Enterprise JavaBeans

Reuven egy olyan környezetet mutat be nekünk, amelynek segítségével Jboss Java alkalmazáskiszolgálót használó megosztott alkalmazásokat készíthetünk.

Ahogy a webes alkalmazások kezdenek egyre komolyabbá válni, a fejlesztők is egyre igényesebbek lesznek az eszközválasztás terén. Az előzőek folyamán (Linuxvilág 11. szám 74. oldal és 12. szám 74. oldal) két objektumrelációs átalakító eszközt (az Alzabo és a DODS) rejtelmeibe is belepillantottunk. Ezek lehetővé tették, hogy az adatbázisokat objektumokon keresztül kezeljük, s ezáltal elkerüljük az SQL-utasítások kódba illesztését.

Számos olyan helyzet akad azonban, amikor ezek a módszerek sajnos nem alkalmazhatók eredményesen: hogyan lehet például az objektumokat különböző számítógépekre szétválasztani?

Amennyiben valahogy mégis sikerül szétválasztani őket, miként találják meg egymást az objektumok? És ha az objektum adott állapota az adatbázis egy vagy több sorának felel meg, akkor hogyan kezeljük a tranzakciókat?

A fent felsoroltak igen fogós és bonyolult kérdések, olyannyira, hogy akár évekig is elbirkózhatnánk megoldásukkal. Ezekre (és számos más) kérdésre az egyik legátfogóbb megoldást a J2EE (Java 2 Enterprise Edition) felület és az Enterprise JavaBeans objektummodell kínálja. Az EJB-t, ahogy gyakran emlegetik, arra tervezték, hogy összetett, nagyméretű weboldalak készítéséhez nyújtson hathatós segítséget, hiszen nagymértékben képes enyhíteni az infrastruktúrával a programozói háttérrel kapcsolatos gondokat.

E hónapban a JBoss alkalmazáskiszolgáló által megvalósított EJB-változat rejtelmeibe kukkantunk bele. A JBoss a GNU Lesser General Public License (LGPL) felhasználási szerződés hatályába tartozik, nem használ különösebben sok memóriát, és viszonylag egyszerű használni.

Amint azt a Java programozási feladatoknál már megszokhattuk, az EJB-vel való munkához ismét meg kell tanulnunk egy új eszközkészletet kezelni, be kell állítanunk néhány XML fájlt, és természetesen vigyáznunk kell, hogy a fordítás és a futtatás alatt a `CLASSPATH` változó a megfelelő értékeket tartalmazza.

Ha viszont végül ezeken a szerkesztési buktatókon sikerül keresztülverekednünk magunkat, a JBoss kitűnő alapot nyújt a kiszolgálóoldali módszerekkel való hatékony munkához.

Java Enhydra és egyéb gondok

Mielőtt továbblépnénk, fontos, hogy nyomatékosítsuk, az EJB és a Java nyelv nem teljes mértékben szabad programok. Igaz ugyan, hogy nem kell fizetnünk a Java vagy a J2EE könyvtárak letöltéséért, de a Sun birtokol mindent, aminek akár csak a legcsekélyebb köze van a Javához, beleértve a leírásokat és szabványokat is. A Sun Community Source License ugyan nyitottabb, mint sok más felhasználási szerződés, azonban igen messze áll a nyílt forráskódtól.

Ez különösen nyilvánvalóvá vált most, hogy a Lutris Corporation, amely a nyílt forrású Enhydra alkalmazáskiszolgálót támogatta, befagyasztotta a J2EE-minősítésű Enhydra Enterprise kiszolgálójának fejlesztését. A Lutris zárt forrású fejlesztéssé változtatta az Enhydra Enterprise-t, arra hivatkozva, hogy a Sun felhasználási szerződése nem teszi lehetővé, hogy teljes mértékben együttműködő, nyílt forrású J2EE-kiszolgálót hozzanak létre.

