XMLC

Reuven az XMLC-vel, az Enhydra alkalmazáskiszolgáló egyik elemével ismertet meg bennünket.

Az utóbbi pár hónapban több módszert is megvizsgáltunk azzal kapcsolatban, hogyan lehetséges kiszolgálóoldali Java segítségével webalkalmazásokat készíteni. Egyszerű servletekkel kezdtünk, végül eljutottunk a JavaServer Pages-hez (JSP). Hogy JSP-nkből eltávolíthassuk a Java-kódot, nekiálltunk JavaBeanst használni, azokat a különleges objektumokat, melyek metódusai lapunkról automatikusan elérhetők lesznek.
Csak addig foglalkoztunk a JavaBeansszel, míg a testreszabott események elő nem kerültek (lásd Linuxvilág, 2001. augusztus, 67–70. oldal). Ezek – JSP-nkben XML-tagoknak és értékeknek álcázott események – egy-egy Java-osztály eljárásaihoz vannak kötve. Más szavakkal tagokat helyezve a JSP-be hatékonyan hívhatunk meg egy vagy több eljárást. A testreszabott tagok és a babok együttes használatával jókora adag Java-kódot távolíthatunk el JSP-programjainkból.
Mire jutottunk? Mint augusztusban láthattuk, a testreszabott eljárások okos használatával saját kis mininyelvet hozhatunk létre ciklusokkal, feltételekkel és változókkal. Saját tagok írásával mentesíthetjük a grafikus tervezőket a Java-használat alól, és jobban elkülöníthetjük a formát és a tartalmat. Gondjaink megoldásához azonban még ez sem elegendő.
Jó megoldás lehet az Enhydra alkalmazáskiszolgáló, amelyről az előző hónapban írtam (szeptemberi Linuxvilág 64. oldal). Az XMLC (vagy XML-fordító) az XML-fájlokat (ideértve a HTML- és XHTML-állományokat is) Java-objektumokká alakítja. Ha az ezekhez az objektumokhoz tartozó eljárásokat hívjuk meg, az éppen készülő HTML-t meg tudjuk változtatni.

XML és XHTML

Az XML – mint azt valószínűleg mindannyian tudjuk – egy bővíthető jelölőnyelv (markup language). Ami évekkel ezelőtt még egyszerű kis szabványnak indult, mára már hatalmas szabvány- és ajánlott szabványgyűjteménnyé nőtte ki magát. Az XML magja azonban mindvégig azonos maradt, így az emberek megegyező formai követelmények alapján készíthetik el a saját jelölőnyelvüket. Az XML nem közvetlen használatra készült; célja, hogy segítségével ki-ki a saját nyelvét készíthesse el. Mivel ezek a nyelvek mind XML-alapúak, jól érthető felépítésük van, amit minden XML-értelmező elfogad. Ráadásul, ha nyelvünkhöz adattípus-meghatározást is készítünk (DTD), az értelmező ellenőrizheti, hogy az elemek és értékek megfelelő formátumban vannak-e.
A HTML és az XML egyaránt a World Wide Web Konzorcium (W3C) szabványa, igen hasonló szerkezettel rendelkeznek, és gyakran veszik őket egy kalap alá. A valóság azonban az, hogy míg a HTML csak egy jelölőnyelv a sok közül, az XML saját jelölőnyelv megalkotására nyújt lehetőséget. Még jellemzőbb, hogy a HTML sokkal lazább felépítéssel rendelkezik, mint az XML, amit nem kis mértékben történelmi tényezőknek köszönhet. A következő kifejezés például helyes HTML-parancs: `<img src="foo.png">`.
Ezzel szemben az XML-alapú nyelvekben minden tagot szigorúan le kell zárnunk, ezért egy XML-dokumentumban az említett példa nem lenne szabályos. Helyesen a következőt kellene írnunk: `<img src="foo.png"/>`.
Hogy a HTML és az XML közti szakadékot áthidalja, a W3C egy XHTML-nek nevezett ajánlást adott ki, mely tulajdonképpen a HTML XML-átirata. Az XHTML használatának meglehetősen sok előnye létezik, melyek közül a legnagyobb talán, hogy HTML-dokumentumainkkal az összes XML-eszköz dolgozni tud.
Természetesen ez egyben azt is jelenti, hogy XHTML-dokumentumaink kicsit formálisabban fognak kinézni, mint a korábban megszokott HTML. Míg a HTML megengedi, hogy hanyagok legyünk, és mindössze egy `<P>`-vel válasszuk el a bekezdéseket, az XHTML sokkal szigorúbb: a bekezdéseket kötelező `<P>`-vel kezdeni és `</P>`-vel befejezni. Az értékeknek kötelezően macskakörmök közé kell kerülniük, amit sokan figyelmen kívül hagynak, amikor egyszerű HTML-ben dolgoznak. Bár a XHTML néha sok gondot okozhat az embernek, a programok terhelését nagymértékben csökkenti – felépítésüket ugyanis szabályosabbá teszi, így könnyebb lesz írni és olvasni őket.
A legnagyobb előny azonban mégis az a tény, hogy az XHTML-dokumentumokat XML-dokumentumnak minősíthetjük.

