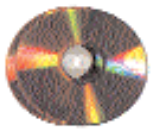

Bemutatkozik az Enhydra

Ügyes segédlet arról, hogyan írjunk servleteket és alapalkalmazásokat az Enhydrával érkező eszközök segítségével.

Cikkünkben azt fogjuk megvizsgálni, hogyan kell servleteket és alapalkalmazásokat készíteni az Enhydrával érkező eszközökkel. Bár az Enhydra webalkalmazásai annyira sem szabványosak, mint a Jakarta-Tomcat servletei, azért jókora erőt képviselnek, és lehetőséget adnak arra, hogy teljesen nyílt forráskódú környezetben használjunk Enterprise JavaBeanst.

Történet és háttér

Az Enhydra Javában írt nyílt forráskódú alkalmazáskiszolgáló, melynek legfőbb célja a Sun J2EE meghatározásainak minél pontosabb betartása. Az Enhydra-fejlesztés éllovasa, a Lutris cég BSD-szerű szabadalom alá helyezte az alkalmazáskiszolgálót. A jelenlegi üzembiztos Enhydra-változat (a 3.5-ös) nagy mennyiségű szabványt támogat, beleértve a servleteket és a JSP-ket is. Az Enhydra Enterprise, melynek tervezett kiadási ideje 2001 nyara volt (lapunk kiadásakor még nem érhető el – a szerk.), további J2EE-képességekkel rendelkezik majd, beleértve az Enterprise Java Beans (EJB) támogatását is. A teljes Enhydra nyílt forráskódú termékként hozzáférhető, azaz letölthetjük a Hálóról, és nyugodtan telepíthetjük. Az olyan ügyfelek számára, akik gyanakvással tekintenek a nyílt forráskódú programokra – tehát szívesebben használnának előrecsomagolt terméket, amelynek minőségét ellenőrizték, továbbá támogatást szeretnének kapni a Lutristól –, az Enhydra kereskedelmi változatai is megvásárolhatók. Természetesen gyanítom, hogy a legtöbb embernek, aki e sorokat olvassa, nemigen van szüksége a Lutris támogatására, de azért jó tudni, hogy készek és hajlandók segíteni másokat a termék használatában.

A Lutris nemcsak a J2EE-piacot célozta meg, hanem a vezeték nélküli internetalkalmazásokét is. Még nem találom túl elterjedtnek a mobil internettechnológia használatát – a mobiltelefonom WAP-szolgáltatása jóindulattal is csak szánalmasnak nevezhető –, de a Lutris ennek az idővel elkerülhetetlenül fontos piacnak egyik játékosaként tekint az Enhydrára.

Bár az Enhydrának van még mit fejlődnie a Zope vagy az ArsDigita Community Systemhez képest a mindshare és a közösség terén, már így is meglehetősen nagyszámú piaci sikert könyvelhet el magának. Például a Hewlett-Packard mostanában jelentette be, hogy a továbbiakban együttműködik a Lutrisszal, és számos alkalmazásban Enhydrát fog használni. Ez ékesen bizonyítja, hogy a nyílt forrású alkalmazáskiszolgálók igenis fontos helyet töltenek be a webes alkalmazásfejlesztés világában, és még olyan cégek esetében is szóba jöhetnek, amelyek egyébként sokkal többet is képesek lennének fizetni értük.

Kezdjünk hozzá!

Most, hogy egy kicsit feltártuk a háttérben zajló folyamatokat, próbáljunk meg az Enhydra segítségével egyszerű alkalmazásokat készíteni. Első feladatunk természetesen a termék letöltése és telepítése lesz. A magam részéről az Enhydra Enterprise próbaváltozatával dolgoztam, mivel EJB-képességeit szerettem volna kipróbálni. A próbaváltozat az Enhydra Enterprise JDK 1.3-ra támaszkodik, ami kellemes újdonságnak tekinthető az előző változatokhoz képest, amelyek a JDK 1.2-vel működtek annak ellenére, hogy az 1.3 már jó ideje elkészült.