Ez természetesen hatalmas felháborodást (és védekezést) váltott ki a legfőbb Enhydra levelezőlistán, és számos megválaszolatlan kérdés is nyitva maradt. Lehet, hogy a Lutrisnak jogilag (és pénzügyileg) valóban muszáj volt megtennie, amit megtettek, azonban az a viselkedésmód ahogyan tették, ékes példája annak, hogyan nem szabad véget vetni egy nyílt forrású fejlesztésnek.

Szerencsére, a JBoss csapat megerősítette, hogy a JBoss továbbra is nyílt forrású marad, tovább növekedik, és támogatni fogja az összes J2EE-szabványt, akkor is, ha az nem rendelkezik a megfelelő hivatalos bizonyítvánnyal, ennek oka pedig általában az, hogy nincs elegendő pénz a Sun bizonyítványának megszerzéséhez.

Mi az az EJB?

A helyes első kérdés ez lenne: „Miért van szükségem az EJB-re?”. Való igaz, sok olyan alkalmazás létezik, amelyhez EJB-t használni merő túlzás lenne. Ugyanakkor az EJB olyan képességeket tesz elérhetővé számunkra, amelyeket magunktól csak nagy erőfeszítések árán fejleszthetnénk ki a kiszolgáló, más néven a tároló (container) belsejében:

- Az EJB lehet az alkalmazással azonos számítógépen, de egy távoli gépen is. Így akár több részből álló alkalmazásokat is készíthetünk, ahol minden egyes rész egy-egy külön gépen fut, miközben a programunk változatlanul működik akkor is, ha egyik részről a másikra másoljuk, vagy megváltoztatjuk a gépek beállításait.
- Az EJB-tároló képes kezelni az objektumrelációs-átalakítás nehézségeit. Csak meg kell határoznunk a táblát és az azt kezelő objektumot, minden további feladatot a tároló végez helyettünk. Avagy, ha jobban szeretjük finomhangolni a dolgokat, lehetőségünk nyílik arra is, hogy babunk saját rétegét módosítsa.
- A relációs adatbázis-kezelők tranzakciókat is képesek használni, ez lehetőséget ad arra, hogy két vagy több műveletet egyetlen műveletként kezeljünk. Az EJB ehhez hasonló tranzakciószerű képességekkel ruházza fel objektumunkat, lehetővé téve, hogy az eljárásunk több műveletet hajtson végre egyetlen „mindent vagy semmit” blokkban.

Fontos tisztában lennünk azzal is, mit nem tud nyújtani az EJB. A hasonló elnevezés ellenére, az Enterprise JavaBeansnek szinte semmi köze nincs a hagyományos

JavaBeanshez. A JavaBeans egységes felülettel rendelkezik, ami lehetővé teszi, hogy JSP-ből egészen kevés kód felhasználásával vagy akár kód nélkül is elérhessük. Az EJB-t ezzel szemben úgy tervezték, hogy bármilyen Java programból használható legyen, ideértve a servleteket is. Továbbá, az EJB-hez tartozó szabványos felület (API) gazdagabb, összetettebb és rugalmasabb, mint a JavaBeanshez tartozó. Sajnálatos módon a JavaBeans kifejezést meglehetősen elkoptatta ez a két népszerű kiszolgálóoldali módszer, azonban úgy tűnik, ez ellen már nem tehetünk semmit.
Az EJB egyik leglenyűgözőbb tulajdonsága, hogy a hozzátartozó API szabványos az alkalmazáskiszolgálókon. Így aztán a fejlesztést elkezdhetjük egy olyan nyílt forrású EJB-kiszolgálón, mint a JBoss, majd, amikor elérkezettnek látjuk az idejét, telepíthetjük egy kereskedelmi kiszolgálóra. (Természetesen az is előfordulhat, hogy amikor megtudjuk, mennyibe is kerülne egy kereskedelmi kiszolgáló, mégis inkább megmaradunk a JBoss mellett.)
A Java-fejlesztésekben talán az a legbosszantóbb dolog, hogy számtalan akronímet, projektnevet és változatszámot kell fejben tartanunk. Ez a cikk például a JDK (Java Development Kit) 1.3 és az EJB 1.1 szabványnak megfelelő Jboss-kiszolgáló 2.4.1a változatára épít. Ráadásul, míg magukat az EJB-osztályokat nem különösebben nehéz létrehozni, a fordítással és üzembe helyezéssel járó feladatok kifejezetten bosszantóak és nehézkesek lehetnek az avatatlanok számára.

