ASP és Java Servlet technológia összehasonlítása

Viczián István

Szerzőről
Viczián István a Debreceni Egyetem programtervező matematikus szakán végzett 2001 nyarán, ahol most levelező PhD. hallgató az elosztott rendszerek, middleware-ek témakörében. Jelenleg szoftver fejlesztőként dolgozik Budapesten, melynek keretében middleware szoftverekkel, alkalmazásintegrációval, webes technológiákkal valamint application mining-gal és reverse enginering-gel foglalkozik. Szabadidejében internetes portált fejleszt, és Java technológiákkal ismerkedik, valamint Java blog-ot ír (JTechLog).
E-mail: viczus@freemail.hu
Honlap: http://dragon.unideb.hu/~vicziani

Budapest, 2002

Bevezetés

Az Internet széleskörű elterjedésével, a technológiák fejlődésével, a hagyományos vállalatok Interneten való megjelenésével egyre nő az igény, hogy ne csak statikus HTML lapok jelenjenek meg a Világhálón, hanem dinamikusan, akár valós időben generált oldalak, melyek az adatokat már régóta használt adatbázis-kezelő rendszerekből olvassák, és azokat megfelelő formába öntsék. Ennek megoldására több technológia is született, melyek mára teljesen kiforrottak, ipari szinten is megállják a helyüket. Ezek közül a Microsoft ASP és a Java Servlet technológiát szeretném bemutatni és összehasonlítani dolgozatomban.

Szerver oldali technológiák

Mind az ASP mind a Servlet technológia szerver oldali, ami azt jelenti, hogy a HTML oldalak a szerver oldalon generálódnak, és a kliens oldalra a böngészőnek már csak ezen oldalak jutnak el. Előnye, hogy a böngészőnek nem kell semmilyen script nyelvet támogatnia, és az algoritmusok is a szerver oldalon futnak, így jelentős terhet levesz a szerver a kliensek válláról. Persze a biztonsági kérdések is jobban kezelhetőek így, mivel nem a kliensnek kell közvetlenül vagy áttételesen az adatbázishoz kapcsolódnia, hanem az ASP-motornak. Ellenérvként az hozható fel ellene, hogy mivel minden a szerver oldalon történik, jelentősen megterheli azt.

Feladat

Mindkét esetben a feladat az adatbázisban található adatok olvasható formában történő listázása, módosítása. A lapok egy komplexebb portál részét képezték, feladatom ezen oldalak részfunkcióinak megvalósítása. A felhasználó regisztrációját, bejelentkezését, nyomkövetését és kijelentkezését is kezelnem kellett, és ellenőrizni a hozzáférési szinteket.

ASP technológia

Az ASP (Active Server Pages) technológiát a Microsoft fejlesztette ki, és támogatja az IIS web szerverében, bár már több alternatíva is létezik. Én nem használtam ki ezeket az előnyöket, HTML fájlokba egyszerűen, szövegszerkesztő segítségével illesztettem be az ASP kódot. Az ASP tulajdonképpen azt biztosítja, hogy különböző script nyelveken (Visual Basic Script és JScript –mindkettő Microsoft által fejlesztett objektumorientált script nyelv) megírt kódrészeket illesztünk HTML oldalakba, akár keverve a script utasításait és a HTML elemeket. Ezen script nyelvek rendelkeznek minden olyan tulajdonsággal, ami a magas szintű objektumorientált nyelveket jellemzik, bár kicsit speciálisabbak és kötetlenebbek is. Az ASP script-ek szerver oldalon futnak le.
Implementáláshoz a Visual Basic Script nyelvet választottam.

Az ASP továbbfejlesztett változata az ASP.NET, mely szerves része a .NET technológiának, mely már sokkal egyszerűbb fejlesztést tesz lehetővé a Microsoft Visual Studio segítségével, melynek legújabb verziójának megjelenése hamarosan várható, a béta stádium végén jár. Ennek segítségével már készíthetőek úgy HTML oldalak, mint a hagyományos Windows-os alkalmazások, annak eseményvezérelt modelljére alapozva.

