Debreceni Egyetem

Informatika kar

Dokumentumok dinamikus megjelenítése a weben

Témavezető:
Bujdosó Gyöngyi
Egyetemi tanársegéd

Készítette:
Nagy Richárd
Programozó matematikus

Debrecen, 2006

Ábrajegyzék:

Köszönetnyilvánítás

Köszönöm, hogy ez a szagdolgozat elkészülhetett édesapámnak, édesanyámnak, nagyszüleimnek és családom minden tagjának, aki hozzájárult tanulmányaim sikeres befejezéséhez.

Köszönöm témavezetőmnek, aki tanácsaival és észrevételeivel segítette a munkámat.

Köszönöm minden barátomnak és hallgató társamnak akik segítségemre voltak, hogy eljussak idáig.

Végül de nem utolsó sorban köszönöm a barátnőmnek, aki türelemmel viseltetett irántam, mikor a dolgozatom több időmet kötötte le, mint szerettük volna, és buzdított, mikor elakadtam és végig mellettem állt.

Előszó

A szakdolgozat célja az, hogy megpróbáljon átfogó képet adni arról, hogy milyen eszközökkel lehet olyan weboldalakat létrehozni, amelyek a tartalmukat dinamikusan változtatják.

Tulajdonképpen annak meghatározása, hogy mi számít dinamikus megjelenítésnek, nem egyszerű feladat, hiszen ez olyan kifejezés, amit mindenki használ, de egzakt módon nincs definiálva. A bevezetésben ezért megpróbálom körülírni.

A téma teljes kifejtése meghaladná egy szakdolgozat kereteit, ezért kénytelen voltam szűkíteni rajta, és csak egy-egy lehetőséget kiemelni. A lehetőségeket megpróbáltam hatékonyság, valamint rendszer általi korlátozás szempontjából sorrendbe állítani és a legtöbb korláttal rendelkező módszertől haladni a felé, ami a legnagyobb szabadságot adja a fejlesztők számára. Ezek a módszerek önmagukban is sok lehetőséget hordoznak, de többet ötvözve lehet igazán kiaknázni a módszerek minden lehetőségét.

A szakdolgozatnak három fő részre tagolódik, amelyek a dinamikus megjelenítési módszereket tárgyalják. A fejezetek két részből állnak. Először a módszerek elméleti hátterével foglalkoznak, amit a gyakorlati részben felhasználtam, majd pedig az elmélet gyakorlatra való átültetésével foglalkoznak.

Az appletekkel foglalkozó részben egy könyvtárban felhasználható, könyvek katalogizálására használható appletet mutatok be.

A következő rész a szkript nyelvekkel, azon belül kiemelten a JavaScripttel foglalkozik, ahol egy webes könyváruházat valósítottam meg. A szkript nyelvek közül többel is foglalkoztam (PHP, Perl), és eleinte a webáruházat megpróbáltam PHP szkript segítségével létrehozni, de a nyelv szintaktikája, olvashatósága és kezelhetősége számomra áthidalhatatlan korlátokat állított elém, ezért esett a választásom a JavaScriptre [1].

A harmadik rész foglalkozik az adatbázisokkal, azok elérésével, és adatainak felhasználásával a weboldalakon. Ide a JDBC-n alapuló adatbázis-kapcsolatot választottam, mert az első két fejezet Java alapú technikákat mutat be, és úgy gondoltam megmaradok ezen a nyomvonalon.

1. Egy on-line könyváruház létrehozása

Napjainkban szinte minden az internet körül forog. Napjaink felgyorsult világában felmerül igényként, hogy az egymástól távol lévő emberek kapcsolatot tudjanak teremteni egymással nagy távolságból is, számítógépes hálózaton keresztül. Gondoljunk csak az üzleti életre, banki rendszerekre, információcserére, oktatásra. Az internet ezáltal közelebb hozza egymáshoz a dolgokat, így a globalizáció a Marshall McLuhan-féle „globális falu" [2] gondolatának megvalósítójává vált.