Munka az EJB-kel

Az alkalmazásunk EJB alá helyezése azt jelenti, hogy az üzleti logika akkora részét tesszük külön objektumokba, amennyit csak tudunk. EJB alatt az objektumok két fajtáját különböztetjük meg:

- Az entitás babok olyan objektumok, amelyek a relációs adatbázisokat alakítják át. Az entitás babok minden egyes példánya általában az adatbázis egy-egy sorának felel meg. A példányváltozók pedig az adattábla oszlopainak feleltethetőek meg. Az objektumunkhoz tartozó adattáblát hagyományos módon kell létrehoznunk az adatbázisban, azonban az EJB-tároló az összes `SELECT`, `INSERT`, `UPDATE`, és `DELETE` lekérdezést már elvégzi helyettünk.
- A session babok műveleteket hajtanak végre, akár saját maguk akár egy vagy több entitás babon keresztül. A session babok szokott esetben nem rendelkeznek saját állapottal, ez pedig hatékonyabbá teszi őket az entitás babok-nál. Akadnak azonban olyan esetek, amikor jól jön, ha mégiscsak létezik állandó elem a session babban. Ezért aztán az EJB lehetőséget nyújt állandósított session babok kialakítására is, amelyek állapota megőrződik az egyes hívások között.

Ha egy hálózati csevegőszobát szeretnénk készíteni EJB-vel, valószínűleg el kellene készítenünk pár entitás babot (és a nekik megfelelő táblákat) a felhasználók, a hírcsoportok és a küldemények számára. Szükségünk lenne természetesen pár session babra is, amelyek a hozzáadást, módosítást és törlést valósítják meg az imént felsorolt entitásbab-típusokra, illetve egész hírcsoportokat vagy egyedi leveleket kérdeznek le.

A JBoss telepítése

A JBoss Java alkalmazáskiszolgáló, amely lehetővé teszi, hogy többszörözött J2EE-alkalmazásokat használjunk. A JBoss nem is próbálja meg kezelni az alkalmazásoldali eseményeket; ha ilyesmire van szükségünk, akkor a Jakarta-Tomcat vagy más servlettárolóhoz kell fordulnunk. Elérhetővé tesz viszont olyan háttérszolgáltatásokat, mint a könyvtárszolgáltatás, az üzenetváltó szolgáltatás illetve maga az EJB-tároló.
Feltételezve, hogy már korábban telepítettük a JDK-t, a JBoss telepítése meglepően könnyű. A Sun rpm-formátumban is elérhetővé tett egy JDK-másolatot, amit a
➲ http://www.java.sun.com honlapról tölthetünk le. Ezenkívül még az Ant eszközt is le kell töltenünk és telepítenünk. Ez a Java program a vénséges-vén `Unix make` programot hivatott helyettesíteni. Ha tájékozottak vagyunk az XML-formátumban és ismerjük a `make` programot, akkor az 1. listában látható (26 CD Magazin/Javabeans), az Ant által használt ***build.xml*** fájl formátumát is viszonylag egyértelműnek fogjuk találni.
A JDK és az Ant telepítése után a JBoss feltelepítése már gyerekjáték. Letöltöttem a bináris kódot a
➲ http://www.jboss.org honlapról, ahol az összevont Jboss- és Jakarta-Tomcat-támogatást állítottam be. A fájl zipállományként töltődik le, ez azt jelenti, hogy szükségünk lesz az Info-zip-alkalmazásokra is (amelyek az általam használt Linux-terjesztésben alapértelmezés szerint benne foglaltattak), hogy kicsomagolhassuk őket.
Kicsomagolás után a Jboss-Jakarta-terjesztés két alkönyvtárat tartalmaz, ***jboss*** és ***tomcat*** néven. Állítsuk be a `JBOSS_DIST` környezeti változót úgy, hogy a ***jboss*** könyvárra mutasson. Így a különféle Jboss-szal kapcsolatos eszközök és lehetőségek képesek lesznek megtalálni a megfelelő fájlokat.
Ha idáig eljutottunk, akár el is indíthatjuk a Jboss-kiszolgálót a következő két paranccsal:

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

Alapértelmezés szerint a JBoss elég sok adatot naplóz a terminálablakban.

Egy számológép bab megírása

Az első EJB-nk a ***Calculator*** (számológép) lesz. Ez egy állandó tag nélküli session bab, melynek `multiply()` tagfüggfénye két egész számot vár, és a szorzatukat adja vissza.
Miután megírtuk a Calculatort és a hozzá tartozó EJB-felületeket, megvizsgáljuk, hogyan tudjuk használni egy önálló Java programból.
Egy szorzóeljárással felruházott egyszerű számológép-osztály elkészítése alapesetben nem lenne túl nagy feladat. Egyszerűen létre kellene hoznunk a ***calculator.java*** fájlt, és meg kell határozni benne a tagfüggvényt a következő módon:

```
public int multiply (int num1, int num2)
```

Az EJB lehetővé teszi, hogy ***Calculator*** babunkat távolról is megkereshessük és meghívhassuk, ez egyben azt is jelenti, hogy meg kell írnunk azokat a tagfüggvényeket, amelyek segítségével meg lehet találni a ***Calculator***t. Végső soron az alkalmazásunk egy távoli hivatkozással fog dolgozni, ahelyett hogy a valódi objektumot érné el. A session babok írása tehát egy Java-osztály és két csatolófelület létrehozását jelenti.
A Java-osztály az a babosztály, amely a tényleges munkát végzi. A babosztály nem tud róla, hogy egy másik gépen lévő objektumról hívták meg; bár lekérdezheti a környezetét, azonban szokásos esetben ez nemigen szükséges. A babosztályához általában az EJB egyszerűsített nevét használják,

3. lista ejb-jar.xml, a Calculator bab telepítés-leírója

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

<ejb-jar>
  <description>ATF Calculator Session
  Bean</description>
    <display-name>Calculator Session Bean
      ➥/display-name>
  <enterprise-beans>
    <session>
       <ejb-name>Calculator</ejb-name>

<home>il.co.lerner.calculator.CalculatorHome
➥</home>

<remote>il.co.lerner.calculator.Calculator
➥</remote>
       <ejb-class>il.co.lerner.calculator.
➥CalculatorBean</ejb-class>
       <session-type>Stateless
➥</session-type>
       <transaction-type>Bean
➥</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

kiegészítve a „bean” szócskával. A ***Calculator*** EJB-nk babosztálya így a ***CalculatorBean*** nevet kapja és a ***CalculatorBean.java*** fájlban határozzuk meg. A babosztálynak, típusának megfelelően, létre kell hoznia egy ***SessionBean*** vagy egy ***EntityBean*** csatolófelületet.
Az első csatolófelület a távoli csatolófelület (remote interface), amely lehetővé teszi, hogy az alkalmazás megtalálja ***Calculator*** EJB-t, illetve hivatkozást kérjen hozzá. A távoli csatolófelületnek hagyományosan egyszerű nevet adunk, mint például ***Calculator***, és ennek megfelelően a ***Calculator.java*** fájlba helyezzük. A távoli csatolófelület a babosztály minden egyes nyilvános eljárásához meg kell határoznia egy tagfüggvényt. A távoli csatolófelületnek az `EJBObject`-osztályt kell kibővítenie.
A második csatolófelület, a saját csatolófelület (home interface), amely lehetővé teszi az EJB-tároló számára, hogy létrehozza, beazonosítsa és törölje, illetve más módokon kezelje az Enterprise JavaBeant. A saját csatolófelületnek hagyományosan ugyanazt a nevet adjuk, mint a távoli csatolófelületeknek, kibővítve a ***Home*** szóval. Az EJB-nk saját csatolófelületének neve tehát ***CalculatorHome*** lesz, és a ***CalculatorHome.java*** fájlban helyezkedik majd el. A saját csatolófelületnek az `EJBHome`-osztályt kell kiterjesztenie.
Az egyik igen hasznos dolog az EJB-ben, hogy a létrehozott osztályaink a legtöbb esetben az alapértelmezett EJB-megoldásokra hagyatkozhatnak. Meglehet, nem ez a leghatékonyabb módja a dolgok kezelésének, de ezáltal a kód funkcionális részeinek megírására összpontosíthatunk az EJB-tárolóra hagyva a háttér kezelésének döntő részét.

