Entitásbabok

Megtanuljuk, hogyan írjunk entitásbabokat, miképpen csatlakoztassuk őket adatbázishoz, illetve hogyan érjük el őket unokatestvéreiken, a sessionbabokon keresztül.

Az elmúlt hónapban bepillantottunk a Sun kiszolgálóoldali webalkalmazásokhoz szánt szabványának (J2EE – Java 2 Enterprise Edition) legfontosabb alkalmazásába, az Enterprise JavaBeans (EJB) világába. Bár sem a Java nyelv, sem a J2EE-szabvány nem nyílt, mégis egyre növekvő számban akadnak Linux-barátok, akik ezek segítségével készítenek kiszolgálóoldali webalkalmazásokat. Az igazság az, hogy úgy tűnik, a J2EE lesz az egyetlen használható lehetőség Microsoft's .NET keretrendszerrel szemben, ami sokak szemében minden bizonnyal még kívánatosabbá teszi.
Az Enterprise JavaBeans két alapvetően fontos – sessionbabnak és entitásbabnak nevezett – elemet tartalmaz. A sessionbabok többnyire folyamatokat modelleznek és nincs állapotuk (lack any state), és lehetővé teszik, hogy az üzleti logikát a JavaServer Pages (azaz JSP) helyett az EJB-be vigyük be. A múlt hónapban tervezett számológép, amely két számot tudott összeszorozni, az egy eljárással rendelkező sessionbabok nagyon egyszerű példája volt. Az entitásbabok ezzel szemben állapottal rendelkeznek, néha akár meglehetősen összetett állapottal. Ez az állapot többnyire egy relációs adatbázis-kezelő táblázatának egyik sorát jelenti, ahol vagy a bab, vagy a saját beépített objektumrelációs átalakító megoldását használjuk (bean-managed persistence, azaz BMP), vagy pedig az EJB-re hagyjuk ennek a feladatnak a megoldását (container-managed persistence – CMP). Az EJB-tároló tranzakciókat is támogat, lehetőséget adva objektumainkon, illetve a nekik megfelelő adatbázison a mindent vagy semmit típusú műveletekre.
Ebben a hónapban egyszerű entitásbabot fogunk írni, adatbázishoz csatlakoztatjuk, majd Java-alkalmazásból sessionbabokon keresztül megpróbáljuk elérni. Ehhez a nyílt forráskódú JBoss EJB-tárolót fogjuk felhasználni (ez a GNU Lesser General Public License, azaz LGPL szabadalom alá tartozik), de kódunk kis módosításokkal bármely EJB-t támogató J2EE-kiszolgálón futni fog.

Készítsünk entitásbabokat!

Amint a múlt hónapban is láthattuk, a sessionbab írása tulajdonképpen három Java-osztály létrehozását jelenti:

- a tényleges feladatot elvégző babosztályét,
- a távoli csatolófelületét, amely a babosztály eljárásainak megfelelő tagfüggvényeket tartalmaz, és tulajdonképpen átjárót (proxy) képez a babunkhoz,
- annak a saját csatolófelületnek létrehozását, amelynek segítségével a babból új példányokat hozhatunk létre, illetve különféle feltételeknek megfelelő babokat kereshetünk ki.

Az entitásbabok esetében ugyanígy mindhárom osztályt létre kell hoznunk. Ezenkívül azonban gyakran egy negyedik, úgynevezett „elsődleges kulcs" osztályt is el kell készítenünk. Bár e havi példánkban nem lesz ilyen elsődlegeskulcs-osztályra szükségünk, az érdekesség és a teljesség kedvéért létre fogjuk hozni.
A legtöbb EJB-alkalmazás legalább egy entitásbabot (az adatmodellezéshez) és legalább egy sessionbabot tartalmaz (az üzleti logika megvalósításához). Minthogy az objektumközpontú programozás alapötlete épp az, hogy az adatokat és a kódot egyetlen csomagban kezeli, az entitás- és a sessionbabok ilyen módon történő szétválasztása kicsit furcsának tűnhet. Ennek ellenére ez a módszer mégis működőképes, és ha egyszer megállapodtunk az alapelvekben, meglehetősen leegyszerűsíti a munka megosztását.

