Más szóval: a fenti tag fogadja a `foo` értéket és ezzel meghívja a calculator-bab `setArg1` eljárását. Amikor viszont csillagot használunk, ahogyan a 4. listában is, akkor azt jelezzük a JSP-motornak, hogy az összes megkapott értékre szükségünk van, és mindegyiket ugyanolyan nevű tulajdonsághoz szeretnénk rendelni. Így az `arg1` érték az arg1 tulajdonságba kerül és így tovább.

Az én rendszeremen, ahol a calculator.jsp-t az examples/jsp címre helyeztem, a következő URL segítségével rendelhetem az `arg1`-hez az ötös értéket és az `arg2`-höz a húszas értéket: „http://localhost/examples/jsp/calculator.jsp?arg1=5&arg2=20”. Természetesen ez nem éppen „bolondbiztos” rendszer. Könnyen futásidejű hibát idézhetek elő, például a következő URL átadásával: „http://localhost/examples/jsp/calculator.jsp?arg1=5&arg2=20.0”. A JSP-motor megpróbálja hozzárendelni a 20,0 (ez egy lebegőpontos szám) értéket az `arg2` paraméterhez (amely egész szám), és kudarcot vall.

Ha el szeretnénk kerülni a futásidejű hibákat, például közrezárhatjuk a `<jsp:setProperty/>` és `<jsp:getProperty/>` tagjainkat scriptlet-tagokkal, kihasználva a hagyományos Java try-and-catch eljárást. A következő kód például szavatolja, hogy soha nem kell foglalkoznunk a nullával osztás kivétellel, amikor a `getQuotient`-et használjuk:

```
<P>Négyzetgyök:
     <% try { %>
          <jsp:getProperty name="calculator"
          property="quotient"/>
     <% } catch (Exception e) { %>
          <B>Hiba! osztás nullával</B>
     <% } %>
</P>
```

Ugyanakkor ez a kódolási mód Javát visz a JSP-be, és pontosan ezt szerettük volna elkerülni, ezért is kezdtünk a babokkal foglalkozni. Az, hogy nem programozó társaink képesek-e kezelni az effajta kódot, nagymértékben függ a környezetünktől, illetve attól hogy milyen típusú babokat készítettünk.

A babok hatóköre (scope)

Amint az a múlt hónapban kiderült, a servlettároló egy időben minden servletről csak egyetlen másolatot tölt a memóriába. Ez azonban

6. lista viewblog.jsp

```
<HTML>
<Head><Title>Web-napló</Title></Head>

<jsp:useBean id="blog"
class="il.co.lerner.Weblog"/>

<Body>
<H1>Aktuális Web-napló tartalom</H1>

<table>
<jsp:getProperty name="blog" property="blog"/>
</table>

</Body>
</HTML>
```

nem okoz gondot, mivel a Java többszálú, és egy adott servletnek lehetősége van egyszerre több HTTP-kérelemmel is foglalkozni. Mivel a JSP-k tulajdonképpen álruhás servletek, ugyanígy rájuk is vonatkozik a többszálas futtathatóság.
Felmerül a kérdés, mi történik JavaBeanjeinkkel: hányszor töltődnek be, mekkora a hatókörük (scope). Ha a JSP egy olyan tulajdonságot állít be, amely az osztály mezőit módosítaná, hatással van-e ez ugyanennek a babnak a többi példányára is?
A válasz: attól függ. A babok elérési területeinek négy fajtáját különböztethetjük meg, és a választott elérés típusa alapjaiban meghatározza a felhasznált bab viselkedését. Az alkalmazásszintű hatókör (Application scope) egyetlen másolatot jelent a babból minden, a servlettárolóban futó JSP-hez. A folyamatszintű (Session scope) a felhasználóhoz rendelt – attól a pillanattól kezdve, hogy belépett a honlapra, addig a percig, amíg ki nem lép onnan. Ha a felhasználó két böngészőablakot nyit a rendszerre, azok azonos folyamatnak minősülnek. A kérelemelérési szint (Request scope) a HTTP-kérelem végéig tart. Ez olyankor hasznos, ha az egyik JSP-ben szeretnénk beállítani a bab tulajdonságát, majd a `<jsp:forward/>` taggal belső átirányítást hajtunk végre egy másik JSP-re, s ebben a második JSP-ben szeretnénk a babot felhasználni. Végül a lapelérési szint (Page scope) egyetlen JSP-lapra vonatkozik. Amikor a lap feldolgozásra kerül, az elérhetőség is megszűnik.
Alapértelmezés szerint a babok a folyamatszintű hatókörrel futnak. Ennek megváltoztatásához a scope-értéket kell a `<jsp:useBean/>`-be helyezni:

```
<jsp:useBean id="simple" scope="application"
             class="il.co.lerner.SimpleBean"/>
```