Az osztályok megírása

Most, hogy már megértettük milyen osztályokat kell létrehoznunk, végre elkezdhetünk magával a kóddal foglalkozni.

Hamar észre fogjuk venni, hogy valójában nem is kell sok kódot írnunk. A ***Calculator*** bab esetében például, sok tagfüggvényt csak üres programtestek határoznak meg.
Ez azért van így, mert a ***SessionBean*** csatolófelület, amelytől a ***CalculatorBean*** örököl, rákényszerít bennünket arra, hogy akkor is megadjuk ezeket a tagfüggvényeket, ha a babunk olyan egyszerű, hogy nem is használja őket. Az üres testek használatával teljesítjük a csatolófelület elvárásait, ugyanakkor a kód is egyszerű marad.
Az összes Java fájlt az ***il.co.lerner.calculator*** csomagba helyeztem, bizonyítván, hogy az én üzleti tartományomból érkeztek, illetve hogy ez a ***Calculator*** nevű projekt. Ennek megfelelően az összes ***.java*** forrásfájlt a ***/il/co/lerner/calculator*** könyvtárszerkezetben helyeztem el.
A babosztályunk a ***CalculatorBean*** (lásd a 2. listát 26. CD Magazin/Javabeans), egyetlen `multiply` tagfüggvényt határoz meg, amely két egész szám bemenetet vár, és egy egész számot térít vissza a hívóhoz. A munkamenet (session) csatolófelület-megvalósításán kívül a ***CalculatorBean***-nek nem túl sok köze van az EJB-hez. Tulajdonképpen ez egy meglehetősen unalmas osztály egyetlen tagfüggvénnyel. Minden, amit a ***System.out***-ra írunk az a JBoss naplójába fog kerülni.
A saját csatolófelületünk, a ***CalculatorHome*** segítségével létrehozhatunk egy új ***CalculatorBean*** példányt. A saját csatolófelület megadja a csatolófelület leírását, ideértve, hogy a visszatérési értéke a távoli csatolófelület (Calculator) egy példánya. Ezenkívül már csak nyúlfarknyi kódot tartalmaz:

```
package il.co.lerner.calculator;

import java.io.serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface CalculatorHome extends EJBHome
{
   Calculator create() throws RemoteException,
   ➥CreateException;
}
```

Végül a ***Calculator*** nevű távoli csatolófelületünk a ***CalculatorBean*** minden nyilvános eljárásához megad egy-egy tagfüggvény-meghatározást:

```
package il.co.lerner.calculator;

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

public interface Calculator extends EJBObject
{
   public int multiply(int num1, int num2 )
➥throws RemoteException;

}
```

Az ügyfélprogram nem közvetlenül a babban, hanem a távoli csatolófelületen keresztül fogja meghívni a tagfüggvényeket. A távoli csatolófelület és a babosztály megadásának meg kell egyeznie, különben komoly nehézségekkel kell szembenéznünk a későbbiek folyamán.