Munka a JAWS-zal

A J2EE egy szabvány, melynek tényleges megvalósítása attól függ, hogy ténylegesen ki írta a kiszolgálót. A J2EE alkalmazáskiszolgáló leglényegesebb részeinek egyike az objektum-relációs átalakító, amely a Java-osztályokat átlátszó módon alakítja át relációs adatbázistábla-sorokká (és fordítva). Ennek az objektumrelációs átalakítónak a lehető legrejtettebbnek kell maradnia, hiszen csak így lehetséges, hogy a háttértárat például Oracle-ről a Java-kód megváltoztatása nélkül egyszerűen MySQL-re váltsuk.
A JBoss objektumrelációs átalakítórendszer (JAWS) általában alig igényel beállítást. Ennek ellenére hasznos lehet a JAWS beállításfájlját végignézni (***standardjaws.xml*** a JBoss ***conf/default*** könyvtárban), hogy tisztában legyünk vele, tulajdonképpen mi zajlik a háttérben.
A ***standardjaws.xml*** elején található meghatározások a teljes JBoss-kiszolgálóra vonatkozó értékeket állítanak be. Itt adhatjuk meg, hogy melyik adatbázis-kezelőt szeretnénk használni. A HyperSonic adatbázis-kezelő a JBosszal együtt érkezik, és a következő példákban mi is ezt fogjuk használni.
A ***standardjaws.xml*** magját többszörös `<type-mapping>` (egytagú) részek alkotják, amelyek az adatbázisok mindegyike esetében az összes `<java-type>`-elemet egy `<jdbc-type>`- és `<sql-type>`-elemhez kapcsolják. Mivel EJB-nk nem készít táblákat és közvetlenül SQL-t sem ír, nagyon fontos, hogy ezeket az értékeket pontosan állítsuk be. Beállításukkal növelhetjük a programunk hatékonyságát és rugalmasságát.
Ne feledjük azonban, ha a JAWS-t azután módosítjuk, miután már adatokat tettünk az adatbázisba, az könnyen zavarhoz, károsodáshoz vagy adatvesztéshez vezethet.
Ha az EJB-t egyszerűen csak ki akarjuk próbálni, akkor egyáltalán nem kell a ***standardjaws.xml***-t módosítanunk. Inkább írjuk át a ***jboss.jcml***-t, ugyanis ez az XML-fájl adja meg azokat a kezelőbabokat (managed beans avagy MBeans), amelyeket a JBoss a rendszer beállítására és irányítására használ.
A ***jboss.jcml*** fájl HyperSonic- és InstantDB-támogatást is tartalmaz. Ahhoz viszont, hogy HyperSonickal is működjön,

előbb az összes InstantDB-vel kapcsolatos bejegyzést el kellett távolítanom a ***jboss.jcml***-ből. Ezt a ***jboss.jcml*** „JDBC” szakaszának módosításával tehetjük meg: töröljük a `JdbcProvider` Mbeanhez tartozó ***Drivers*** részből az `org.enhydra.instantdb.jdbc.idbDriver` bejegyzést, majd a teljes `XADataSourceLoader <mbean>` szakaszt, illetve minden `XADataSource` típusú vagy InstantDB nevű szolgáltatást.
Ha a ***jboss.jcml***-ből már minden InstantDB-vel kapcsolatos sort töröltünk, indítsuk el a JBosst:

```
cd $JBOSS_DIST/bin
sh run.sh
```