Servlet technológia

A Java dinamikus szerver oldali HTML generálásra való technológiája. Jelentős eltérés az ASP technológiától, hogy itt a programkódot nem HTML oldalakba kell beilleszteni, hanem egy külön alkalmazást kell írni, aminek kimenete a HTML fájl. A web szerver (pontosabban a

JServ) tulajdonképpen továbbítja a kérést a megfelelő servlet-nek, majd annak HTML formátumú válaszát visszaadja a böngészőnek. A Java-ban már van technológia a HTML kódba illeszthető Java alapú script-re, JSP néven. Ennek ismertetése azonban sajnos nem témája a dolgozatomnak.

Környezet

A fejlesztési környezet Microsoft Windows 2000 Professional operációs rendszeren futott. Az adatbázis-kezelő rendszer a Microsoft Access volt, mely a Microsoft Office XP programcsomag tagja. Az adatbázishoz mindkét esetben ODBC driver-en keresztül csatlakoztak az alkalmazások, a servlet-ek esetében szükséges volt egy ODBC-JDBC fordító driver-re is, hogy a beépített JDBC API (Java DataBase Connectivity Application Programming Interface) objektumait és metódusait használni tudjam. A web szerver az ASP esetében Microsoft Internet Information Server (IIS) volt, melyet a Windows 2000 alapkiépítésben tartalmaz, míg a servlet-ek esetében az ingyenes Apache szoftver szolgálta ki a HTTP kéréseket, és az abba beépülő JServ modul, mely a servlet-ek példányosítását, futtatását és megszüntetését végzi. Böngészőként Microsoft Internet Explorer-t használtam, de mivel a generált oldalak általában statikus HTML lapok voltak, ennek a célnak bármelyik piacon lévő, akár régebbi verziójú böngésző megfelel.

HTTP protokoll

Szintaxis

ASP

Az ASP fájl nem más, mint egy HTML fájl, mely tartalmazhat szöveget, HTML és XML elemeket valamint script-eket. Az ASP technológián belül alapesetben használhatunk VBScript-et, esetleg JScript-et is. A HTML kódba ezt a következőképpen kell beilleszteni:

```
<%
response.write(„Hello World!”)
%>
```

Persze illeszthetünk be egyéb nyelvű script-eket is, csak akkor a megfelelő script-motornak installálva kell lennie a számítógépen. Mivel a script-ek a szerver oldalon futnak le, a böngészőnek nem kell támogatnia a script-eket.

Java Servlet

A servlet-ek valójában speciális Java osztályok, amik a `Servlet`, speciálisan a `HTTPServlet` osztályt terjesztik ki. Egyéb megszorítás nem vonatkozik rájuk.

Válasz egy kérésre

ASP működése

Ha egy böngésző egy ASP fájlt kér le, az web szerver átadja a kérést az ASP-motornak, mely beolvassa az ASP fájlt soronként, és futtatja a benne elhelyezkedő script-et. Végül az eredményt visszajuttatja a böngészőnek, mint szabványos HTML fájl.

Java servlet-ek életciklusa

Egy servlet életciklusa alatt annak példányosítását, kliensek kiszolgálását és megszüntetését értjük. A servlet példányosításának ideje nem definiált, a web szerver indítása és az első kérés érkezése között bármikor megtörténhet (implementációfüggő). A servlet példányainak száma

sem meghatározott, általában annyi, amennyi virtuális néven az adott web szerver elérhető, azaz az esetek többségében egy. A példány létrehozása után a szerver meghívja annak `init` metódusát. Ezek után a servlet már kiszolgálhatja a bejövő kéréseket. A metódustól (GET, POST) megfelelően a következő metódusokat kell felüldefiniálnunk:

- `doGet(HttpServletRequest, HttpServletResponse)`
- `doPost(HttpServletRequest, HttpServletResponse)`

Gyakori megoldás, hogy az egyik metódust implementálják, míg a másik hívja ezt.
A `HttpServletRequest` a HTTP kérést reprezentálja, míg a `HttpServletResponse` a választ.
A metódusok általában lekérik a kérés tulajdonságait, paramétereit, feldolgozzák azt, majd elkérik a válasz `PrintWriter`-jét, és abba írnak vissza egy szabályos HTML formátumú adatfolyamot., amit a böngésző a kliens oldalon megkap. Mivel egyszerre több kérés is befuthat, gondoskodnunk kell arról, hogy ne okozzon gondot, ha a kiszolgáló metódust egyszerre több programszál hívja meg.
Amikor a web szerver úgy dönt, hogy nincs szükség többé az adott servlet-re (tipikusan leállításkor), meghívja annak `destroy` metódusát.
Dolgozatomban csak a HTTP protokoll felett működő servlet-ekkel foglalkozom.