A DOM

Az XML-dokumentum egy faszerkezet, mely tény ismerősen csenghet azok számára, akik főiskolán tanultak számítástechnikát. A fákkal való munka elméletben figyelemreméltóan könnyű, a gyakorlatban viszont az alkalmazott csatolófelülettől függően időnként trükkösnek bizonyulhat.
Két népszerű felületfüggetlen API létezik az XML-hez: a SAX (Simple API for XML) programot úgy tervezték, hogy XML-adatfolyamokkal tudjon dolgozni, ezáltal kicsi és hatékony maradt. A DOM (Document Object Model) vele ellentétben a felhasználót egyszerre engedi hozzáférni a teljes dokumentumfához, miáltal lehetővé válik, hogy lerövidítsünk vagy megváltoztassunk csomópontokat, beleértve az új csomópontok felvételét és törlését is. Emellett ez azt is jelenti, hogy a teljes dokumentumot a memóriába kell tölteni, mielőtt a DOM segítségével dolgozni kezdhetnénk rajta. Ezek a képességek a SAX-nál hatékonyabbá, ám egyszersmind lassabbá és erőforrás-igényesebbé is teszik.
Az XMLC XML-fájlokat, rendszerint HTML-ben vagy XHTML-ben írt állományokat alakít át olyan Java-osztályokká, amelyek a DOM-fát létrehozzák, illetve módosítják. A megszokott DOM-eljárásokat használhatjuk csomópontok hozzáadására, törlésére és módosítására, megváltoztatva az éppen kimenetre kerülő dokumentumot.
Az XMLC igazán nagy ötlete a HTML `"id"`-értékek használata. Amikor az XMLC fordító egy `id`-értéket talál, készít egy eljárást, melynek segítségével lekérdezhetjük, illetve módosíthatjuk az adott értékben található szöveget. A laptervezők

tehát HTML-ben dolgozhatnak, és az egyes dinamikus szövegmezőket egyedi címkékkel jelölik meg. Amikor a tervezők elkészítették az eredeti HTML-lap vázlatát, egy Java-osztállyá fordítják le (`xmlc`-t használva). A fejlesztők ezután olyan servleteket készíthetnek, amelyek létrehozzák ezt az osztályt, az eljárások segítségével dinamikusan létrehozott tartalommal helyettesíthetik a vázlatszöveg mezőit, és a dokumentumot a felhasználó böngészőjére küldhetik ki.
Az alapötlet az, hogy a tervezőknek nem kell vegyesen szöveges és HTML formátumon dolgozniuk, hanem egyszerűen csak a végleges kimenet vázlatát készítik el. Amíg az `id`-értékek nem változnak meg, a HTML-fájl és a servlet fejlesztése párhuzamosan is folyhat, és sem a tervezőknek, sem a fejlesztőknek nem kell a társaikra várniuk.