Entitásbabunk megírása

Soron következő entitásbabunk egy egyszerű könyvesboltot fog modellezni, ahol minden egyes könyv címmel, szerzővel, kiadóval és árral rendelkezik. Az egyszerűség és a tömörség kedvéért most eltekintünk attól, hogy egy könyvnek több szerzője és kiadója is lehet. Továbbá az adatokat normalizálni sem fogjuk, ami egyébként azt jelentené, hogy olyan példányváltozóink lennének, amelyek maguk is entitásbabok.
A `BookBean`-osztály megírásával kezdünk, amit az *1. listá*-ban is olvashatunk, illetve a 28. CD Magazin/Beans könyvtárában is meglelhetünk. A BookBean jó példa az egyszerű tárolókezelésű babosztály meghatározására; minden egyes nyomon követendő adatbázisoszlophoz megad egy-egy mezőt, ideértve az `id` egész típusú mezőt is, amely elsődleges kulcsként szolgál.
Meg kell adnunk egy `ejbCreate()` tagfüggvényt, ami a saját csatolófelület `create()` tagfüggvényének felel meg. Valahányszor csak meghívjuk a saját csatolófelületünk `create()` tagfüggvényét, az EJB-tároló az `ejbCreate()`-et adott értékekkel meghívja a babosztályunkra. Tulajdonképpen az `ejbCreate()`-ben megy végbe a valódi létrehozatal; a CMP entitásbabnak egyáltalán nem kell az objektumrelációs átalakítással törődnie, viszont megfelelő értékűre kell beállítania a példányváltozókat.
Az ejbCreate()-en kívül már csak a „leszedő” és „feltöltő” tagfüggvényeket kell minden egyes mezőhöz létrehoznunk, hogy objektumaink a mezők tartalmát le tudják kérdezni, illetve módosítani tudják. Példánkban természetesen minden tagfüggvény meglehetősen egyszerű, mindössze módosítja vagy visszaadja egy példányváltozó értékét.
A távoli csatolófelület (amit a 2. *listában* találunk meg) neve *Book.java*, és felülete csaknem teljesen megegyezik a babosztályával. Az alkalmazások többnyire a távoli csatolófelülettel beszélgetnek, és ha valami gond merül fel, egy `RemoteException`-t dobnak.
Meg kell adnunk egy `create()` tagfüggvényt tartalmazó saját csatolófelületet is (ezt a 3. *listában* találjuk), amely a `Book` objektumpéldányok létrehozására szolgál (illetve közvetett módon egyúttal adatbázistáblánkban is új sorokat hoz létre), ha a könyv összes adatát átadjuk neki. Amennyiben elég lelkesek lennénk, többfajta `create()` függvényt is felajánlhatnunk, egyet-egyet minden értékszámhoz.
Figyeljük meg, hogy `create()` tagfüggvényünknek az elsődleges kulcsot közvetlenül kell megadnunk. A tapasztalt adatbázis-programozók tudják, hogy az elsődleges kulcsokat lehetőség szerint nem szabad szem előtt hagyni, és ennek önműködővé tételére a legtöbb adatbázis-kezelő lehetőséget is ad; a PostgreSQL ***SERIAL***-típusa, a MySQL ***AUTO INCREMENT***, és az Oracle-szekvenciák a szokásos megoldások erre a feladatra. Sajnos nincs rá igazán egyszerű lehetőség, hogy ezeket az önműködően előállított elsődleges kulcsokat EJB-ből kihasználjuk. Így aztán vagy közvetlenül meg kell adnunk őket (ahogyan az e havi példában is tettük), vagy egy külső értéket kell használnunk, például az ISBN-számot, amely karaktersorozat is lehet. E szolgáltatás hiánya döbbentett meg engem a leginkább az EJB-ben, és remélem, a szabvány következő változatai már orvosolni fogják a hibát.
A `findBy...()` tagfüggvény segítségével megkereshetjük és lekérhetjük a `Book` egyes példányait.
A `findByPrimaryKey(5)` például azt a `Book`-objektumot fogja visszaadni, amelynek elsődlegeskulcs-értéke 5. Az összes `findBy...()` tagfüggvény az EJB-tárolóban már meg van határozva, így nekünk már nem kell megtennünk.
A `findAll()` tagfüggvény az összes ilyen típusú objektum egy csoportját adja vissza (azaz az adatbázistábla összes sorát), így végiglépkedhetünk rajtuk.
A beépített `findAll...()` tagfüggvény sajnálatos módon

2. lista Távoli csatolófelületünk, a Book.java forráskódja – egyszerűen csak tükrözi a BookBean-osztályunkban meghatározott tagfüggvényeket

```
package il.co.lerner.book;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

/* A Book entitÆsbab tÆvoli csatol fel lete.
   A BookBean minden tagf ggvØnyØhez
   ➥megadunk egy megegyezı jelzØsß
   ➥tagf ggvØnyt . */

public interface Book extends EJBObject
{
   public int getId() throws RemoteException;
   public void setId(int newId) throws
      ➥RemoteException;

   public String getTitle() throws
   ➥RemoteException;
   public void setTitle(String newTitle)
      ➥throws RemoteException;

   public String getAuthor() throws
      ➥RemoteException;
   public void setAuthor(String newAuthor)
      ➥throws RemoteException;

   public String getPublisher() throws
      ➥RemoteException;
   public void setPublisher(String
   ➥newPublisher)
      ➥throws RemoteException;

   public double getUsDollarPrice()
      ➥throws RemoteException;
   public void setUsDollarPrice
   ➥(double newDollarPrice)
      throws RemoteException;
}
```