Adatbázis hozzáférés

Az adatbázis-kezelő alkalmazás mindkét esetben Microsoft Access volt, a benne létrehozott táblákhoz ODBC driver-en keresztül fértek hozzá az alkalmazások. A fejlesztés során csak szabvány SQL lekérdezéseket használtam, semmi adatbázis-kezelő specifikus kiegészítést. Éppen ezért az adatbázis-kezelő alkalmazás lecserélése elképzelhető bármily másra, az a fontos, hogy legyen hozzá ODBC-driver, ami a ma használt rendszerek esetén alapkövetelmény. Az alkalmazásak kipróbáltan működnek Oracle9i, Microsoft SQL, PostgreSQL, MySQL adatbázis-kezelő alkalmazásokkal, csupán a táblákat kell átvinni. Ezért szabvány SQL adattípusokat használtam a táblák létrehozásakor. A servlet-ek futtatásakor szükség van még a JDBC-ODBC fordító driver-re, mely a Java API-ban specifikált JDBC hívásokat konvertálja át ODBC hívásokká ugyanúgy, ahogy adat transzformációt is végez. A Java eredeti elképzelésének (maximális platformfüggetlenségbe) ez megfelel, így ugyanis a kód változtatása nélkül kicserélhetjük az alatta futó adatbázis-kezelő alkalmazást, csak legyen hozzá JDBC driver, és az JDBC-ODBC fordítónak hála elég az ODBC driver. Persze használhatunk gyártó specifikus hívásokat is a driver-en keresztül, csak akkor sérül a hordozhatóság.

Mindkét esetben kétrétegű adatbázis-elérési modellt használtam, azaz az alkalmazások közvetlenül az adatbázis-kezelő rendszerrel kommunikálnak a driver-en keresztül. Persze az adatbázis-kezelő rendszer más gépen is elhelyezkedhet, mint ahol a web szerver. Ekkor a kommunikáció hálózaton keresztül történik. Mivel a feladat nem tartalmazott bonyolult üzleti logikát, mags szintű tranzakciókat, és rendkívüli terhelésre sem kell számítani, mindkét esetben kihagytam egy köztes réteg implementálását, mely a kapcsolatokat kezelné, optimalizálná. Rendkívüli biztonsági szintek bevezetésére sem volt szükség.

ASP megoldás ADO-val

A VBScript-ek a Microsoft ActiveX Data Objects (ADO) technológián keresztül kapcsolódtak az adatbázishoz, esetemben az ODBC driver-hez, mely rendszeradatforrásként lett deklarálva a Windows rendszerben. Az ADO egy Microsoft Active-X komponens. Az IIS installálásakor az ADO is automatikusan elérhető lesz a gépünkön.

Az adatbázis elérése a következő lépésekből áll:

- ADO kapcsolat létrehozása
- A kapcsolat megnyitása
- ADO recordset létrehozása
- Recordset megnyitása
- Az adatok kinyerése a kapott recordset-ből
- Recordset bezárása
- Kapcsolat bezárása

A kapcsolat létrehozását és megnyitását a következő ASP kód (VB Script) végzi:

```
<%
  Set conn = Server.CreateObject("ADODB.Connection")
  conn.open "BOO" , "user", "pswd"
%>
```

Ebben az esetben a BOO nevű ODBC driver-t használjuk, és a user felhasználóként lépünk be, harmadik paraméterként a jelszót adjuk meg.

SQL lekérdezést használva következőképpen kérhetünk le adatokat a Members táblából:

```
<%
  Set rsMembers = Server.CreateObject("ADODB.Recordset")
  strMembersSql = "select FirstName,LastName from Members"
  rsMembers.Open strMembersSql, conn
%>
```

Ekkor létrehozunk egy ADO recordset-et, illetve egy string-et, mely az SQL lekérdezést tartalmazza. A recordset-et megnyitni az SQL script string reprezentációja és a kapcsolat paraméterként való megadásával kell.