Az Enhydra telepítése

Amint azt fentebb említettem, az `xmlc` az Enhydra-alkalmazás kiszolgálórésze. A 3.x-változatú Enhydra már termelésre alkalmasnak mondható, és tartalmaz egy `xmlc`-t, amelyet a legtöbb felhasználó több mint megfelelőnek fog találni. Mivel engem az Enhydrában most leginkább az érdekelt, miképpen tudnám az Enterprise JavaBeanshez (EJB) felhasználni, a 4.x próbaváltozattal dolgoztam, más néven az Enhydra Enterprise-változattal. Időközben valószínűleg az Enhydra Enterprise végső kiadása is elérhető, melyben a webfejlesztők egy nyílt forrású, termelési minőségű, J2EE-megfelelő alkalmazáskiszolgálóhoz juthatnak (még mindig nem jelent meg – a szerk.).
Az `xmlc`-vel való munkához letöltöttem az Enhydra Enterprise próbaváltozatot egy 15,7 MB méretű ***enhydra4.0.tar.gz*** nevű fájl képében. Utóbbit megnyitva gazdag könyvtárkészletet, alkalmazásokat, és leírást találunk az Enhydra alkalmazáskiszolgálóhoz, melyről előző számunkban már írtunk. Ezek nagy részét most figyelmen kívül fogjuk hagyni, és kizárólag az XMLC-re összpontosítunk.
Csaknem a teljes Enhydra héjprogramból hívott Java-osztályokból áll. Annak érdekében, hogy a héjprogramok megtalálják a Java-osztályokat, be kell őket állítani az adott telepítéshez, amit könnyen megtehetünk, ha belépünk az Enhydra könyvtárba (az én rendszeremen ***enhydra4.0***) majd lefuttatjuk a `configure` parancsfájlt:

```
./configure /usr/java/jdk1.3
```

A `configure` alapesetben egyetlen értéket vár: JDK 1.3 telepítésünk könyvtárának a teljes nevét. Bár az Enhydra korábbi változatai (különösen a korábbi Enhydra Enterprise-ok) nem működnek JDK 1.3 alatt, a jelenlegi változatok csak és kizárólag ezzel dolgoznak. Mivel a JDK 1.3 számos előnyös tulajdonsággal rendelkezik, és a Sun támogatja a linuxos változatot, nem rossz ötlet feltelepíteni.
Ha az Enhydrát a */usr/local/enhydra* könyvtáron kívüli helyre telepítettük, előfordulhat, hogy az `ENHYDRA` környezeti változót be kell állítanunk a telepítés könyvtárára.
Az XMLC teljes kihasználásához három különféle .JAR-fájlt kell `CLASSPATH`-unkba helyezni. Mivel a cikk további részeiben az `xmlc` használatán lesz a hangsúly, nem árt, ha most rögtön hozzáadjuk őket a `bash` formai követelményeit használva:

```
export CLASSPATH=$ENHYDRA/lib/xmlc.jar:\
$ENHYDRA/lib/enhydra.jar:\
$ENHYDRA/lib/xmlc-support.jar
```

A hozzám hasonlók bizonyára az Enhydrához kapcsolódókon kívül még számos elemet is a `CLASSPATH` változóban szeretnének tárolni. Nézzük meg, hogyan állítottam be én a `CLASSPATH` változómat!

```
export CLASSPATH=$ENHYDRA/lib/xmlc.jar:\
$ENHYDRA/lib/enhydra.jar:\
$ENHYDRA/lib/xmlc-support.jar:\
$TOMCAT_HOME/classes:\
$TOMCAT_HOME/lib/servlet.jar:\
/usr/share/pgsql/jdbc7.1-1.2.jar
```

Figyeljük meg, hogy rendszeremen az Enhydra .JAR-fájljai a többiek elé helyeztem, így elkerülhettem az ütközés miatt felbukkanó gondokat. Mivel néhány osztályból az Enhydra már a legújabb változattal rendelkezik, például amelyek a DOM-mal dolgoznak, ezeknek elöl kell lenniük.
Megjegyzem, az XMLC-vel való munka nem minden szakaszában szükséges az Enhydra mindhárom .JAR-állománya. Én azonban úgy gondoltam, kényelmesebb az összes lépésben mindet hozzáadni, így a későbbiekben elkerülhetem a kellemetlen meglepetéseket.