3. lista Saját csatolófelület megadása

```
package il.co.lerner.book;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import java.util.Collection;

/* Ez az osztÆly hatÆrozza meg Book
/* entitÆsbabunk
➥sajÆt csatol fel letØt. */

public interface BookHome extends EJBHome
{
    /* j Book-pØldÆny kØsz tØse */
    public Book create(int newId, String
    ➥newTitle,
                  String newAuthor, String
                  newPublisher,
                  double newUsDollarPrice)
      throws RemoteException, CreateException;

    /* Keress k meg az adott azonos t jœ
    /* Book-pØldÆnyt. Az EJB-
    tÆrol  ezt a tagf ggvØnyt  nmßk dıen
    ➥hozza lØtre helyett nk. */

    public Book findByPrimaryKey (int id)
      throws RemoteException, FinderException;

    /* Egy k nyvgyßjtemØny visszaadÆsa,
    /* ahol
    a szerzı neve megegyezik. Ezt a
    ➥tagf ggvØnyt
az EJB-tÆrol   nmßk dıen hozza lØtre. */

    public Collection findByAuthor
    ➥(String authorName)
      throws RemoteException,
      ➥FinderException;

    /* Az adatbÆzis  sszes
    /* Book-objektumÆnak visszaadÆsa */
    public Collection findAll()
        throws RemoteException,
FinderException;
}
```

egyszerű egyenlőségvizsgálatot végez. Nem tudunk például szabványos kifejezések szerint keresni vagy más módszereket alkalmazni, például kikeresni az összes O betűvel kezdődő könyvcímet, vagy azokat a könyveket, amelyeknél a kiadó neve A betűvel kezdődik és M-re végződik. Ehelyett a teljes készletet kénytelenek vagyunk a `findAll()` használatával végigpörgetni, és a szükséges szűréseket magunk elvégezni (azaz pontosan azt veszítjük el, amiért érdemes adatbázis-kezelőt használni ezzel az erővel akár egy fájlban is lehetne – a ford.).
Végül a *4. listában* található elsődlegeskulcs-osztályunk (***BookPK.java***) egyetlen példányváltozót vezetett be (`id`),

4. lista BookPK.java, az entitásbabunkhoz tartozó elsődlegeskulcs-osztály

```
package il.co.lerner.book;

import java.io.Serializable;

public class BookPK implements
java.io.Serializable
{
    public int id;
    public BookPK () { }

    public BookPK(int value)
    {
        id = value;
    }

    public boolean equals(Object obj)
    {
        if (obj == null ||
        ➥!(obj instanceof BookPK))
            return false;

        if (((BookPK)obj).id == id)
            return true;

        return false;
    }

    public int hashCode ()
    {
        return id;
    }


    public String toString()
    {
        return String.valueOf(id);
    }
}
```

amelyet elsődleges kulcsként használunk. Az `equals()` tagfüggvény azt mutatja meg, hogy két `BookPK`-példány azonosnak tekinthető-e, ezáltal a rendszer képes lesz összehasonlítani két `Book`-példányt. A `hashCode()` tagfüggvénynek minden példányhoz egyedi azonosítót kell visszaadnia, amelyet aztán az adott esetben kulcsként használhatunk fel.
A `toString()` tagfüggvény feladata, hogy az elődleges kulcs karakteres értékét, azaz egyszerűen az osztályunk `String.valueOf(id)` értékét adja vissza.
Minthogy mind a négy fent említett osztály az ***il.co.lerner.book*** csomagban található, mind a négy forrásfájlt (***Book.java***, ***BookBean.java***, ***BookHome.java*** és ***BookPK.java***) a ***$BOOK/il/co/lerner/book*** könyvtárba helyeztem, ahol a BOOK projekt a saját könyvtárat jelenti.