Letöltöttem és kicsomagoltam az Enhydra tarfájlt, amely számos fájlt és könyvtárat hozott létre az enhydra4.0 könyvtárban. A leírást szokás szerint a doc könyvtár tartalmazza, a lib könyvtárban találhatók azok a .JAR-fájlok, amelyek az Enhydra számára szükségesek, végül pedig a conf könyvtár tartalmazza az Enhydra általános beállítófájljait. Itt helyezkedik el továbbá egy bin könyvtár is, amelyben csaknem kizárólag parancsfájlok rejlenek (a windowsos .BAT megfelelőjükkel), ezek hajtják végre az Enhydrát beállító különféle Java-programokat.

Futtassuk le a `configure` parancsfájlt az Enhydra-terjesztés főkönyvtárában (melyre a következőkben $ENHYDRA-ként fogok hivatkozni), hogy biztosak lehessünk benne: az Enhydra-parancsfájlok és Javaprogramok figyelembe veszik azt a könyvtárat, ahová az Enhydrát telepítettük. A `configure` egyetlen kötelező értéket vár: a JDK-telepítésünk könyvtárát. A `configure` számos Makefile-t és más beállításfájlt módosít, de semmilyen kód újrafordítását nem kényszeríti ki. A `configure` lefuttatása után (amely semmilyen látható kimenetet nem fog adni) az $ENHYDRA könyvtárban a `bash` parancsfájlt (`setup.bash`) ugyancsak futtassuk le, ez a JDK végrehajtható állományokat tartalmazó könyvtárát adja hozzá a PATH-hoz.

Az Enhydra számos különféle, egymással összekapcsolódó programcsomagot tartalmaz. Maga az alkalmazáskiszolgáló (más néven multiserver) képes közvetlenül a szemközti HTTP-ügyfelekkel dolgozni, vagy akár egy olyan proxyként működő webkiszolgálóval együttműködni, mint az Apache. Ha a `loadOrder` tulajdonságot az $ENHYDRA/conf/bootstrap.conf fájlban átírjuk, csökkenthetjük az alkalmazáskiszolgáló által elindított szolgáltatások számát, vagy megváltoztathatjuk az indítási sorrendjüket.

A kiszolgáló indításához egyszerűen csak futtassuk le az `$ENHYDRA/bin/multiserver` parancsot. A program indulásakor a szolgáltatásokat egymás után indítja, míg végül a következő üzenetet láthatjuk: „Bootstrapper initialized normally”, azaz a betöltő parancsfájl hiba nélkül lefutott. Ezen a ponton kipróbálhatjuk a kiszolgálót: ha a böngészőt a 8001-es kapura irányítjuk, egy irányítópultot hoz fel, amelyen megfigyelhetjük a multiserver pillanatnyi állapotát.

Az indításra tett első kísérleteim kudarcot vallottak, mivel a program mindig arról panaszkodott, hogy nem találja az enhydra.jar fájlt a CLASSPATH-ban. Ez azért volt különösen zavaró, mert az egész Enhydra-terjesztésben semmiféle enhydra.jar nevű fájlnak nem akadtam a nyomára.

A megoldás, mint kiderült, meglehetősen egyszerű, ha nem is nyilvánvaló: az Enhydra tudja, hogy milyen CLASSPATH szükséges az egyes programok futtatásához, de ezeket a beállításokat figyelmen kívül hagyja, ha a CLASSPATH-t már eleve beállítottuk. Ezért a CLASSPATH-t még a kiszolgáló futtatása előtt töröljük, eltávolítva ezáltal a környezeti változót. Ha megtettük, a multiserver a várakozásnak megfelelően fog elindulni.

Egyszerű webalkalmazás készítése

Az elmúlt pár hónapban bepillantást nyerhettünk a Java-servletek és a JavaServer Pages világába. Az Enhydra mint J2EE-megfelelő

1. lista Foo.java

```
/*
 * Egyszerű servlet, amely azt mutatja be,
 * hogyan lehet azokat az Enhydrába illeszteni.
 * Ez a servlet a /foo URL alatt lesz elérhető.
 */

package atf.presentation;

// Servlet import
import javax.servlet.*;
import javax.servlet.http.*;

// Standard import
import java.io.*;
import java.text.*;
import java.util.*;

public class Foo extends HttpServlet {


   public void doGet(HttpServletRequest request,
     HttpServletResponse response)
       throws ServletException, IOException
   {
       // kimeneti folyam a STDOUT-ra
       PrintWriter out = response.getWriter();

       // kiírunk némi HTML-t
       out.println("<HTML><Head><Title>");
       out.println("Foo</Title></Head>");
       out.println("<Body>");
       out.println("<P>Foo</P>");
       out.println("</Body>");
       out.println("</HTML>");
   }
}
```