A számítógép térhódítása mellett a kultúra sem szabad, hogy háttérbe szoruljon, ebből kifolyólag gondoltam arra, hogy egy on-line könyváruházat valósítok meg, ami a kettőnek a kombinációja. A mai felhasználóbarát programok és honlapok készítésének világában arra van szükség, hogy egy weboldal könnyen kezelhető legyen, valamint a látogatók vizuális igényeit is kielégítse. Természetesen a vizualitás nem elegendő a funkcionalitás hiányában.

A tervezési szakaszban az volt az elsődleges probléma, hogy meg kellett határozni mindazon funkciókat, amelyeket egy webáruháznak el kell látnia. Minden ilyen jellegű áruháznak rendelkeznie kell átlátható struktúrával, biztosítania kell választási lehetőséget, és a nem átgondolt döntéseknek megváltoztathatóaknak kell lenniük.

Az on-line áruház létrehozásakor elsődleges szempont volt számomra a strukturáltság és a bővíthetőség. A struktúra kialakítása érvényes mind magára a honlapra, mind pedig a forráskódokra. Az átlátható szerkezet nagyban hozzájárul ahhoz, hogy az alkalmazás a későbbiekben a lehető legegyszerűbben bővíthető legyen.

A webáruházhoz kiegészítőként tartozik egy katalógus applet, amelyik könyvekről tud információkat megjeleníteni. Jelenleg könyveknek csak a borítóját jeleníti meg, de fejlesztés alatt áll. A későbbiekben további információk megjelenítésére is képes lesz, mint például könyvek könyvtári adatai (például kiadó, kiadás éve, ISBN szám), vagy akár egy rövid tartalmi ismertető is.

A könyvesbolt megvalósításához szükséges volt valamilyen szkript nyelv ismerete. A megvalósítás kezdeti szakaszában PHP nyelven implementáltam az adott felhasználói interakci-

ókra reagáló szkripteket, és csak jelentős idő és energiabefektetés után döntöttem úgy, hogy célszerűbb számomra JavaScripttel dolgozni.

A továbbiakban mindezek gyakorlati és elméleti kifejtése következik.

2. Megjelenítés a weben

2.1. Webes megjelenítés

Kezdetben a számítógépek önállóan, egymástól függetlenül léteztek. Az információcsere közöttük körülményes volt, nehezen megoldható. A szöveges információt ki kellett nyomtatni, valamilyen úton el kellett juttatni a másik számítógéphez, ahol a papíron lévő adatokat fel kellett vinni a számítógépre. Ez rengeteg esetben hosszadalmas és fáradságos procedúra volt.

Ennek elkerülésére kifejlesztették azt a módszert, hogy vezetéken keresztül - először még csak szöveget - juttassanak el adatokat egyik géptől a másikig. Létrejöttek a számítógépi hálózatok.

Aztán idővel az információ áramlás sebessége nagyobb lett, az elszigetelt számítógépi hálózatok összekapcsolódtak, és az adatok típusát tekintve egyre szélesebb skálán képesek küldeni, illetve fogadni. Megjelent az Internet.

Az Interneten való információ közlést viszont szabványosítani kellet. A World Wide Web konzorcium megalkotta a HTML (HyperText Markup Language) oldalleíró formát, ami néhány év alatt meghódította a világot.

2.2. XHTML röviden

A HTML 4 előtt a megjelenítésre vonatkozó információk be voltak építve a HTML kódba elemek, attribútumok és értékek formájában. Csakhogy az elemek pontos helyét a weblapon belül majdnem lehetetlen volt megadni. A táblázatokat lehetett mondjuk formai megjelenítésre használni, nem pedig táblázatos adatok közlésére. Ugyanez igaz a szövegbehúzásra is. A CSS előtt nem volt lehetőség behúzás megjelenítésére. A behúzott szöveg megjelenítésére behúzásos elemeket, például listákat és idézetblokkokat használtak. Ha a megjelenített szöveg nem lista, vagy idézett blokk volt, akkor ugyanaz történt, mint a táblázat formázásra történő használatakor, felborult az adott elem szerepe és felhasználási módja közötti logika.