Ha egyszer végrehajtottuk a fentieket, az egyik JSP beállította értékek az összes többiben is elérhetők lesznek. Természetesen – mivel az alkalmazás- és kiszolgálószintet egy időben több JSP is elérheti – szálbiztosnak kell lennie.
Gondoljunk rá, milyen elérhetőségű lesz a babunk és ha szükséges, tegyük szálbiztossá. Ha a bab nem szálbiztos, mindenképpen jelezzük ezt a leírásban, nehogy mások véletlenül hibásan használják fel.

Webnapló

Az elmúlt hónapban folytattuk egyszerű vizsgálódásunkat a JSP-vezérelt webnaplók terén, melyek közvetlenül értek el relációs adatbázisokat a legutóbbi webnaplóbejegyzés lekéréséhez. Bár az ilyen JSP természetesen jó, de meglehetősen nehézkes, bonyolult a nyomon követése, és a kódtól a logikát nem választja szét olyan választékosan, mint ahogyan azt elvárhatnánk.
Ha az adatbázis-logikát JavaBeanbe helyezzük, célkitűzéseink közül elérhetünk néhányat: a kód újrafelhasználható, akár a nem programozók számára is elérhető lesz, és lehetővé teszi, hogy anélkül változtassuk meg az adatforrást vagy a belső logikát, hogy ehhez újra kelljen írnunk a JSP-t. Az 5. lista (elérhető a ➲ ftp://ftp.ssc.com/pub/lj/listings/issue86 címen) tartalmazza babunk forráskódját, amelyet szálbiztossá is tettem (a `synchronized` kulcsszó felhasználásával a `getBlog` tagfüggvényben), így alkalmazásszinten elegendő egyetlen példányt készítenünk. Ez a JavaBean hozzákapcsolódik a webnapló-adatbázishoz és lekérdezi a legfrissebb adatokat. A *6. lista* egy kis JSP-t tartalmaz, amely JavaBean segítségével jeleníti meg a webnaplót. Tulajdonképpen semmi különleges nincs az 5. listában, viszont számos olyan dolgot egyesít magában, amelyeket az elmúlt pár hónapban megvitattunk. Immár rendelkezünk egy olyan módszerrel, amellyel mindenki képes webnaplónk adataihoz hozzáférni anélkül, hogy egyetlen sor kódot is kellene írnia! Néhány egyszerű JSP-taggal a jelenlegi blog-tartalmat könnyen és egyszerűen behelyezhetjük a HTML-lapba.

Összegzés

A JavaBean-babok nagyszerűek: a JSP-be kerülő kódmennyiség csökkentése végett helyezzük a lapba, mellyel egyúttal a használatát is egyszerűsítjük. Ugyanakkor egy összetett JSP még mindig tartalmaz valamennyi kódot: ha például a ciklusokban kell ismételni valamit, vagy összetett adatokkal kell dolgozni.
A következő hónapban megtudjuk, miként írhatunk saját – az e havihoz hasonló – jsp: tagokat a JSP testreszabott eljáráskezelő képességének kihasználásával. Így elkészíthetjük a saját tagjainkat, melyekhez olyan kódot rendelhetünk, amilyet csak akarunk. A JSP lehetővé teszi, hogy felépítsük saját formátumnyelvünket és emellett az eddig elérhető tagokat is felhasználhassuk.

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 Software Foundation Jakarta-Tomcat Java servletekkel és JSP-kel foglalkozó projektje a ➲ http://jakarta.apache.org/ címen érhető el. Erről a címről letölthetjük a forráskódot, a binárisokat, a leírást, valamint a telepítéshez szükséges utasításokat, illetve a teljesítmény növelését segítő ötleteket.
➲ http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/mod_jk-howto.html

Hans Bergsten, az O'Reilly and Associates által közelmúltban kiadott JavaServer Pages igen kitűnő forrásmunka azok számára, akik JSP-oldalakat szeretnének készíteni.
➲ http://www.oreilly.com