Az adatokat a recordset-ből a következőképpen kell kinyerni:

```
<%
  Do While Not rsMembers.EOF
    response.Write(rsMemberNames("FirstName") & " " & rsMemberNames("LastName") +_
    "<br>")
  rsMembers.MoveNext
%>
```

Ekkor a script végiglépked a recordset összes elemén, amely a lekérdezés eredményét tartalmazza, és kiírja a megfelelő oszlopokhoz tartozó értékeket.

Végül le kell zárni mind a recordset-et, mind az adatbázis kapcsolatot:

```
<%
  rsMember.close
  conn.close
%>
```

Servlet megoldás JDBC-vel

Először a JDBC-ODBC fordító driver-t kell betölteni közvetlenül a `Class.forName()` metódussal:

```
try {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (ClassNotFoundException e) {
  System.out.println("Error loading JDBC-ODBC Bridge.");
}
```

Ezután szintén egy kapcsolatot kell felépíteni:

```
String url = "jdbc:odbc:Telephone";
Connection con = DriverManager.getConnection(url);
```

Ezután létrehozunk egy `Statement` objektumot, mely a lekérdezést tartalmazza, majd a lekérdezés eredményét egy `RecordSet` objektumba kapjuk vissza:

```
String query = "SELECT DISTINCT * FROM Bejelentkezett";
Statement statement = con.createStatement();
ResultSet rs = statement.executeQuery(query);
```

Abban az esetben, ha egy paraméterezett lekérdezést szeretnénk megadni, akkor használjuk a `PreparedStatement` objektumot, és akkor az objektumok konverzióját az objektum automatikusan elvégzi:

```
String query = "SELECT Nev FROM Kedvencek WHERE Tulajdonos = ?";
PreparedStatement statement = con.prepareStatement(query);
statement.setString(1, username);
ResultSet rs = statement.executeQuery();
```

Ha a táblát módosítani akarjuk, akkor használjuk a `Statement` vagy `PreparedStatement` objektum `executeUpdate` metódusát.

Lépkedjünk végig a ResultSet elemein, és írjuk ki egy kiviteli csatornába:

```
while (rs.next()) {
out.println(rs.getString("nev") + ":" + rs.getString("ip"));
}
```

Majd zárjuk be a kapcsolatot:

```
con.close();
```

Form-ok kezelése

A szerver oldali technológiák során gyakran van szükség arra, hogy egy HTML form-on lévő adatokat, amiket a felhasználó írt be, fel kell dolgozni. A form adatai kétféleképpen juthatnak el a szerverhez. Ez a két metódus a GET és a POST. A GET metódus esetén a form komponenseinek nevei és értékei az URL-ben megfelelően kódolva jutnak el a szerver oldalra.

```
<form method="get" action="pg.asp">
First Name:  <input type="text" name="fname"><br>
Last Name: <input type="text" name="lname"><br>
<input type="submit" value="Send">
</form>
```

E form esetén, ha a felhasználó megnyomja a „Submit” gombot, az URL a következőképpen alakul:

```
http://www.boo.com/pg.asp?fname=John&lname=Smith
```

A POST metódus esetén az adatok „láthatatlanul”, a HTTP kérés fejlécében adódnak át.

ASP megoldás

A GET és POST metódus esetén az ASP fájlban két külön parancsot kell alkalmaznunk a form adatainak lekérdezésekor. GET metódus esetén ez a következőképpen alakul:

```
<body>
Welcome
<%
response.write(request.querystring("fname"))
response.write(" ")
response.write(request.querystring("lname"))
response.write("!")
%>
</body>
```

Aminek eredménye a böngészőben:
Welcome John Smith!

A POST metódus esetén a form parancsot kell használni:

```
<body>
Welcome
<%
response.write(request.form("fname"))
response.write(" ")
response.write(request.form("lname"))
response.write("!")
%>
</body>
```

Melynek az eredménye ugyanaz lesz, mint az előző script-nek, ha a form GET metódusát POST-ra cseréljük.

Servlet megoldás

Mindkét esetben a `HTTPServletRequest getParameter` metódusát kell használni. Azaz a kódrészlet mindkét esetben:

```
out.println(request.getParameter(„fname”)+” ”+ request.getParameter(„lname”)+”!”);
```

Ahol a `request` a HTTP kérésnek megfelelő osztály példánya, az `out` pedig az a `PrintWriter`, melyet a választ reprezentáló osztály példányától kértünk el.