Az Internet fejlődésével eljutottunk oda, hogy az információnak helyesen kell megjelennie egy webböngészőben, egy kézi számítógépen, vagy akár egy mobiltelefonon is. Viszont ha a HTML oldal elemeit nem a nekik szánt szerepkörben használjuk, akkor nem minden eszközön azt a képet fogjuk látni, amit várunk. Tehát a helyes megjelenítés érdekében az elemek csak a logikailag megállapított céljukra szabad használni: a címsoroknak címsoroknak kell lenniük. a táblázatba táblázatos adatokat kell felvennünk.

Az elemek logikai szerepének, és a megjelenítésre vonatkozó információk elkülönítésére szolgál az XHTML (eXtensible HyperText Markup Language) és a CSS (Cascading Style Sheets).

Az XHTML jelentése magyarul bővíthető hiperszöveges leírónyelv. Már az elnevezés is sokat elárul arról, hogy mennyivel képes többre, mint a HTML. Az XHTML az XML (eXtensible Markup Language) nyelvtani szabályait követi. A bővíthető leírónyelvek bővíthetők olyan modulokkal, amelyek képesek olyan feladatok végrehajtására, amilyen például a matematikai számítások elvégzése, vagy a képek megrajzolása. Az XHTML-ben megírt weblapok könnyen együtt tudnak működni az XML nyelvvel.

Az XHTML szolgál a tartalom logikai felosztására, megmondja az olalon szereplő elemek funkcióját.[3]

2.3. Dinamikus megjelenítés meghatározása

Az internetes információáramlás kezdetben csak szövegek közlésére korlátozódott, majd később már képeket is meglehetett jeleníteni a hiperszöveges dokumentumokban. De a honlapok még mindig statikusak maradtak. Ez azt jelenti, hogy valahányszor megjelenítjük, mondjuk egy webböngészőben, akkor mindig ugyanazt fogjuk látni.

Ahogyan létrejöttek az adatbázisokra épülő többfelhasználós portálok, felmerült az igénye annak, hogy a felhasználók saját igényeiknek, illetve jogosultságaiknak megfelelő információkat, dokumentumokat tekintsenek meg, tehát a megjelenítésnek dinamikussá kellett válnia.

A webes fejlesztőeszközök fejlődése is követte ezt az irányt. Létrejött a Java applet, a különböző szkript nyelvek (pl.: PHP, JavaScript), valamint a Flash.

Az applet interaktív, eseményvezérelt Java program futtatását teszi lehetővé HTML

környezetben, a szkript nyelvek segítségével elérhetjük, hogy a honlap forráskódja dinamikusan változzon, az oldal újraírása nélkül, míg a Flash-sel pedig interaktív, multimédiás animációkat hozhatunk létre, amivel gazdagíthatjuk honlapunk látvány és hangulatvilágát.

3. Java appletek dinamikus lehetőségei

Az applet olyan Java program, amelyik beágyazható HTML oldalba, a hálózaton közzétéve pedig távoli gépekről elérhető, és ott lokálisan futtatható. Így nem csak interaktív weblapokat, hanem kliens-szerver alkalmazásokat is létre lehet hozni.

Az appletek dinamikussága azon alapul, hogy bár tudnak kommunikálni a böngészőprogramokkal, önálló programok. Önálló grafikus kezelőfelülettel rendelkeznek, ami eseményvezérelt programozást tesz lehetővé, valamint felhasználói interakcióra történő dinamikus válaszadást. A JDBC-n (Java DataBase Connectivity) keresztül pedig adatbázishoz is lehet kapcsolódni, ahonnan információt nyerhetünk ki a felhasználók számára.