alkalmazáskiszolgáló ezeket a módszereket teljes mértékben támogatja. Ráadásul nyílt forrású kiszolgáló lévén az Enhydra a servlet és a JSP-motor alapjául a Jakarta-Tomcat-motort használja. Amint azt nemsokára látni fogjuk, lehetőség nyílik arra (sőt, gyakran előnyösebb), hogy az Enhydra saját fejlett webalkalmazásait használjuk.
Az Enhydrával érkező eszközökkel viszonylag könnyű servleteket készíteni. Az Enhydra alkotói minden bizonnyal sok időt szenteltek egy olyan rendszer megalkotásának, amely nemcsak telepítve erős, hanem a fejlesztés alatt is viszonylag könnyen kezelhető.
Ha valaki általában csak servleteket szokott írni, fordítani és könyvtárba helyezni, tapasztalni fogja, hogy az eljárást megnehezítvén az Enhydra csak útban van. Ez nem kis részben annak a módnak „köszönhető”, amely szerint az Enhydra-alkalmazásokat telepíteni kell: külső kiszolgáló igénybevétele nélkül, ugyanis az Enhydra arra számít, hogy a legtöbb webalkalmazást a többitől függetlenül szeretnénk kipróbálni (és futtatni).
Egyszerű servlethez az Enhydra részét képező alkalmazáskészítő varázslót fogjuk használni (`appwizard`), mely az `$ENHYDRA/bin/appwizard` paranccsal hívható meg. Az `appwizard` ugyan nem IDE, de elég kifinomult fájlmásoló program, amely olyan egyszerű csontvázalkalmazásokat képes készíteni, melyek már önmagukban működnek.
Az `appwizard` első futtatásakor megkérdezi, hogy webalkalmazásokat (például egyszerű servlet) vagy Enhydra-szuperservleteket szeretnénk-e fejleszteni. Válasszuk az egyszerű (standard) webalkalmazást. (A szuperservletekről később még szót ejtünk.) A következő képernyő arra kérdez rá, hogy HTML-kimenetet szeretnénk-e vagy WML-t, ez utóbbi lassanként a mobil internetalkalmazások XML-alapú szabványává válik. A HTML-t fogjuk használni, és a projektkönyvtárat, valamint a csomagot is „atf”-nek nevezzük. Alapértelmezés szerint az Enhydra-alkalmazások a saját könyvtárunkba, az enhydraApps alkönytár alá kerülnek. Válasszunk a kiadott kódhoz szabadalomtípust, és az `appwizard` máris elkészíti új alkalmazásunkhoz a fájlokat.
Az `appwizard` nagyszámú önműködően készített fájlt és könyvtárat hoz létre, többek között:

- Egy általános Makefile-t, amely lehetővé teszi, hogy az alkalmazást lefordítsuk. Az egyes alkalmazások alkönyvtáraiban szintén egyedi Makefile-okat találunk.
- A config.mk-t, ez számos környezeti változót állít be, amelyek a Makefile futását befolyásolják. Olyan adatokat találhatunk itt, mint az Enhydra-változat, a JDK-telepítés és az Enhydra-telepítés könyvtárát.
- Az src könyvtár a Java-servletekhez és a HTML-fájlokhoz tartozó kódot tartalmazza. Az src-n belül egy általános WEB-INF könyvtárra bukkanhatunk, amelyben web.xml fájl nevez meg minden egyes telepítendő servletet.
- Az atf könyvtárat, melynek neve a készülő projekt nevétől függ. Négy alkönyvtárat tartalmaz: business, data, presentation és resources. Minket most leginkább a presentation és a resources könyvtár érdekel, hiszen az első tartalmazza a servleteket, a második pedig a HTML-fájlokat és a JSP-ket.