Egyszerű HTML-állomány

Most, hogy már minden, az `xmlc`-hez szükséges dolgot feltelepítettünk, próbáljuk is ki egy egyszerű HTML-fájlon:

```
<html>
    <head><title>This is a title</title></head>

    <body>
        <h1>This is a headline.</h1>
        <p id="firstpara">This is a
                          paragraph.</p>
        <img src="foo.gif"/>
        <p>This is a second paragraph.</p>
    </body>
</html>
```

Bár az `xmlc` egyszerű HTML-fájlokkal is éppen olyan jól dolgozik, az XHTML használata jobb ötlet, ugyanis megakadályozza, hogy olyan fájlokat készítsünk, amiket a DOM nem tud megjeleníteni. Az XML például tiltja az átfedő tagokat:

```
<i><p>Wow</i>, he thought.</p>
```

A fenti sor HTML-ben még éppen elfogadható, de tilos az XML-ben és az XHTML-ben. Így – bár a böngésző ezzel a HTML-sorral még valahogy elboldogul és értelmezni is képes – az `xmlc` figyelmeztetést fog küldeni arról, hogy eldobott egy használhatatlannak minősített bezáró tagot. Az `xmlc` gyakran fog figyelmeztetni, ha a HTML nem helyesen van megformázva, ezzel segítve a lehetséges hibák felderítését. Bár egyszerű HTML-ek írásakor nemigen kell a dokumentumunk szerkezetével foglalkoznunk, a módosítások, amelyeket az `xmlc` segítségével végre szeretnénk hajtani, a dokumentumok kiírási szabályainak világos ismeretét követelik meg.
Az előző példa első bekezdését a `"firstpara" id`-érték azonosította. Hamarosan meglátjuk, miképpen tudjuk ezt a szöveget Java-programból megváltoztatni, hivatkozási pontként használva az `id`-értéket.
Hogy a dokumentumot Java-osztállyá változtathassuk, meg kell hívnunk az `xmlc` programot. Feltételezve, hogy a fenti HTML-fájlt ***foo.html***-nek neveztük el, írhatjuk be a következőt:

1. lista. A PrintFoo.java – rövid parancssoros program, amely módosítja a „firstpara” id-vel jelölt szöveget, majd a dokumentumtartalmat a kimenetre írja

```
**// Az EnhydrÆval Ørkezı DOM-osztÆlyok
**// bet ltØse
import org.w3c.dom.html.*;

public class PrintFoo {

    // Csak egy eljÆrÆst hatÆrozunk meg,
    // amely a parancssorb l lesz megh vva.
    public static void main (String[] args)
{

        // a "foo" lap objektumunk
        // pØldÆnyÆnak elkØsz tØse.
        foo myfoo = new foo();

        // a "firstpara" bekezdØs nk
        // sz vegØnek megvÆltoztatÆsa.
        myfoo.setTextFirstpara
        ("This has been changed");

        // Az eredmØny megjelen tØse
        System.out.print(myfoo.toDocument());
    }
}*
```

```
$ENHYDRA/bin/xmlc -parseinfo -verbose -keep
➥foo.html
```

Ezáltal a foo.html ***foo.java*** Java-forrásfájllá alakul, amelyből fordítás után létrejön a ***foo.class***. A `-keep` érték megtartja a foo.java fájlt ahelyett, hogy a foo.class elkészítése fordítás után letörölné. Bár szükségtelenek, én a `-parseinfo` és `-verbose` értékeket is szeretem használni, amikor az `xmlc`-vel dolgozom, mivel így némi visszajelzést kapok a fordítás menetéről.
A Java `xmlc` által készített forráskód meglehetősen hosszú és unalmas, bár megjegyzésekkel bőven el van látva. Azok számára, akik módosítani szeretnék a foo.html-t, a foo.java legfontosabb része a `getElementFirstpara()` és a `setTextFirstpara()` eljárás. Az első visszaadja a `"firstpara"` által azonosított szöveget, az utóbbi pedig lehetővé teszi, hogy valamilyen tetszőleges karaktersorozatra cseréljük le.
Az ***1. lista*** egy rövid parancssoros Java-osztály (***PrintFoo.java***) forráskódját tartalmazza, amely kiírja a „javásított” foo.html-változat tartalmát. Mielőtt azonban kiírná, a `setTextFirstpara()` segítségével megváltoztatja a kimenetet:

```
myfoo.setTextFirstpara("This has been
changed");
```