Cookie-k

A cookie-k azonos kliens két különböző kapcsolat közötti adatok megőrzésére használatosak. A legtöbb böngésző támogatja, bár engedélyezésük kikapcsolható. Ezek szöveges információk, amit a kliens oldali böngészőprogram tárol. A szerverrel való kapcsolatfelvétel során a kliens program automatikusan küldi a cookie-kat is a szervernek, így az elemezni tudja őket. Persze a szerver a válaszban utasíthatja a böngészőt, hogy tároljon egy újabb cookie-t, vagy módosítson egy már meglévőt. Így őrizhető meg az információ két kapcsolat között. A cookie-k szintén a HTTP fejlécben utaznak észrevétlenül, ebből is látszik, hogy a cookie-k csak HTTP protokoll felett működnek. A cookie-knak több tulajdonsága is van: név, érték, élettartam, megjegyzés, verzió és biztonság. Ebből az első kettőt támogatja általában minden böngésző.

ASP megoldás

Egy új cookie létrehozása:

```
<%
Response.Cookies("firstname")="Smith"
%>
```

Cookie tulajdonságainak állítása:

```
Response.Cookies("firstname").Expires="May 10,2002"
```

Megjegyzendő, hogy az előző parancsokat a `<HTML>` tag előtt lévő script-ben kell kiadni, ugyanis a cookie-k a HTML fejlécben kerülnek átadásra.

Servlet megoldás

A cookie-kat a `Cookie` osztály reprezentálja, ami a `javax.servlet.http` csomagban van. A `HttpServletRequest` példány `getCookies()` metódusával tudjuk lekérdezni a kérésben érkezett cookie-kat, mely egy tömböt ad vissza. Új cookie-t egy új példány példányosításával hozhatunk létre, metódusokkal beállítva a tulajdonságait, és végül a `addCookie` metódussal adhatjuk hozzá a `HttpServletResponse` példányhoz.

Felhasználó nyomkövetése

Két HTTP kérés közül sajnos nem lehet eldönteni, hogy melyik melyik felhasználóhoz tartozik. Nincs egyedi azonosító hozzárendelve egy böngészőhöz sem, ami az azonosítást megoldaná. Az IP-címek tárolása sem jó megoldás, ugyanis több felhasználó kérése is jöhet azonos IP-címről, ha tűzfal például mögött vannak. Pedig több lapból álló műveletsornál szükség lenne a felhasználó azonosítására. Ennek feloldására léteznek úgynevezett munkafolyamatok (session-ök), melyekben információ tárolható két servlet, vagy akár ugyanazon servlet két egymás utáni meghívása között. Egy session pontosan egy felhasználóhoz tartozik, így annak nyomon követése megoldott.

ASP megoldás

Az ASP beépített session-kezelése cookie-kat használ. A rendszer minden felhasználóhoz generál egy egyedi azonosítót, és ezt küldi el a kliensnek, hogy tárolja el cookie-ként. Minden

kérés esetén a böngésző így elküldi a szervernek az egyedi azonosítóját. Ennek a módszernek hátránya, hogy a böngészőkben ezek elfogadásának engedélyezése biztonsági okokból általában kikapcsolható. Csak kliens oldalon így a felhasználók követése nem megoldható. Ezért szükség volt egy mechanizmusra, ami felhasználók követését végzi, cookie-k nélkül.

Saját megoldás

Egy megoldás, hogy adatbázisban tároljuk a session információkat, hogy ahhoz minden szervlet hozzáférhessen, és az egyedi azonosítót egy script generálja, ami szerver oldalon ebben a táblában lesz eltárolva, kliens oldalon pedig az URL-ben adódik mindig tovább.
A tábla szerkezete:

AdminUserSessions		
UserID	int	<pk>
IPAddress	varchar(50)	
SessionID	varchar(50)	
LastRequest	datetime	