4. lista UseCalculator.java, amely összekapcsolódik a Calculator EJB-vel és felhasználja azt

```
package il.co.lerner.calculator;


import javax.naming.initialContext;
import javax.rmi.portableRemoteObject;

import il.co.lerner.calculator.calculator;
import
➥il.co.lerner.calculator.calculatorHome;


class UseCalculator
{
  public static void main(String[] args)
  {
    try
       {
          //ElnevezØs context megszerzØse
          InitialContext jindiContext = new
            ➥InitialContext ();

          //Calculator Bean hivatkozÆs
          // megszerzØse
          object ref = jindiContext.lookup
      ➥("calculator/calculator");
          System.out.println("Got reference");

          //HivatkozÆs megszerzØse a
          //bab sajÆt csatol fel letØhez
          CalculatorHome home =
      ➥(CalculatorHome)
             PortableRemoteObject.narrow
      ➥(ref, CalculatorHome,class);

          //Calculator objektum kØsz tØse a
          // a sajÆt csatol fel letbıl
            Calculator Calculator =
      ➥home.create()

          //multiply() megh vÆsa
          System.out.println("Multiplying
➥2 x 3:");

System.out.println(Calculator.multilpy(2, 3));
       }
      catch(Exception e)
          {

System.out.println(e.toString());
          }
  }
}
```

A bab telepítése

Most, hogy elkészítettük, itt az ideje, hogy a ***Calculator*** nevű session babunkat elültessük a Jboss-kiszolgálóba. A session babunk elültetése annyit jelent, hogy valamennyi összetevőt egyetlen Java archív állománnyá (jar) alakítjuk. A ***.jar*** állománynak a lefordított `Calculator`-, `CalculatorHome`- és `CalculatorBean`-osztályokat fogja tartalmazni.
Található benne továbbá egy ***ejb.jar.xml*** nevű XML fájl, az úgynevezett „telepítés-leíró" (deployment descriptor), amely az EJB-tároló számára írja le a ***.jar*** fájl tartalmát. A telepítés-leíró kötelező része az EJB-szabványnak és minden alkalmazáskiszolgálón azonosan működik. Feladata az, hogy közölje az EJB-tárolóval az általunk kiválasztott osztályok és csatolófelületek neveit, illetve magunk is meghatározhatunk itt bizonyos elemeket, például a babunk által támogatott tranzakciók típusait.
A ***Calculator*** EJB-nkhez tartozó telepítés-leírót a *3. listában* találjuk. Ezt is ugyanoda kell helyeznünk, ahová a többi ***.java*** forrásfájt tettük.
A ***.jar*** fájlunk tartalmaz egy ***jboss.xml*** nevű rövidke XML fájlt is, amit az ***ejb-jar.xml*** mellé fogunk helyezni:

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

A ***jboss.xml*** fájl egyedi Jboss-jellegzetesség, amely a babunkat a Java elnevezés és könyvtár felületéhez (Java Naming and Directory Interface azaz JNDI) kapcsolja. Ha a ***jboss.xml*** a helyén van, bármely ügyfélprogram, amely a JNDI-n keresztül a ***calculator/Calculator*** után érdeklődik, visszakaphatja a vonatkozó hivatkozást.
Kézzel is felépíthettük volna a ***.jar*** fájlt, azonban miért ne használhatnánk fel az Antot a fájl létrehozására és megfelelő helyre mozgatására. Az 1. lista egy Ant ***build.xml***-t tartalmaz, amely elvégzi a telepítést és kezeli az (alapértelmezett) ***ejb-jar***-állományt. Ha az ***build.xml***-t a `$CALCULATOR` könyvtárba helyezzük, a ***.java*** fájljainknak, illetve az ***ejb-jar.xml*** és ***jboss.xml*** fájloknak a `$CALCULATOR/il/co/lerner/calculator` könyvtárban kell kerülniük. Az eredményállományokat az Ant a `$CALCULATOR/build/calculator` könyvtárba helyezi, a ***build.xml*** `build.calculator.dir` tulajdonságának megfelelően.
Ha az Ant a `$ANT` könyvtárba telepítettük, lefordíthatjuk ***.java*** fájljainkat, ezáltal EJB-megfelelő ***.jar*** fájllá változtatva őket (az ***ejb-jar.xml*** a kötelező `META-INF` könyvtárba kerül) és beilleszthetjük a JBoss rendszerébe a következő paranccsal:

```
$ANT/bin/ant deploy
```