Ezután a változtatás után már megjeleníthetjük a szöveget:

```
System.out.print(myfoo.toDocument());
```

A DOM-fát magunk is átalakíthatjuk, ha kikeressük az adott `id`-vel rendelkező csomópontokat, majd kézzel megváltoztatjuk őket. Az `xmlc` kényelmes eljárásai azonban az ilyen szövegek módosítását különlegesen egyszerűvé és magától értetődővé teszik.
Ha most lefuttatjuk a `PrintFoo`-t, megfigyelhetjük, hogy HTML-kimenet az eredeti szóközök nélkül jelenik meg. Az eredménydokumentum ugyan nehezebben olvasható az emberi szem számára, a böngészőknek azonosnak számít. A magam részéről mindig igyekeztem, hogy HTML-dokumentumaim a könnyebb hibakeresés érdekében szépen formázottak legyenek, ezért jó néven venném az `xmlc`-től, ha egy `-preserve-whitespace` (őrizd meg a szóközöket) kapcsolót is támogatna.
Abból, amit eddig láttunk, kiderült, hogy az `xmlc`-vel egyszerű egy teljes bekezdést megváltoztatni, de nehéz módosítani egyetlen szót. Az `xmlc` azonban kihasználja a HTML `"span"`-tag előnyeit, amely elfogadja az `id`-értéket, és lehetővé teszi, hogy azonosítóval lássunk el módosítani kívánt egyedi szavakat, karaktereket vagy képeket, például:

```
<P id="para">This is a paragraph,
  <span id="phrase">and this is a phrase</span>.
</P>
```

Miután az `xmlc`-vel ezt a HTML-t lefordítottuk, a `SetTextPara()` eljárással az egész bekezdés tartalmát megváltoztathatjuk, a `SetTextPhrase()`-zel pedig a `span`-tagok közötti részt módosíthatjuk.

Servletek

Most már látjuk, miképpen lehet az `xmlc`-vel parancssorból dolgozni, nézzünk meg hát egy servletet is, amely ugyanezt a feladatot látja el. A kezdők kedvéért: az egyszerű PrintFooServlet egy HTTP-kérelmet fog kapni, eredményül pedig a HTML-dokumentumot fogja visszaadni.
A ***2. lista*** tartalmazza a foo.html-t megjelenítő servlet kódját. Akárcsak parancssoros testvére, `"foo"` osztályunkból létrehoz egy példányt, megváltoztat benne néhány szöveget, végül az XML-fa szöveges megjelenését a kimeneti folyamra írja. Ebben az esetben a kimeneti folyam éppen a felhasználó böngészőjéhez kerül. A felhasználó így a módosított vázlatot látja, anélkül, hogy tudná, két Java-osztály (és az eredeti HTML-dokumentum) is részese volt a folyamatnak.
A servlet működéséhez a foo.class egy másolatát kellett behelyeznem a Jakarta-Tomcat servletmotor `CLASSPATH` környezeti változója által meghatározott egyik könyvtárba. Úgy döntöttem, hogy a legfelső szintű ***$TOMCAT/classes*** könyvtárba teszem. Ha egy termelési osztály lenne, nyilván értelmesebb helyre raktam volna, kihasználva a Java egymásra épülő névterét (namespace). Igaz, már az `xmlc`-t is a csomagok meghatározása nélkül futtattam le, ami azt jelenti, hogy a foo.class-t mindenképpen a legfelső szintű névtérben kell elhelyezni, amihez az ***il.co.lerner*** névtérben a `-class` lehetőséget kellett volna használnom:

```
$ENHYDRA/bin/xmlc -class il.co.lerner.foo\
-parseinfo -verbose -keep foo.html
```

A ***$TOMCAT/classes***-ben található foo.class segítségével a PrintFooServlet.java fájlt sikeresen le tudtam fordítani. Immár az egyetlen kihívás a servlet végrehajtása, továbbá a megváltozott HTML-lap megjelenítése maradt. Még egyszer hangsúlyozom: meg kellett ugyan változtatnom a `CLASSPATH`-t, de ebben az esetben a megváltoztatandó `CLASSPATH` a Tomcat servletmotoré volt, amely számunkra a servleteket végrehajtja.