3.1. Applet beágyazása

Minden egyes beágyazott applet esetén a böngészőprogram először példányosítja az Appletet, majd létrehoz egy AppletStub nevű interfészt imlementáló beágyazó objektumot, amin keresztül az applet és a böngésző egymással kommunikálni képesek. A példányosítással egyidőben beállítódik a futtatási környezet is, ami tulajdonképpen magát az Appletet megjelenítő programot reprezentáló AppletContext interfészt implementáló objektum. Ezen keresztül kommunikálhatnak egymással az azonos HTML oldalba ágyazott appletek. Az applet konstruktorának lefutásakor még nem állnak rendelkezésünkre a böngészőspecifikus lehetőségek, ugyanis ekkor a programot reprezentáló objektumpéldány még nem jött létre.

Egy appletet az <APPLET> kulcsszóval lehet beágyazni egy HTML oldalba. Ennek szintaxisa a következő:

```
<APPLET [CODEBASE=url] [ARCHIVE=archívum]
        CODE=fájlnév vagy OBJECT=fájlnév
        [ALT=szöveg]
        [NAME=azonosító]
```

WIDTH=szám HEIGHT=szám
[ALIGN=érték]
[VSPACE=szám] [HSPACE=szám]
[MAYSCRIPT]
>
[<PARAM NAME=azonosító VALUE=érték>]…
[HTML sorok]
</APPLET>
[4]

A beágyazott appletet a böngészőprogramba beépített Virtuális gép hajtja végre. A paraméterek átadásakor a HTML-től eltérően a Java különbséget tesz a kis- és nagybetűk között.

A paraméterek jelentése:

- **Codebase:** Az applet kódját tartalmazó könyvtár URL címe. Ha nincs megadva, akkor a beágyazó HTML dokumentum könyvtárának URL címe kerül felhasználásra.
- **Archive:** az applet kódját és erőforrásait tartalmazó archívum(ok) neve(i).
- **Code:** az applet kódját tartalmazó, a codebase-hez relatív bájtkód fájl neve.
- **Object:** az appletet szerializált formában tartalmazó fájl neve.
- **Alt:** ez a szöveg jelenik meg ha a böngésző nem tudja megjeleníteni az appletet, de felismeri az <APPLET> kulcsszót.
- **Name:** a továbbiakban ezzel a névvel is lehet hivatkozni az appletre.
- **Width:** az applet számára biztosítandó grafikus terület szélessége képpontban mérve.
- **Height:** az applet számára biztosítandó grafikus terület magassága képpontban mérve.
- **Align:** az applet ábrázolásának igazítását lehet vele szabályozni.
- **Vsapce:** az applet alatt és felett üresen hagyandó képpontsorok száma.
- **Hspace:** az applet mellet üresen hagyandó képpontoszlopok száma.
- **Mayscript:** ha szerepel, akkor az applet kommunikálhat JavaScripttel.

Ha a böngésző képes appletek megjelenítésére, akkor az összes <APPLET> nyitó és záró kulcsszavak között szereplő HTML utasítás figyelmen kívül marad, különben pedig az appletspecifikus kulcsszavaknak nem lesz hatásuk.

Az applet számára paramétereket a <PARAM>kulcsszóval adhatunk át, ami két attribútummal rendelkezik.

- **Name:** az appletnek átadandó paraméter neve. Nincs különbség kis- és nagybetűk között.
- **Value:** az appletnek átadandó paraméter értéke. Különbség van a kis- és bagxbetűk között.