Az alkalmazás felépítéséhez futassuk le a `make`-et a projekt gyökérkönyvtárában. (Az Enhydra Enterprise leírásában kiemelték, hogy a `make` helyett immár Java-alapú Ant-építőeszközt alkalmaztak, de úgy tűnik, az alkalmazáskészítés azért továbbra is a `make`-en alapul.)
Miután a `make` befejezte munkáját, alkalmazásunk legfelső szintjén, az src és az input mellett új kimeneti alkönyvtárat kell találnunk. A kimenő könyvtár mindent tartalmaz, ami az alkalmazás elindításához szükséges lehet, beleértve a .CLASS fájljainkat tartalmazó normál Java .WAR (web archivum) fájlt, XML-meghatározókat, JSP-ket és képeket:

```
WEB-INF/classes/atf/presentation/WelcomeHTML.class
WEB-INF/classes/atf/presentation/WelcomeServlet.class
WEB-INF/classes/atf/presentation/RedirectServlet.class
media/Enhydra.gif
index.jsp
WEB-INF/web.xml
```

Figyeljük meg, hogy itt már három .CLASS fájlunk létezik, holott az src/atf/presentation/ csak kettőt tartalmazott. Ez azért történik így, mert az XMLC a src/resources könyvtár HTML-fájlját Java-forrásfájllá alakította, mely végezetül Java .CLASS fájllá alakult át.
Így mindössze két parancs (az `appwizard` és a `make`) segítségével képesek voltunk egy teljes, működőképes Enhydra-alkalmazás elkészítésére. Az alkalmazás jelen állapotában semmi különöset vagy érdekeset nem végez, de kitűnő vázat ad, amely módosítható és kibővíthető.
Alkalmazásunkat az `output/start4` segítségével futtathatjuk.

Ez a 9000-es kapun indítja el az alkalmazást (a kapu száma az input/conf/servlet/servlet.conf.in fájlban adható meg). Ha a böngészőnket ekkor a http://localhost:9000-re irányítjuk, a servlet kimenetét fogjuk látni, azaz az Enhydra-logót, alkalmazásunk nevét (atf), a pillanatnyi időt és a dátumot, valamint egy hivatkozást, amely az alkalmazásra visz bennünket vissza.
A HTML-lap az XMLC segítségével készült, és jól szemlélteti, miként épül be az XMLC az Enhydra legtöbb alkalmazásába.
Az XMLC a src/atf/resources/Welcome.html fájlt Java-servletté fordítja, mely ezután .CLASS fájllá alakul. Az XMLC által létrehozott Java-osztály minden egyes `<span>`-taghoz egy horgot (hook) készít, így a `<span>`-tagok között bármit módosíthatunk, amelynek ID-tulajdonságot mutat fel.
A WelcomeServlet, az alkalmazásunk elején végrehajtott servlet a pillanatnyi időt és a dátumot úgy jeleníti meg, hogy egy példányt állít elő az XMLC által készített osztályokból:

```
now = DateFormat.getTimeInstance(DateFormat.MEDIUM).
                ➥format(new Date());
welcome = new WelcomeHTML();
welcome.getElementTime().getFirstChild().setNodeV
                ➥alue(now);
```

Másképpen fogalmazva: a `<span>`-tagok közti ID-vel azonosított szöveget azáltal helyettesítjük az idővel, hogy a HTML-fájlt DOM által elérhető fává alakítottuk, majd egy adott csomópont értékét megváltoztattuk.
Alkalmazásunkhoz további servleteket adhatunk, ha a src/atf/presentation könyvtárba másoljuk, vagy már eleve itt hozzuk létre őket. Fontos tudni, hogy a csomag neve nem egyszerűen atf, hanem atf.presentation lesz.
Ezekután egy nagyon egyszerű osztályt fogunk írni, melynek a Foo nevet adtuk, és az *1. listában* látható. A csomag nevét kivéve semmiféle különbség nem látható a hagyományos servletek és a Foo.java közt.
A servletmotornak meg kell mondanunk, hogy servletünkhöz egy URL-t rendeljen, amit legegyszerűbben az src/WEB-INF/web.xml fájlban tehetünk meg. Ez az XML-állomány két részre osztható: az első rész servletosztályokat rendel servletnevekhez, a második servletneveket URL-ekhez. A *2. listán* a Foo-servletünk kezeléséhez igazított web.xml fájlt láthatjuk.
Végül az új osztályunkat a Makefile-ba be kell vinnünk, amit úgy tehetünk meg, hogy osztályunk nevét hozzáadjuk a CLASSES változóhoz:

```
CLASSES = WelcomeServlet \
          ➥RedirectServlet Foo
```

Futtassuk a `make`-et és ellenőrizzük, hogy a Foo.class valóban hozzáadódott-e az alkalmazáshoz:

```
jar tvf output/archive/atf.war
```

Ha működik, futassuk az `output/start4-et`, és a böngészővel mutassunk http://localhost:9000/foo címre. Most az új Foo servletünk által kiküldött HTML-kimenetet kell látnunk.