Telepítésleíró

Most, hogy az entitásbabot elkészítettük, itt az ideje, hogy az EJB-tároló számára le is írjuk. Telepítésleírónk egy ***ejb-jar.xml*** nevű fájl – ez látható az *5. listában*. Az ***ejb-jar.xml***-t a

5. lista Az ejb-jar.xml – az entitásbabunkhoz tartozó telepítésleíró

```
<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar>
  <description>ATF Book Bean</description>
  <display-name>ATF Book</display-name>
  <enterprise-beans>
     <entity>
        <description>Models a book</description>
        <ejb-name>Book</ejb-name>
        <home>il.co.lerner.book.BookHome</home>
        <remote>il.co.lerner.book.Book</remote>
        <ejb-class>il.co.lerner.book.BookBean
             </ejb-class>

        <persistence-type>Container</persistence-type>
        <reentrant>False</reentrant>

        <prim-key-class>il.co.lerner.book.BookPK
             </prim-key-class>

        <cmp-field><field-name>id</field-name>
             </cmp-field>
        <cmp-field><field-name>author</field-name>
             </cmp-field>
        <cmp-field><field-name>title</field-name>
             </cmp-field>
        <cmp-field><field-name>publisher</field-name>
             </cmp-field>
        <cmp-field><field-name>usDollarPrice
             </field-name></cmp-field>
     </entity>
  </enterprise-beans>
</ejb-jar>
```

$BOOK/il/co/lerner/book/ könyvtárba helyezzük, azok mellé a Java-osztályok mellé, amelyek majd az entitásbabunkat fogják alkotni. Amikor a babot az Anttal felépítjük, a ***META-INF*** alkönyvtárba fog kerülni.

Az ***ejb-jar.xml*** entitásbabokra vonatkozó legérdekesebb része a `<persistence-type>` szakasz (CMP esetén ***Container***-re kell állítani), a `<prim-key-field>`- és `<prim-key-class>` szakaszok (ahol elsődleges kulcsaink osztályát nevezhetjük meg), és a `<cmp-field>` szakasz (amely a tárolóvezérelt mezőket írja le a JBossnak).

A telepítésleíró az EJB szabványos része, és minden EJB-kiszolgálóval és- tárolóval működnie kell. Nem adja meg azonban az összes futásidejű beállítási lehetőséget, ezért aztán a JBoss helyes működéséhez egy ***jboss.xml*** nevű fájlt is be kell fűznünk, amely a kiszolgálónak megmondja, hogy a babokat a hálózaton hol találjuk meg. A ***jboss.xml*** másolata, amit az ***ejb-jar.xml*** mellé a ***$BOOK/il/co/lerner/book/***-ba kell helyeznünk, mely folyamat az alábbi listán látható:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>Book</ejb-name>
      <jndi-name>Book</jndi-name>
    </entity>
  </enterprise-beans>
</jboss>
```

Alkalmazás

Egyszerű próbaalkalmazásunk (***UseBook.java***) forráskódja a 6. listában látható a 28. CD Magazin/Beans könyvtárában. Szépen bizonyítja, hogy mégoly kevés kóddal is milyen sokat el lehet érni. Csak a `main()` tagfüggvényt határozza meg, és máris az EJB-kel kezd dolgozni. Először elfogja a JNDI-környezetet és felkeresi az általunk megadott `Book` babot. Ezáltal lehetővé válik egy `BookHome`-példány létrehozása, s következő lépésben már új `Book` babot hozhatunk létre:

```
Book book =
home.create(testPrimaryKey,
   ➥"Book title",
                    ➥"AuthorFirst
AuthorLast",