A paramétereket az Applet osztály getParaméter metódusával lehet lekérdezni. A paramétereként megadott nevű appletparaméter értékét String objektumként adja vissza. Ha az appletnek nincs ilyen nevű paramétere, akkor a visszatérési érték null. Ha nem String típusú paramétert szeretnénk átvenni, akkor a kapott Stringet még konvertálni kell a megfelelő típusra. Az <APPLET> kulcsszónak is le lehet kérdezni a paramétereit, így megtudható például az applet számára fentartott grafikus terület mérete, vagy mondjuk az applet neve is.

3.2. Az appletek életciklusa

A konstruktor végrehajtásakor az applet beágyazó objektuma még nem létezik, ezért a konstruktorban még nem hivatkozhatók az AppletContext metódusai. Az applet futásának vezérlésért appletpéldányonként külön szálcsoport a felelős. Ennek megvalósítása implementációfüggő. Ez a szálcsoport a konstuktor lefutása után a következő négy metóduson keresztül vezérli a program futását:

- **public void init() –** az applet inicializálásakor kerül végrehajtásra, az applet konstruktorának lefutása után közvetlenül. A megjelenítést vezérlő metódusok csak akkor hívódnak meg, ha ez már lefutott.
- **public void start() –** az applet indításakor kerül végrehajtásra. Közvetlenül a meghívás előtt az applet aktív állapotba kerül. Az init metódus végrehajtása után hívódik meg, illetve minden egyes alkalommal, amikor az appletet újra kell indítani.
- **public void stop() –** az applet megállításakor kerül végrehajtásra. A meghívása előtt az applet inaktív állapotba megy át. Akkor hívódik meg, amikor az appletnek már nem kell tovább futnia.

- **public void destroy()** – az applet megszüntetésekor kerül végrehajtásra. Végrehajtásra, közvetlenül a finalize metódus meghívása előtt, amit a szemétgyüjtögető hív meg akkor, amikor az objaktumra már nem történik hivatkozás, és az általa lefoglalt memória felszabadítható. Itt célszerű az applet által lefoglalt erőforrásokat felszabadítani, a megnyitott adatfolyamokat lezárni.

3.3. Grafikus felhasználói felület

Az appletekhez minden esetben tartozik GUI (Graohical User Interfész), azaz grafikus felhasználói felület, amin keresztül az applet a felhasználóval kommunikál. Ez legtöbbször egérrel való kattintás, és szövegbevitel feldolgozása, valamint az ezekre az eseményekre való reagálás. Mindezekből következik, hogy az appletek megírása eseményvezérelt programozási szemléletet igényel.

Minden appletnek ősosztálya a **java.applet.Applet** osztály. A legegyszerűbben grafikus felhasználói felületet az Applet ősosztálytól örökölt **public void paint(java.awt.Graphics)**, illetve **public void update(java.awt.Graphics)** metódusok felüldefiniálásával lehet. A **paint** metódust a böngészőprogram automatikusam meghívja, amikor az appletet újra kell rajzolni, míg a **public void repaint()** metódussal való direkt újrarjzoltatás az **update** metódust hívja meg, amely alapértelmezetten először szürke színnel átfesti az applet teljes területét, majd meghívja a **paint** metódust. Ha a szürkére való festést el szeretnénk kerülni, akkor felüldefiniálhatjuk az **update** metódust, vagy használhatjuk az úgynevezett offsreen technikát is, amikor a képet először a memóriapufferben állítjuk elő, és csak azután jelenítjük meg.

3.3.1. Appletre vonatkozó grafikus megszorítások

Mivel az appletek megjelenítése HTML oldalba ágyazva történik, ezért biztonsági és egyéb

technikai okok miatt megszorításokat kellett tenni a megjelenítésre vonakozóan.