Számos üzenetet láthatunk a képernyőn, melyek a fordítás és a telepítés menetét írják le. Ha a fordítás vagy a telepítés kudarcot vall, ellenőrizzük, hogy a környezeti változóink jól vannak-e beállítva vagy a Java fájlokban nem maradt-e hiba, illetve hogy a könyvtárak megfelelő jogosultságokkal bírnak-e.
Amennyiben a Jboss-kiszolgáló már fut, a Calculator ***.jar*** fájl telepítésekor észre fogjuk venni, hogy a kiszolgáló újraindítás nélkül, önműködően felismeri és beilleszti a fájlt.
Ez a JBoss egyik nagyon kényelmes szolgáltatása; ha telepíteni szeretnénk egy fájlt, egyszerűen csak bemásoljuk a `$JBOSS_DIST/deploy` könyvtárba.

Írjunk alkalmazást!

Most, hogy végre elkészült egy telepített ***Calculator*** EJB-nk, írjunk egy rövid Java fájlt, amely ki is használja. A *4. listább* van egy ilyen osztály, mely az ***UseCalculator.java*** forráskódját tartalmazza.

Bár programunk teljes mértékben független az EJB-osztályoktól, és külön is lefordítható, valamint futtatható (akár egy másik gépen is), most is az Antot használjuk a `CLASSPATH` követésére (amelynek egyaránt kell tartalmaznia a ***.jar*** fájlunkhoz tartozó és a Jboss-osztályokat), a fordításhoz és a futtatáshoz is. Az alkalmazásunk futtatásához egyszerűen ennyit szükséges begépelnünk:

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

Az Ant lefuttatja a programunkat, azonban előbb biztosítja, hogy EJB-nk le legyen fordítva, ***.jar*** fájllá alakított és fel van telepítve.

Bármi, amit a `UseCalculator.main()` a ***System.out***-ra ír (más néven `stdout` fájlkezelő), megjelenik a képernyőn, amikor lefuttatjuk az Antot. Ugyanakkor minden, amit a `CalculatorBean` tagfüggvény ír a `stdout`-ra, az a JBoss naplókimenetén fog megjelenni.

Ha az Antot az egyik, a Jbosst pedig egy másik terminálablakban futtatjuk, láthatjuk amint párbeszédet folytatnak egymással.

Az `UseCalculator main()` tagfüggvénye néhány hagyományos lépést alkalmaz EJB-nk elérésére és használatára. Először felvesszük a kapcsolatot a JNDI-vel, amely a Jbossban jelenleg telepített objektumokat tartja nyilván. Ezt a kapcsolatot nevezik `context`-nek. A programunk egy rövid tulajdonság-leíró fájlt, a ***jndi.properties***-t keresi, amelyből megtudhatja, hol találhatja a `context`-et (ezt a fájlt a ***build.xml***-nek megfelelően a `$CALCULATOR/resources/` könyvtárba kell helyezni). Ez a fájl Java erőforrás-formátumú, ahol minden sor `nØv=ØrtØk` alakú.

```
java.naming.factory.initial =
➥org.jnp.interfaces.NamingContexFactory
java.naming.provider.url = localhost:1099
java.naming.fasctory.url.pkgs =
➥org.jboss.naming:org.jnp.interfaces
```

Amint megszereztük a `contex`-et, megkereshetjük az objektumunkat az ***ejb-jar.xml***-ben, azon a néven, amit a ***jboss.xml***-ben adtunk neki. A ***jboss.xml*** nélkül a JBoss nem rendelt volna helyes nevet az EJB-nkhez, ebből következik, hogy nem is tudtuk volna megtalálni a JNDI-n keresztül.