➥"PublisherName", 10.50);*
```

Amint láthatjuk, az adatbázishoz adandó értékeket beégettük a kódba, az elsődleges kulcsot is beleértve. Ez egy valós alkalmazás esetében nyilvánvalóan elfogadhatatlan lenne. Egy valódi alkalmazás az elsődleges kulcsot (és más értékeket) egy fájlból, a parancssorból, a környezeti változóból, egy webes HTML-űrlapból venné, vagy éppen önműködően készítene egyet.

Figyeljük meg, hogy egyszer sem kellett SQL-lekérdezésket vagy adatbázis-lekérdezéseket beszúrnunk. Háttértárolónk feltételezhetően relációs adatbázis, de Java ügyfélprogramunk erről nem tud, és nem is számít neki.

Miután beszúrt egy új sort a táblába, az UseBook megváltoztatja néhány értékét (a példányváltozók értékeinek az átírásával), majd az adatbázisban található összes `Book`-példányt lekérdezi (a `findAll()` segítségével). A teljes futás alatt állapotüzeneteket ír a ***System.out***-ra, amelyek a konzolra íródnak.

Az Ant beállításfájl

Programunk fordításához és telepítéséhez az Antot használjuk, a hagyományos make program Java-megfelelőjét.

A 7. lista (elérhető a 28. CD Mgazin/Beans könyvtárában) a felhasznált ***build.xml*** fájlunkat mutatja be, amely a ***$BOOK/il/co/lerner/book*** könyvtárban az összes ***.java*** forrásfájlt lefordítja, majd a telepítésleíróval és a JBoss futásidejű beállításfájljával egyetemben Java ***.jar*** fájllá alakítja őket, végül a fájlokat a JBOSS könyvtárba telepíti. Ha az Antot az ***ANT***-ba telepítettük, a következő paranccsal az összes fájlt lefordíthatjuk, majd a JBoss telepítési könyvtárába telepíthetjük, és elindíthatjuk az `UseBook.main()`-t:

```
$ANT/bin/ant use-book-ejb
```

Amint a JBoss felfigyel az új (vagy frissített) ***.jar*** fájlra, a kime-

netet a kiszolgáló tevékenységét mutató képernyőn már láthatjuk is. Az Ant kimenete ezzel szemben azokat az elemeket fogja mutatni, amelyeket a `main()`-ből küldtünk a `System.out`-ra – a 6. listában olvasható állapotüzenetek mutatják, mit is csináltunk. Valahányszor csak a `main()`-t új `ID` értékkel fordítjuk le és futtatjuk, mindig egy-egy új sor szúródik be az adatbázistáblába.

Az EJB a jövő?

A Sun azt szeretné, ha azt hinnénk, hogy az EJB minden kiszolgálóoldali alkalmazás jövője. A Microsoft természetesen mindent megtesz, hogy azt bizonyítsa be: a .NET a valódi jövő. Hogy igazodjék el egy magányos fejlesztő az óriások csatájában, és ugyan mit tehetnek a szabad programok hívei? A jó hír az, hogy a J2EE kitűnő szerkezettel és filozófiával rendelkezik. Nem éppen a legelegánsabb és nem is a legrugalmasabb, sőt, egyetlen nyelvhez láncol, általánosságban azonban jó benyomást tett rám. A J2EE láthatóan fontos mérföldkő a webalkalmazások programozása terén.

Sajnálatos módon számos olyan gond van a J2EE-ben, amelyekkel mindig szembekerülök, valahányszor csak megpróbálok benne alkalmazást írni. Az első, hogy sem a Java, sem a J2EE nem nyílt forrású, annak ellenére sem, hogy a Java ingyenes és a JBoss LGPL felhasználási szerződésű. A Sun általában becsületesen játszik, de mégiscsak a haszonelv alapján működő vállalat, amelynek megvannak a saját érdekei. A nyílt forrás híveit könnyen érheti az a meglepetés, hogy a Sun egyszercsak korlátozni kezdi a kód vagy a szabvány használatát.

Továbbá úgy tűnik, hogy az Enhydra Enterprise-részleg végleg eltűnik, és a JBoss marad az egyetlen nyílt forrású J2EE-alkalmazáskiszolgáló a piacon. A JBoss nem rendelkezik a hivatalos J2EE-alkalmas minősítéssel, igaz, inkább csak azért, mert a Jboss-csapat nem tudja a hivatalos minősítést megfizetni. Ez rámutat, milyen rövidlátó is a Sun; nyugodtan felajánlhatnának egy olcsó vagy ingyenes minősítést a GPL vagy LGPL felhasználási szerződésű kiszolgálóknak, hiszen (a Berkeley felhasználási szerződéssel szemben) a (L)GPL szavatolja, hogy a minősített kiszolgálót semmilyen cég sem alakíthatja kereskedelmi programmá. A hivatalos J2EE-minősítés nekem ugyan nem sokat számít, de számos döntéshozó számára fontos dolog, és ez azt jelenti, hogy a Jbosst gyakran érdemtelenül hagyják figyelmen kívül.

A Java és az EJB elég összetett, és bizony időbe telik, míg egy programozó megtanulja használni őket. De tapasztalataim szerint a Java- és az EJB-programozás eltörpül ama munkamennyiség mellett, amit a hihetetlen mennyiségű beállításfájl, az új meghatározások, a környezeti változók és más, csak a Javára jellemző dolgok megtanulása jelent. Egy jobb leírás nyilván segítene, ami azonban igazán hiányzik nekem, az a CPAN javás megfelelője, amely a kiszolgálóoldali Java-beállításokat könnyíthetné meg, és így lehetővé válna, hogy a programozók rendszerkarbantartási feladatok helyett inkább a programozásra összpontosítsanak.

A .NET egyetlen olyan jellemzője, amit tényleg érdekesnek találok, az viszonylag nagy nyitottsága a különféle programozási nyelvek irányában. Alapértelmezés szerint a J2EE mindenképpen Javát igényel, ami gyakran valóban a helyes választás, de nem kellene, hogy az egyetlen választás legyen. A SOAP és az XML-RPC lehetővé teheti a nyelvek közötti szakadék áthidalását – de a kellemes tranzakciókezelés és objektumrelációs modell nélkül, amit az EJB hozott meg számunkra. Jelenleg úgy tűnik, az egyetlen lehetőség a Python és az EJB közti párbeszédre a SOAP vagy XML-RPC (avagy Jython), de szívesen látnék újabb lehetőségeket is a közeljövőben.

Összefoglalás

Az EJB lenyűgöző módszer, és sokkal többre képes, az olyan egyszerű objektumrelációs átalakítóknál, mint az Alzabo vagy a DODS. Tapasztalataim szerint az EJB-vel való munka sokkal inkább szervezési és logisztikai feladat, semmint technikai. Az EJB elsajátítása mindenképpen jó ötlet minden webalkalmazás-fejlesztő számára; nyilvánvaló, hogy ez a szabvány jelentős mértékben terjedni fog az iparban, és számos komoly alkalmazást EJB-ben fognak írni a jövőben. A minősített nyílt forrású megoldások a programozóknak még könnyebbé tennék az EJB kipróbálását, és javasolnám a Sunnak, hogy amint lehetséges, induljon el ebben az irányban.

A következő hónapban sebességet váltunk és a Zope-ot kezdjük vizsgálni, ezt a nagyrészt Pythonban íródott és az eddigiektől merőben eltérő webalkalmazás-keretrendszer. A Zope az utóbbi pár évben meglehetős népszerűségre tett szert, és gyakran ez az alkalmazás az, amely a Pythont a programnyelvek élvonalába emeli. Érdemes tehát egy pillantást vetnünk a Zope-ra és megvizsgálnunk, hogy mi módon használhatnánk fel saját alkalmazások írására.

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

A JBoss hivatalos lapja a ➲ http://www.jboss.org. A honlapon fórumokat, levelezési listákat, letöltési lapokat és leírást is találhatunk. A Jboss-leírás néhol kicsit zavaros, nem mindig részletezi a dolgokat, és elég komoly EJB-hez kapcsolódó tudásbázist tételez fel. Ugyanakkor majdnem mindent tartalmaz, amire csak szükségünk lehet, és kellemesen egyszerű nyelven íródott, illetve néhány jól használható példakódot is tartalmaz.

Kitűnő bemutatkozó írás EJB-témában *Richard Monson-Haefel* könyve, az Enterprise JavaBeans, amely az O'Reilly kiadásában jelent meg. A ➲ http://www.jboss.org példáival és útmutatójával együtt ez a könyv lehetővé teszi, hogy néhány egyszerű EJB-t viszonylag rövid időn belül el tudjunk készíteni.

Az Ant – a Java válasza a `make`-re és a `Makefile`-okra – az Apache Software Foundation Jakarta Project része. Letölthető a ➲ http://jakarta.apache.org/ant címről.

A Sun honlapja a ➲ http://java.sun.com, igen sok adatot tartalmaz a J2EE-szabványról, az Enterprise JavaBeansről, illetve a kiszolgálóoldali Javáról.