- A böngészőprogram az <APPLET> HTML kulcsszó feldolgozásakor rögtön létrehozza az adott applet egy példányát a paraméterben megadott méretekkel. Ezt a későbbiek folyamán semmilyen módon nem lehet megváltoztatni, és a böngészőprogram minden ilyen irányú kísérletet figyelmen kívül hagy.
- Az appleten belül nem lehet sehogysem megtudni az applet HTML oldalon belüli koordinátáit. Minden applet saját magát a (0,0) koordinátán látja, ugyanis nem a böngésző program koordinátarendszerét használja, hanem a számára lefoglalt ablakterületét.
- Az appletek számára nem javasolt önálló ablak nyitása biztonsági okokból önálló ablak nyitása biztonsági okok miatt, nehogy megtévesszék a felhasználót.Ha egy applet mégis egy önálló ablakot nyitna meg, azt a böngészőnek minden esetben figyelmeztető felirattal kell ellátnia.
- Az appletet nem befolyásolja a HTML oldalbeli környezete. Soha nem veszi figyelembe a HTML oldal beállított színeit, sem az oldal hátterének beállított képet. Az appletben alapértelmezésben beállított színkombináció a szürke háttérszín és a fekete előtérszín.

3.4. Applet kommunikációja a böngészővel

Az applet életcikllus-metódusai csak a böngészőprogram és az applet közötti kommunikáció egyik irányát jelentik. Az appletek igénybe vehetik még a beágyazási környezetük, rendszerint egy böngészőprogram szolgáltatásait. Ezeket a szolgáltatásokat csak a már beágyazott appletek vehetik igénybe, tehát az applet konstruktorában megadott metódusok például nem. A böngészőprogram által nyújtott lehetőségek a következők:

- Egy applet lekérdezheti kódjának, illetve az őt tartalmazó HTML oldalnak az URL

címét.

- Kép, hang megjelenítése, letöltése, illetve lejátszása.
- Új URL mjegjelenítése a böngészőben.
- Böngészőprogram státuszsorának használata.
- Kommunikáció ugyanazon HTML oldal többi appletjével.

3.4.1. Új oldal megjelenítése

A következő metódusok lehetővé teszik, hogy egy applet tetszőleges, URL-jével megadott dokumentumot megjelenítsen a böngészőprogram valamely frame-jében, azaz a <FRAME> HTML kulcsszóval definiált elkülönülő böngészőablak-területen:

- **showDokument(URL):** az adott dokumentum megjelenítése a teljes böngészőben.
- **showDokument(URL, String):** adott dokumentum megjelenítése a kijelölt frame-ben. A frame.et meghatározó String paraméter a frame HTML neve, vagy a következő értékek valamelyike lehet:
    - **_self:** a megjelenítés az aktuális frame-ben történik.
    - **_parent:** a megjelenítés a szülő frame-ben történik.
    - **_top:** a megjelenítés a böngészőablak egészében történik.
    - **_blank:** a megjelenítés egy új, név nélküli böngészőablakban történik.
    - Egyéb érték esetén a rendszer egy ablak vagy egy frame neveként használja a String-et. Ha az adott nevű ablak, vagy frame már létezik, akkor ott jelenik meg, különben pedig egy új böngészőablak jön létre, aminek a neve a String lesz.[5]

3.5. A könyvkatalógus applet bemutatása

Az appletek eseményvezérelten való programozását és a dokumentumok frame-ekbe való betöltését kihasználva készíthetünk akár olyan appletet is, amivel megjeleníthetjük modjuk egy könyvtárban található könyveknek a borítóját.

A példában olyan HTML oldalt láthatunk, amely két frame-et tartalmaz. A bal oldalon található maga az applet, amelyet úgy használhatunk, mintha menüből választanánk ki a számunkra érdekes könyvet. A könyvek címei nyomógombokként jelennek meg.