Két főbb függvény lett írva a session-ök kezeléséhez. Mindkét függvény használatakor szükség van két include fájlra is, melyek az adatbázis kapcsolatot nyitják meg és zárják le, ezek részletezésével nem foglalkozom.

```
Function RecordSession(UserID, strSessionsTableName)
	Dim rs
	Set rs = Server.CreateObject("ADODB.Recordset")
	strSQL = "Select * From " & strSessionsTableName & " Where UserID = " & UserID
	rs.Open strSQL, conn, adOpenForwardOnly, adLockOptimistic, adCmdText
	ClientIP = Request.ServerVariables("REMOTE_ADDR")
	SID = GenerateNewSID(strSessionsTableName)
	if rs.EOF then
		' no record - must add new one
		rs.AddNew
		rs("UserID") = UserID
	End If
	rs("IPAddress") = ClientIP
	rs("SessionID") = SID
	rs("LastRequest") = Now()
	rs.Update
	rs.Close
	Set rs = Nothing
	RecordSession = SID
End Function
```

Ez a függvény elindít egy session-t. Megnézi, hogy az adott felhasználónak van-e id-je a táblában, ha nincs, hív egy `GenerateNewSID` függvényt, mely generál egyet, és rögzíti a táblában. Majd visszaadja ezt az id-t. Ezt a függvényt a bejelentkező oldalon kell elhelyezni, és amikor az azonosítás sikerült (pl.. jelszó alapján), meghívni.
A második, `BuildLink` függvény a paraméterként megadott string-hez hozzáfűzi a `"?SID=" & SID` string-et, ahol az egyenlőség utáni változó a paraméterként megadott id. Ez a függvény illeszti hozzá minden linkhez az id paramétert.

```
<%
Function BuildLink(strPageName)
	If Len(Trim(SID)) = 0 Then
		SID = Request("SID")
	End If
	BuildLink = strPageName & "?SID=" & SID
End Function
%>
```

A SID változó beállításához és a bejelentkezés ellenőrzéséhez minden oldal elejére be kell szúrni a következő kódot, ami ha nincs bejelentkezve a felhasználó, vagy a bejelentkezése lejárt, átirányítja a bejelentkező oldalra.

Sajnos ez a megoldás sem tökéletes, ugyanis nem lehet kihasználni a session-ök azon nagy előnyét, hogy egyéb adatott is megőrizzenek két servlet hívás között, mint az id-t. Ezért ha adatot akarunk átadni, akkor azt az URL-ben kell, úgy hogy az aktuális oldalon megfelelő linkeket hozunk létre, majd meghívva rá a `BuildLink` függvényt.

Eredeti megoldás

Ha egy új felhasználó kér le egy ASP fájlt, automatikusan létrejön hozzá egy session objektum. Ennek beállíthatjuk a lejárati idejét:

```
<%
Session.Timeout=5
%>
```

Manuálisan meg is szüntethetjük a session-t:

```
<%
Session.Abandon
%>
```

Tárolhatunk és lekérdezhetünk benne értékeket például a következőképpen:
Érték tárolása egy ASP fájlban:

```
<%
Session("username")="Hege"
Session("age")=24
%>
```

Érték olvasása egy másik fájlban:

```
Welcome <%Response.Write(Session("username"))%>
```

Ajánlott, hogy kevés adatot tároljunk, és válasszuk meg az időkorlátot jól, nehogy lejárjon egy felhasználó időhatára, és a session-ben tárolt adatok elvesszenek.

Servlet megoldás

Egy session-t a `Session` osztály reprezentál. A Java megoldás esetén választani lehet, hogy az állapot információkat cookie alapján tárolja, vagy minden url-be generáljon egy egyedi azonosítót a nyomon követés érdekében.
Jellemzői az azonosító, létrehozás időpontja, legutolsó hozzáférés időpontja, élettartam. Az aktuális kéréshez tartozó `Session` objektumot a `HttpServletRequest` `getSession` metódusával lehet lekérdezni.
A session-be adatokat tenni, és azokat kivenni a `putValue` és `getValue` metódusokkal lehet, véglegesen eltávolítani a `removeValue`-val.
A session-t mi magunk is megszüntethetjük az `invalidate()` metódusával.

Kompatibilitási kérdések

Az általános leírásnál és adatbázis hozzáférésnél volt már szó a kompatibilitási problémákról, de álljon itt egy összefoglaló, hogy a rendszer egyes elemei hogyan cserélhetők le.