A JNDI egy objektumhivatkozást ad vissza, amit aztán ***CalculatorHome*** példánnyá sorolhatunk be, amivel végül létrehozhatjuk a ***Calculator*** egy példányát. Figyeljük meg, hogy a ***CalculatorBean*** példány helyett a ***Calculator*** (a távoli csatolófelület) egy példányát hozzuk létre. A távoli csatolófelület ugyanis átlátszó kapcsolatot biztosít számunkra a kiszolgálón lévő ***CalculatorBean*** példánnyal, akárhol legyen is az. Így aztán egyáltalán nem kell tudnunk, hol van valójában a ***CalculatorBean*** példány.

Végül, meghívjuk a ***Calculator***-ban (a távoli csatolófelületen) megadott egyik tagfüggvényt. A tagfüggvényhívásunk a `CalculatorBean`-hez (a babosztályhoz) továbbítódik, ahol végrehajtódik, kiír néhány naplóadatot, majd visszatér (ahol aztán az eredményt a `stdout`-ra írjuk).

Összegzés

Írásunkban az Enterprise JavaBeans rejtelmeibe pillanthattunk bele, amely tulajdonképpen Javát használó megosztott alkalmazások létrehozására használható háttér. Igaz, az EJB messze összetettebb, mint a SOAP (lásd Linuxvilág 2001. áprilisi szám 70. oldal), XML-RPC vagy egyéb megosztott objektumrendszerek, azonban sokkal bonyolultabb feladatok kezelésére is tervezték. (Például a SOAP ne is próbálja kezelni a tranzakciókat; hagyja az alkalmazásrétegre ennek megvalósítását.)

Másrészről a Javával való munka gyakran azt jelenti, hogy az ember több időt tölt felügyeleti és logisztikai feladatokkal, mint tényleges programozással. Kitalálgatni, hogy melyik fájlt melyik könyvtárba kell helyezni, elég elkedvetlenítő lehet, különösen, ha valamilyen sokkal dinamikusabb nyelven szoktunk dolgozni, mint például a Perl vagy a Python. Mindazonáltal a fájdalom gyorsan elmúlik, amint meglátjuk, milyen könnyedén tudunk megosztott alkalmazásokat írni az EJB segítségével. Az a tény, hogy a JBoss könnyen letölthető, telepíthető és futtatható, ráadásul viszonylag kevés memóriát fogyaszt, lehetővé teszi mindenki számára, hogy kipróbálja az EJB-t.

Következő alkalommal folytatjuk a munkát az EJB-vel. Belenézünk a belsejébe és megvizsgáljuk az entitás babokat, amelyek a relációs adatbázisok eléréséhez nyújtanak számunkra objektumalapú felületet.

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 oldala a ➲ http://www.jboss.org. A honlapon találunk fórumot, levelezési listát, letöltési oldalakat és leírást is. A Jboss-leírás néhol kicsit zavaros, nem mindig írja le átfogóan a dolgokat és meglehetősen jó EJB-tudást feltételez. Ugyanakkor a legtöbb dolgot, amire szükségünk lehet, megtaláljuk itt. Nyelvezete elég egyszerű és szép példaprogramokat is találhatunk, amelyeket könnyen fel tudunk használni.

Richard Monson-Haefel tollából az Enterprise JavaBeans című O'Reilly által kiadott könyve, kitűnő forrásmunka az EJB-témában. A ➲ http://www..jboss.org bevezetőjével és útmutatóival kiegészítve, e könyv segítségével már viszonylag rövid idő alatt elkezdhetünk egyszerű EJB-ket írni.

Az Ant a `make` és a Makefile fájlok javás változata, az Apache Software Foundation Jakarta projektjének része. Az Antot a ➲ http://*jakarta.apache.org/ant* címről tölthetjük le.

A Sun honlapja a ➲ http://www.java.sun.com rengeteg adatot tartalmaz a J2EE-leírásról, az Enterprise JavaBeans-ről és a kiszolgálóoldali Java általános alkalmazásairól.