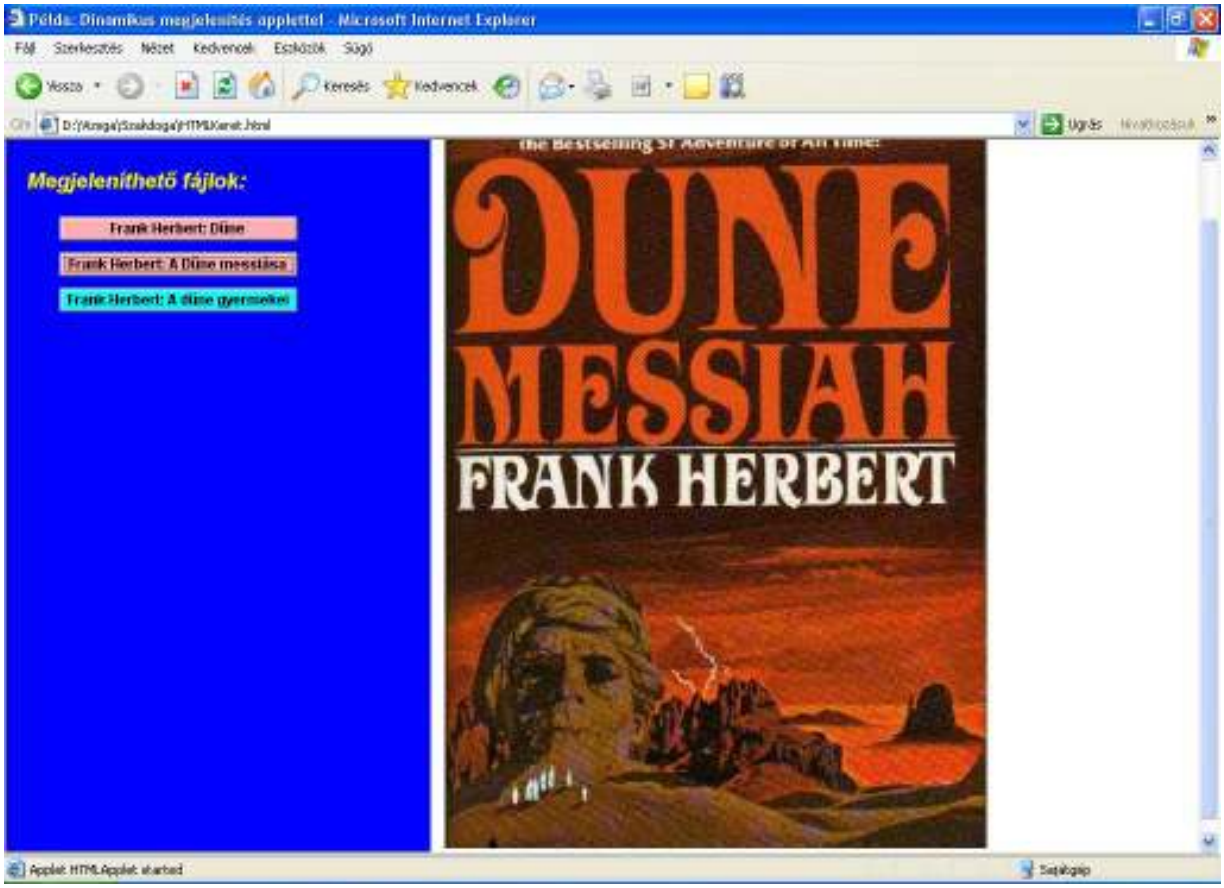
Ha valamelyik nyomógombot megnyomjuk az egér bal oldali gombjának kattintásával, akkor a jobb oldali, eredetileg üres frame-ben megjelenik a kiválasztott könyv borítója.