Enhydra-alkalmazás készítése

Tapasztalt webfejlesztők nyelvtől vagy környezettől függetlenül rendszerint minden egyes honlaphoz külön programot szoktak írni. Ha öt különböző dinamikusan létrehozott lapot szeretnénk megjeleníteni, akkor öt külön CGI-programot, mod_perl-kezelőt, servletet vagy JSP-lapot kell készítenünk.

2. lista A web.xml módosított változata

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app>
    <servlet>
        <servlet-name>
            welcome
        </servlet-name>
        <servlet-class>
            atf.presentation.WelcomeServlet
        </servlet-class>
     </servlet>
     <servlet>
        <servlet-name>
            redirect
        </servlet-name>
        <servlet-class>
            atf.presentation.RedirectServlet
        </servlet-class>
    </servlet>
     <servlet>
        <servlet-name>
            foo
        </servlet-name>
        <servlet-class>
            atf.presentation.Foo
        </servlet-class>
    </servlet>


    <servlet-mapping>
        <servlet-name>
            welcome
        </servlet-name>
        <url-pattern>
            /welcome
        </url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>
            redirect
        </servlet-name>
        <url-pattern>
            /redirect
        </url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>
            foo
        </servlet-name>
        <url-pattern>
            /foo
        </url-pattern>
    </servlet-mapping>
</web-app>
```

Az Enhydra lehetővé teszi, hogy eltérjünk ettől a modelltől egyedi lapok helyett alkalmazásszinten gondolkodva. Ehhez nyújtanak segítséget a szuperservlet néven ismert alkalmazások, ahol egyetlen alkalmazásobjektumot rendelünk számos megjelenítő felülethez.

A megjelenítő objektumokat az Enhydra URL-ben könnyen azonosíthatjuk: a .po utótag árulja el az Enhydrának, hogy az URL-ben megnevezett objektumot kell meghívnia. Így az Abc.po kérelem az Abc nevű megjelenítő objektumhoz tartozó `run()` eljárást indítja el.
A hagyományos Java servletekkel ellentétben a megjelenítő objektumok minden HTTP-kérelem esetén újraértelmeződnek. Lehet, hogy ez valamivel kevésbé hatékony, mintha egyetlen servletpéldányhoz több szálat használnánk, viszont megkímél minket a szálbiztos kód írásának fáradalmaitól.
Egy egyszerű Enhydra-alkalmazás tehát legalább egy alkalmazásobjektumot vagy megjelenítő objektumot tartalmaz. Ezek a PO-k kapcsolják össze az Enhydra másik két fő objektumtípusát: az üzleti (business) objektumokat (ezek tartalmazzák a gyakran használt szolgáltatásokat) és az adatobjektumokat (ezek állandó tárolók, például relációs adatbázist rendelnek Java-osztályokhoz). Amint már korábban is említettük, az alkalmazás src könyvtárában mindhárom objektumtípus – presentation, business és data – saját könyvtárral rendelkezik. Továbbá az összes ilyen objektum a háromrétegű webalkalmazás egy-egy alapvető rétegét képezi. Ezért – bár beletelhet egy kis idő, amíg megszokjuk a három objektumtípus elkülönítését – ez a modell egyre inkább terjed a webalkalmazások körében.
Akárcsak az imént, az alkalmazás vázának elkészítéséhez az Enhydra `appwizard`-ját használjuk, amit azután megváltoztatunk. Futassuk le újra az `appwizard`-ot és az első képernyő listájából az egyszerű webalkalmazás helyett válasszuk a *super servlet* lehetőséget! Én a projektet egyszerűen myprojectnek neveztem el és az il.co.lerner csomagba helyeztem – cégünknél általában ezt használom a belső projektekhez. Az `appwizard` ezután az alkalmazás vázát az ~/enhydraApps/myproject könyvtárban hozza létre. Az alkalmazás hasonló szerkezetű, mint a servletünk, és hasonló könyvtárszerkezettel rendelkezik. Az src/il/co/lerner alatt egy presentation, egy data és egy business könyvtárunk lesz. Akárcsak az imént, most is létezik egy legfelsőbb szintű Makefile, amely lefordítja és elkészíti szuperservletünket.
Nézzünk bele a presentation/WelcomePresentation.java fájlba, ahol a tulajdonképpeni megjelenítést végző megjelenítő objektum forráskódját találjuk. Amennyiben a legfelsőbb szintű könyvtárba begépeljük a `make` parancsot, az alkalmazás indításához lefuttatjuk az `output/start4` parancsot, majd a webböngészőt a

3. lista WelcomePresentation.run()