2. lista. PrintFooServlet.java, a PrintFoo.java servletváltozata

```
**// PrintFooServlet servletvÆltozat
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Az EnhydrÆval Ørkezı DOM-osztÆlyok
// bet ltØse
import org.w3c.dom.html.*;

// a "foo" osztÆly bet ltØse
import il.co.lerner.*;

public class PrintFooServlet extends
        HttpServlet {

   public void doGet(HttpServletRequest
   request,
        HttpServletResponse response)
        throws IOException, ServletException
   {
      // a MIME tartalomt pus beÆll tÆsa
      // a vÆlaszhoz
      response.setContentType("text/html");

      // a kimeneti folyam beÆll tÆsa
      // STDOUT-ra
      PrintWriter out = response.getWriter();

      // a "foo" lapobjektumunk pØldÆnyÆnak
      // elkØsz tØse.
      foo myfoo = new foo();

      // a "firstpara" bekezdØs nk
      // sz vegØnek megvÆltoztatÆsa.
      myfoo.setTextFirstpara
      ("This has been changed");

      // Az eredmØny megjelen tØse
      out.println(myfoo.toDocument());
   }
}*s
```

Megváltoztattam a *$TOMCAT/bin/tomcat.sh* fájlt, mely mielőtt éppen exportálta volna a `CLASSPATH`-t, hozzá kellett adnom a három Enhydra .JAR állományt, majd újraindítanom a Tomcatet. Pár pillanattal később böngészőmmel a servletre mutatva elégedetten figyelhettem meg képernyőmön az eredeti HTML-fájl megváltoztatott formáját.

Adatbázisok és az XMLC

Könnyű belátni, hogyan lehetne a lapot relációs adatbázisból származó adatokkal benépesíteni: például itt van egy kicsi PostgreSQL-tábla, amelyben az egyes naptári napokhoz tartozó mondásokat (`saying`) tárolhatjuk:

```
CREATE TABLE DailySayings (
    date     TIMESTAMP    NOT NULL,
    saying   TEXT         NOT NULL,
    UNIQUE(date)
)
```

Most szúrjunk be pár mondást a táblánkba:

```
INSERT INTO DailySayings(date, saying)
VALUES (CURRENT_DATE,
·A bird in the hand is worth two in the
bush.·);
INSERT INTO DailySayings(date, saying)
VALUES (CURRENT_DATE+1,
·A penny saved is a penny earned.·);
INSERT INTO DailySayings(date, saying)
VALUES (CURRENT_DATE+2,
·The rain in Spain falls mainly in the
plain.·);
```

A mai mondás lekérdezéséhez mindössze a következőre van szükségünk:

```
SELECT saying
FROM DailySayings
WHERE date = CURRENT_DATE
```

Ha olyan servletet szeretnénk írni, amely megjeleníti a mai mondásokat, két osztályra lesz szükségünk: egy mintára, amit az `xmlc`-vel készítünk el, (illetve a ***saying.html***-re, amiből a ***saying.class***-t fordítjuk), és egy másikra, amely betölti és módosítja a mintát (***DailySaying.java***). XMLC-dokumentumunk és a módosítóosztály fejlesztése közben megegyezünk abban, hogy a `"saying" id` fogja a kettőt összekapcsolni.

XMLC-dokumentumunk eléggé magától értetődő:

```
<html>
   <head><title>Today·s saying</title></head>

   <body>
      <h1>Today·s saying</h1>

      <p>And now, as you requested, today·s
      saying:
        <span id="saying">Saying Goes
        Here</span>.</p>
   </body>
</html>
```

Ezt a HTML-dokumentumot az ***il.co.lerner.saying*** Java-osztállyá fordítottam le, a móka kedvéért a .java fájlt is megtartva:

```
$ENHYDRA/bin/xmlc -class il.co.lerner.saying\
-parseinfo -verbose -keep saying.html
```