ASP

Mivel az ASP egy Microsoft technológia, én csak Microsoft termékeket használtam futtatási környezetként. Viszont léteznek alternatívák is, ha már egy kipróbált, beállított rendszerünk van, ne kelljen az ASP technológia miatt átállnunk másra. Windows operációs rendszer és IIS web szerver helyett használhatunk Solaris, Linux, AIX vagy HP-UX operációs rendszert, Apache, Lotus, iPlanet vagy O'Reilly webszervert a Sun Chili!Soft termékének köszönhetően.

Másik technológia az Instant ASP, mely 15 operációs rendszert és 14 web szervert ismer, persze ezek nem mindegyik kombinációja megengedhető.

Java Servlets

A servlet-ek futtatásához egy olyan web szerver szükséges, mely támogatja a servlet-ek futását, vagy egy olyan, amelyhez létezik add-on, ami a servlet-ek futtatását végzi. Rengeteg ilyen termék létezik, szinte minden operációs rendszerre. Abban az esetben, ha váltani akarunk, meg kell nézni, hogy alkalmazásunk a Java Servlet API mely verzióinak lehetőségeit használja ki, és a választott web szervernek, vagy szervlet futtató környezetnek támogatnia kell azt. A legújabb verziója a Servlet API-nak a 2.3-as. A gyártók általában többletlehetőségeket is beépítenek, nem csak a standard API-t implementálják, de ezek használatával vigyázzunk, mert sérülhet a servlet-ünk portabilitása. A szervlet-ünk fordításához szükség van még a Java Development Kit-re, illetve a JSDK-ra (Java Servlet Development Kit), mely szintén rengeteg platformon elérhető. A JSDK emellett tartalmaz egy egyszerű web szervert és egy servlet futtató környezetet, mellyel tesztelhetünk.

Összegzés

Elmondható, bármely technológiát is választjuk, a fejlesztés és az üzembe helyezés folyhat eltérő platformokon, sőt közben is válthatunk, csak figyelni kell, hogy ne alkalmazzunk gyártó specifikus megoldásokat.

Összefoglalás

A feladat hasonlósága és bonyolultsága miatt kellően összehasonlíthattam az Microsoft ASP és Java Servlet technológiát. Mindkettő azonos probléma megoldására jött létre, különböző gyártótól. Későbbi technológia a JSP, mely szintén Java alapú, és jobban hasonlít az ASP-re, ugyanis a Java és HTML kód tetszőlegesen keverhető. Viszont a háttérben ugyanaz történik, ugyanis JSP futtatásakor egy servlet keletkezik, példányosodik, mely a kérés kiszolgálását ténylegesen végzi.
Mindkét esetben ugyanazok a problémák merülnek fel, és a megoldásuk is hasonló. Mindkettőnek van előnye és hátránya egyaránt, ezen finomságokat elemezve, és figyelembe véve válasszunk a technológiák közül. Persze legtöbb esetben szubjektív szempontok döntenek, illetve hogy melyik technológiát alkalmazták már hasonló probléma megoldására, milyen szoftverkomponensek vannak máris készen. A két technológia hasonlósága és hatékonysága miatt ez a viselkedésforma nem elítélendő.

Irodalomjegyzék

1. Csizmazia Balázs: *Hálózati alkalmazások készítése*, Budapest, Kalibán BT. (1998)
2. Eckel, Bruce: *Thinking in Java 2nd Edition*, New Jersey, Prentice Hall (2000)
3. *Java 2 útikalauz programozóknak*, (Szerk.: Nyékiné Gaizler Judit) Budapest, ELTE TTK Hallgatói alapítvány (2000)
4. Vég Csaba, dr. Juhász István, *Java - start!*, Debrecen, Logos 2000 Bt. (1999)

Internet címek

5. The Source for Java™ Technology
http://java.sun.com
6. Java™ 2 Platform, Standard Edition
http://java.sun.com/j2se
7. Java™ 2 Platform, Standard Edition, v1.2.2 API Specification
http://java.sun.com/products/jdk/1.2/docs/api/index.html
8. Bruce Eckel's Mindview, Inc: Free Electronic Book: Thinking in Java, 2nd Edition
http://www.mindview.net/Books/TIJ
9. W3Schools Online Web Tutorials
http://www.w3schools.com
10. Instant ASP 2.0
http://www.halcyonsoft.com/products/iasp.asp
11. Java Servlet Technology – Industry Momentum
http://java.sun.com/products/servlet/industry.html
12. Sun Chili!Soft ASP
http://www.chilisoft.com/