```
public void run(HttpPresentationComms comms)
   throws HttpPresentationException, IOException {

   WelcomeHTML welcome;
   String now;

   welcome = (WelcomeHTML)comms.xmlcFactory.create
             ➥(WelcomeHTML.class);
   now = DateFormat.getTimeInstance
         ➥(DateFormat.MEDIUM).format(new Date());
   welcome.setTextTime(now);
   comms.response.writeDOM(welcome);
}
```

➲ http://localhost:9000 címre irányítjuk, azt látjuk, hogy böngészőnk átirányítódik a http://localhost:9000/WelcomePresentation.po címre. Ez a lap ugyanazt a kimenetet adja, mint amit a vázservletünk írt ki, azaz az Enhydra-logót, a pillanatnyi időt és a dátumot láthatjuk.
A po utótag – mint már ismerhetjük – utasítja az Enhydrát, hogy hívja meg a `WelcomePresentation` `run()` eljárást.
Az önműködően létrejövő vázalkalmazás, a `WelcomePresentation.run()` valahogy úgy fog kinézni, ahogyan azt a *3. listában* megfigyelhetjük.
A szuperservlet csatolófelülete hasonló a hagyományos servletéhez, így a servleteket ismerő programozónak nem kerül sok fáradságba az elsajátítása. A `run()` eljárás egyetlen HttpPresentationComms típusú értéket fogad, amely megjelenítő objektumunkat teszi elérhetővé mindazzal a kapcsolattartási lehetőséggel, amire a külső világ eléréséhez szükség van, ideértve a HTTP-kérelmeket és válaszobjektumokat is.
A `run()` eljárás a kimenet megjelenítéséhez elkészíti a WelcomeHTML egy példányát. Ez az a Java-osztály, amit az XMLC a Welcome.HTML-ből készített. Ezt követően a `run()`a „time” ID-vel jelölt `<span>`-tagok tartalmát a pillanatnyi dátummal és idővel helyettesíti. Ezekután írjuk a HTTP-válaszobjektumba az üdvözlés tartalmát, amelyet a DOM-fa tartalmaz.
Elkészíthetjük saját megjelenítő objektumunkat, a FooPresentationst, ahogy a *4. listában* láthatjuk. Ne felejtsük el az új objektumot a presentation könyvtár Makefile-jának CLASSES sorához hozzáadni! Amikor a legfelsőbb szintű alkalmazáskönyvtárból a `make`-et újrafuttatjuk, a FooPresentation lefordítódik és bekerül Enhydra-alkalmazásunkba.
Nagyon szép, hogy saját megjelenítő objektumot tudunk írni, de hol marad az alkalmazásobjektum, amely irányítja? A fő forráskönyvtárban ugyanazon a szinten, ahol a presentation, a data és a business könyvtárakat találhatjuk, létezik egy, a projekt nevével azonos nevű Java-osztályfájl; a mi esetünkben tehát egy src/il/co/lerner/myproject.java fájlt kell keresnünk.

Szuperservlet telepítése

Szuperservletünket önmagában mostanáig az átfogó alkalmazáskiszolgálón kívül futtattuk. Ez a képesség azon fejlesztők számára nagyszerű, akik a fő (termelő) weblap zavarása nélkül szeretnék a programokat kipróbálni, de alkalmazásukat végül a kiszolgálóba építik.
Az Enhydra segítségével viszonylag egyszerűen elvégezhető ez a feladat is: az alkalmazásunkkal kapcsolatos adatokat a kiszolgáló beállítófájljához adjuk, így az megtalálja az alkalmazást. Ezután alkalmazásunkat az irányítópanel segítségével az alkalmazáskiszolgálón egy tetszés szerinti URL alá helyezzük, ezáltal mindenki

4. lista FooPresentation.java

```
/*
 *  myproject
 *
 *  Enhydra szuperservlet megjelenítő objektum
 *
 */

package il.co.lerner.presentation;