Ezután az eredményül kapott saying.class fájlt bemásoltam a ***$TOMCAT_HOME/classes/il/co/lerner*** könyvtárba, ahol egyébként a servletekkel kapcsolatos osztályaimat tartom. Miután telepítettem a dokumentumot, meg kellett írnom a módosítóosztályt, ami végrehajtja a fent bemutatott SQL-lekérdezést, letölti az eredményeket és beilleszti a lefordított XMLC-dokumentumba. A 3. lista (20. CD-mellékletünkön találhatják meg) tartalmazza a servletünkhöz tartozó forráskódot, amit lefordítás után a Tomcat kiszolgáló működő

Kapcsolódó címek

Amint már említést nyert, az XMLC a Lutris Enhydra alkalmazáskiszolgáló része, amely teljes mértékben együttműködő próbál maradni a J2EE-vel. Az Enhydra 4, amely nem sokkal e cikk megírása előtt lépett a próbaváltozat állapotába, az Enterprise JavaBeansnek és más magas szintű programrendszernek az Enhydrába történő bevezetését célozta meg. Többet is megtudhatunk az Enhydráról, és a 3.x vagy 4.x sorozat legújabb változatait tölthetjük le a ➲ http://www.enhydra.org/ címről.

Az Enhydráról általánosságban szóló, néhány XMLC-példával megtűzdelt írás *Roger Metcalf*-nak az ArsDigita Systems Journalban található cikke, lásd: ➲ http://www.arsdigita.com/asj/enhydra/.

Ugyancsak kitűnő XMLC-segédanyag található a Weben, a ➲ http://www.plugged.net.au/publications/xmlc-tutorial/urls.html címen.

A W3C (➲ http://www.w3.org/TR/2000/REC-xhtml1-20000126/) honlapján érhető el az XHTML-t bemutató dokumentumuk, amely figyelemre méltóan jól olvasható ismertetés az XHTML-ről. Csak azt írja le, ami szabályos (és ami nem). Bár meglehet, hogy XMLC-alkalmazásainkban nem akarunk szigorú XHTML-t használni, azért nem árt, ha megismerjük.

Az Apache Jakarta-Tomcat JSP és a servletmotorról szóló tájékoztató a ➲ http://jakarta.apache.org/ címen érhető el.

Végül két egymást kiegészítő *O'Reilly*-könyv elég ismerettel szolgálhat, hogy az XMLC-vel dolgozni kezdjünk. A *Brett McLaughlin* (most a Lutrisnál dolgozik) által írt „Java and XML" részletesen tárgyalja a DOM-ot, összehasonlítja a SAX-szal és más XML-értelmező lehetőségekkel. A „Java Servlet Programming" második kiadása *Jason Hunter*-től és *William Crawford*-tól egy teljes fejezetet szentel az XMLC-nek, rögtön a hasonló, de versenytársként szereplő kiszolgálóoldali Javát alkalmazó dinamikus lapelőállító rendszerek fejezete után.

servlettárolójába (active servlet context) helyeztem. A Tomcat és az Apache újraindítása után böngészőmön keresztül már hozzá tudtam férni a mai szóláshoz a HTML-dokumentumba helyezett SQL-eredmények segítségével.

Jó megoldás az XMLC?