// Enhydra Szuperservlet importok
import com.lutris.appserver.server.httpPresentation
➥.HttpPresentation;
import com.lutris.appserver.server.httpPresentation
➥.HttpPresentationComms;
import com.lutris.appserver.server.httpPresentation
➥.HttpPresentationException;


// Standard imports
import java.io.IOException;
import java.util.Date;
import java.text.DateFormat;

public class FooPresentation implements
➥HttpPresentation {


   public void run(HttpPresentationComms comms)
     throws HttpPresentationException,
     ➥IOException {

     // A tartalomtípus célja a felhasználó
     // böngészője

comms.response.setContentType("text/html");


     // Írjunk ki némi HTML-t
     comms.response.writeHTML("<HTML>");
     comms.response.writeHTML("<Head>
     ➥<Title>Foo</Title></Head>");
     comms.response.writeHTML("<Body>");
     comms.response.writeHTML("<P>Foo</P>");
     comms.response.writeHTML("</Body>");
     comms.response.writeHTML("</HTML>");
   }

}
```

számára elérhetővé válik.
Ennek kivitelezéséhez az alkalmazáskiszolgálót még egyszer újra kell indítanunk az `$ENHYDRA/bin/multiserver` parancsfájl segítségével. Ezután a `multiserver` a 8001-es kapun érhető el. Töltsük be a rendszergazdai képernyőt a böngészőbe, és nézzük meg a rendelkezésre álló alkalmazások listáját a bal felső sarokban. Másoljuk az output/conf/myproject.conf nevű alkalmazásbeállító fájlt – de ne a teljes alkalmazást – az $ENHYDRA/apps alá. Ha ezt megtettük, szerkesszük át az $ENHYDRA/apps/myproject.conf fájlt, megváltoztatva a `Server.ClassPath[]` értékét. Két lehetséges `Server.ClassPath[]` érték létezik eleve a myproject.conf fájlban: egy az alkalmazás önálló módú futtatásához, egy másik pedig a `multiserver` alatti működéshez. Tegyük megjegyzésbe az önálló futásértéket (az első), és szedjük ki a megjegyzésből a második értéket.
Most vissza kell térnünk a böngészőhöz, és az irányítópanelén kattintsunk az *Add* gombra (amelyiken egy nagy + jel látható). Egy új alkalmazást szeretnénk beilleszteni, nevének (myproject) már a listában kell lennie. Válasszunk alkalmazásunkhoz egy kezdő-URL-t és gépeljük be azt a szöveget, amit ehhez az alkalmazáscsoporthoz szeretnénk rendelni. Az alkalmazás beillesztéséhez kattintsunk az *OK* gombra, majd frissítsük az irányítópanelt. A bal felső sarokban a „myproject” szót kell látnunk a többi betöltött alkalmazással egyetemben. Ha a myproject névre kattintunk, a képernyő jobb alsó részén adatokat láthatunk a projektről.
Az alkalmazás futtatásához egy vagy több kapcsolatot hozzá kell rendelnünk, majd el kell indítanunk. Alkalmazásunk alapértelmezetten a 8002-es és a 8003-as számú kapun fut, amelyhez – ha akarjuk – egy vagy több kapcsolatot adhatunk hozzá. Ha a kapcsolatokat már meghatároztuk, kattintsunk a képernyő bal szélén található *run* gombra. A kapcsolat URL-ek hivatkozássá válnak, és rájuk kattintva a webalkalmazásunk egyik vagy másik kapcsolatát nyithatjuk meg (azaz az Enhydra-logót és a pillanatnyi időt), egy új ablakban megjelenítve. (A JavaScriptet és az új ablakokat általában bosszantónak találom, de az Enhydra fejlesztői – ami a szép webfelületet illeti – meglehetősen jól egyensúlyoznak ízlésesség és használhatóság között.)
FooPresentation objektumunkat az URL megváltoztatásával ki is

További érdekességek

Az Enhydrával való munka előfeltételezi a Java ismeretét, de természetesen nem árt némi servletekkel kapcsolatos háttértudás sem.

Az Enhydra viszonylag jól megalapozott termék, de a leírás megértése nem kevés időt vesz igénybe. Ráadásul az elérhető leírások jó része az Enhydra 2.x és 3.x változataihoz készült, így némileg már idejétmúlt. Mindenképpen javaslom az Enhydra-kézikönyv elolvasását: ➲ enterprise.enhydra.org/software/documentation/ee4b1/index.html (lásd a képen)

Az Enhydrát általánosságban tárgyaló, néhány XMLC-példával is megtűzdelt kitűnő cikk olvasható *Roger Metcalf* tollából, mely az ArsDigita Systems Journalban jelent meg, és a ➲ http://www.arsdigita.com/asj/enhydra címen érhető el.

Két egymást kiegészítő O'Reilly-könyv elég adatot szolgáltathat ahhoz, hogy elindulhassunk az XMLC témakörében. *Brett McLaughlin*, a Lutris munkatársának „Java and XML” című könyve nagy részletességgel szól a DOM-ról, összehasonlítva a SAX- és más XML-értelmező lehetőségekkel.

A Java Servlet Programming második kiadásában pedig *Jason Hunter* és *William Crawford* szentel egy teljes fejezetet az XMLC-nek, melyet kiszolgálóoldali Javát használó, hasonló dinamikus laplétrehozó rendszerek fejezetei követnek.

próbálhatjuk. Helyettesítsük a WelcomePresentation.po-t FooPresentation.po-ra, ekkor nagy valószínűséggel a Foo HTML-kimenetet találjuk a képernyőn.
Alkalmazásunkat egy vagy több kapuról kizárólag a webalapú irányítópanel használatával is eltávolíthatjuk, esetleg magából a kiszolgálóból. Végezetül a rendszert akár az irányítópanelen leállíthatjuk, vagy üssük le a CTRL+C-t abban a terminálablakban, ahol a multiservert elindítottuk.

Összegzés

Servleteket nem különösebben nehéz írni, de az Enhydra sokkal többet kínál ennél. Különlegessége, hogy olyan környezetet nyújt, ahol a servletek egyszerűen létrehozhatók, valamint teljes, összetett webkiszolgáló futtatása nélkül is kipróbálhatók. Ráadásul az Enhydra által nyújtott szuperservletekkel sokkal könnyebb dolgozni, mint a hagyományos servletekkel, különösképpen azért, mert sem szálproblémákkal, sem pedig az egyes lapokhoz új vezérlő írásával nem szükséges bajlódnunk.
Természetesen néhány hátulütője is akad a dolognak. Akárcsak a többi Java-program, az Enhydra CLASSPATH-beállításához is némi türelem szükséges. (Bár a Lutris javára kell írni, hogy a saját CLASSPATH-változóm eltávolítása az összes gondot megoldotta.) Igaz, az Enhydra önműködően létrehozott makefájljai óriási mértékben képesek csökkenteni a gondolkodási időt, amit egy kész alkalmazás létrehozásába bele kell fektetni, ám a Java-programok azonban még így is legalább tízszer annyi fájlból állnak, mint Perl- vagy Python-testvéreik.
Bár a szuperservletek nyilvánvalóan fejlettebbek „nem szuper” társaiknál, azért én még mindig habozom egy kicsit, mielőtt fejest ugranék egy olyan módszerbe, amely annyira eltér a régi, jól bevált és kitűnően leírt szabványtól – különösképpen most, amikor a nyílt forrás közössége egyre inkább az Enhydrára összpontosít. Végül, mivel az Enhydra Enterprise még nem került forgalomba, a telepítés és a leírás némi kívánnivalót hagy maga után.
Mindenezek ellenére könnyen megeshet, hogy a jövőben az Enhydra segítségével fogok Javában fejleszteni az egyszerű Jakarta-Tomcat helyett, amelyet ez idáig használtam. Az XMLC és az összevont fejlesztőkörnyezet kettőse több szempontból is meglehetősen vonzó.
Mint korábban említettem, az egyik ok, ami miatt az Enhydra Enterprise a leginkább felkeltette a figyelmemet, az, hogy képes kapcsolódni a Sun Enterprise JavaBeanshez. A következő két hónapban közelebbről is megvizsgáljuk az Enhydrát, elsőként az XMLC-vel, majd a DODS eszközzel ismerkedünk, amely relációs adatbázisokat rendel Java-objektumokhoz. Ezt követően egy kicsit az EJB világába is bekukkantunk, hogy szemléltessük: bár nyílt forráskódú programokra alapozunk, eszközeink semmivel sem lesznek kevésbé hatékonyak, mint az üzleti programozókéi.

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/).