Amikor először kezdtem foglalkozni az XMLC-vel, komoly kétségeim voltak a használhatóságával kapcsolatban. Több év vegyes sablonon alapuló munka után egyszerűen túl vadnak látszott a HTML-fájlok Java-osztállyá változtatása csak azért, hogy a DOM segítségével kezelhessük őket. És hát egy DOM-értelmezőt elindítani valóban sokkal több erőforrást igényel, mint egyszerűen megjeleníteni a fájlt.
Ahogyan azonban egyre többet dolgoztam már az XMLC-vel, fokozatosan felfigyeltem a sablonokkal szembeni előnyeire. Az XMLC lényegében rákényszeríti a tervezőket és a fejlesztőket, hogy megállapodást kössenek, vagy API-leírást alkossanak dokumentumaik és a program között. Ha egyszer ez az API elkészült, nem lehet egykönnyen megváltoztatni, ami nem feltétlenül rossz dolog. A legfontosabb, hogy a fejlesztők és a tervezők közötti API üzembiztossága lehetővé teszi számukra az egymás tevékenységét szinte egyáltalán nem zavaró párhuzamos munkát.
Mivel a Java-módosító osztály a lefordított dokumentum HTML-tartalmát bármilyen módon meg tudja változtatni, könnyen elképzelhetünk egy olyan helyzetet, amelyben egyszerre három osztállyal dolgozik: egy fejlécfájllal, a dokumentum testével és a láblécfájllal. Az osztályunk ezek után a fejlécet a DOM-eljárások segítségével hozzácsatolhatja a dokumentum elejéhez, a láblécet pedig a végéhez. Ezzel a módszerrel általános formátumot tudunk adni a honlapnak anélkül, hogy ugyanazt a szöveget kellene minden fájl elejére másolnunk. Természetesen számos bosszantó részlet is akad a XMLC-vel való munka során. Hamar unalmassá válik például, hogy minden HTML-fájlhoz servletet kell írni. Igaz, egyetlen servletet is írhatnánk, amely a fájlnevet egy lekérdezésből kapja meg, és tulajdonképpen majdnem úgy működik, mint egy XMLC által készített különböző osztályokhoz tartozó dokumentumsablon. Meglehet, hogy egyszerűen csak nem néztem át elég tüzetesen az Enhydrát ahhoz, hogy megtaláljam e kérdésre a választ, vagy az is lehet, hogy az Enhydra-fejlesztők gyorsan hozzászoknak ahhoz, hogy minden megjeleníteni kívánt laphoz két Java-osztályt kell készíteniük. Mindenesetre ezáltal igen rövid idő alatt óriási számú osztály jöhet létre, még kicsi vagy közepes honlap esetén is.
A legnagyobb gond az XMLC-vel kapcsolatban szerintem a magas szintű HTML (és XML – a pontosság kedvéért) módosító API hiánya. Az XMLC egyik GYK-ja a „Hogyan adhatok egy sort egy HTML-táblához?" Egy ilyen, a HTML-ben magától értetődő feladat hamar terhessé válhat az XMLC-vel. Először is meg kell találni annak a táblának az alját, ahová a sort be szeretnénk szúrni, majd különleges csomópontokat (és értékeket) kell hozzáadni a csomóponthoz. Ez igencsak nem-HTML „szagú" és rákényszeríti a fejlesztőt, hogy csomópontokban gondolkodjon, amikor ő HTML-ben szeretne gondolkodni. Annak alapján, hogy az Enhydra egy Java-eljárások segítségével történő SQL-lekérdezés szerkesztési lehetőséget tartalmaz, úgy gondolom, egy ehhez hasonló HTML-módosító API nem lenne túl bonyolult.

Következtetés

Az XMLC igen fondorlatos, az Enhydra alkalmazáskiszolgáló szívében helyezkedik el. Az XMLC rákényszeríti a fejlesztőket (és a tervezőket), hogy mielőtt dolgozni kezdenének, megtervezzék, hogyan fognak együttműködni, majd lehetővé teszi, hogy egymástól függetlenül dolgozzanak. Bár az ilyesfajta művelet kibillentheti egyensúlyukból a tapasztalt sablonhasználókat, hamarabb válik természetessé, mint az ember valaha is gondolná. Az igazság az, hogy a Zope ZPT-je is hasonló módszert alkalmaz az űrlaptartalom szétválasztására, ami talán a webfejlesztő közösség jelenlegi irányvonalát mutatja. A közeljövőben várhatóan további XMLC-szerű rendszerekkel fogunk találkozni. Ha szerencsénk van, lehet, hogy valamiféle szabványosítás is lezajlik a minták közt, így a tervezők könnyen mozoghatnának a különféle rendszerek között a köztük lévő apró különbségek elsajátítása nélkül.

Reuven M. Lerner (reuven@lerner.co.il) egy kis webes-internetes tanácsadó cég tulajdonosa. Feleségével, Shirával és lányával, Atara Margalittal a „jövő városában", az izraeli Modi'inben él.
➲ http://www.lerner.co.il/atf/