Az applet forráskódja a következő:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;

public class HTMLApplet extends Applet implements ActionListener {

 public static final Color BGCOLOR=Color.blue;
 public static final Color TEXT=Color.yellow;
 public static final Color BGLINK1=Color.cyan;
 public static final Color BGLINK2=Color.pink;
 public static final Color LINK=Color.black;
 public static final int LINKNUM=3;
 public static final String FNAME="jobb";

 public Label label=new Label();
 public Button[] b=new Button[LINKNUM];
 public boolean[] click=new boolean[LINKNUM];
```

```
public String[] link={"Dune-Cover.jpg",
                      "Dune-Messiah_cover.jpg",
                      "Dune-Children_cover.jpg"};
public String[] cimke={"Frank Herbert: Dűne",
                       "Frank Herbert: A Dűne messiása",
                       "Frank Herbert: A dűne gyermekei"};

public void init() {
 int x=35,y=50,dy=30,w=200,h=20;

 setLayout(null);
 label.setBounds(5,5,200,30);
 label.setBackground(BGCOLOR);
 label.setForeground(TEXT);
 label.setFont(new Font("Arial",Font.BOLD+Font.ITALIC,18));
 label.setText("Megjeleníthető fájlok:");
 add(label);

 for(int i=0;i<LINKNUM;i++) {
  b[i]=new Button();
  b[i].setBounds(x,y,w,h);
  y+=dy;
  click[i]=false;
  b[i].setBackground(BGLINK1);
  b[i].setForeground(LINK);
  b[i].setFont(new Font("Arial",Font.BOLD,12));
  b[i].setLabel(cimke[i]);
  add(b[i]);
  b[i].setActionCommand(""+i);
  b[i].addActionListener(this);
```

```
 }
 }

public void actionPerformed(ActionEvent ev) {
 URL url=null;
 AppletContext ac=getAppletContext();

 try {
  int i=Integer.parseInt(ev.getActionCommand());
  url=new URL(getCodeBase(),link[i]);
  ac.showDocument(url,FNAME);
  click[i]=!click[i];
  if(click[i]) b[i].setBackground(BGLINK2);
  else b[i].setBackground(BGLINK1);
  }
 catch(Exception e) {
  System.out.println(e.toString());
  }
 validate();
 }

public void paint(Graphics g) {
 setBackground(BGCOLOR);
 }
}
```



1. Könyvkatalógus képe

Az applet továbbfejleszthető úgy, hogy nagy könyvtári rendszerek is használhassák akár. Mondjuk új borító felvételének, illetve meglévők törlésének a segítségével, valamint egy keresővel, amivel szűrni lehetne a megjelenő könyveket, valamint információt is lehet megjelíteni a tartalomról, vagy a könyv adatairól.

4. A szkript nyelvek dinamikus lehetőségei

A szkript nyelvek rendelkeznek azzal a tulajdonsággal, hogy forrásszinten beágyazhatók egy HTML oldalba, így egy interaktív HTML oldal készítésekor nem válik külön az oldalnak és az interaktivitást megvalósító vezérlésnek a megírása.

4.1. JavaScript meghatározása

A JavaScript a Java nyelv szkript változata, annak szinte minden lehetőségével rendelkezik.

A JavaScript, mint programozási nyelv több rétegből tevődik össze. Ezek a rétegek a következőek:

- **JavaScript alapnyelv (Core):** ez definiálja a nyelv alapelemeit és szintaxisát, például a vezérlési szerkezeteket, típuskonstrukciókat, alaptípusokat.
- **Kliensoldali JavaScript:** az alapnyelvet böngészők vezérlését lehetővé tevő objektumokkal, illetve egy HTML oldalt leíró dokumentum objektummodellel (Document Object Model (DOM)) egészíti ki.
- **Szerver oldali JavaScript:** az alapnyelvet szervereken végezhető funkciók (például adatbázis-elérés, fájlműveletek) vezérlését lehetővé tevő objektumokkal egészíti ki.

4.2. A JavaScript és a Java összehasonlítása

Beágyazás szemontjából a JavaScript kódja be van ágyazva a HTML oldalba, míg az applet kódja elkülönül a HTML oldalétól, csak a vezérlése történik onnan.

Végrehajtás szempontjából a kliensgép böngészőprogramja interpretálja a HTML oldalba beágyazott JavaScript forrásszöveget, míg applet esetében a szerver gépen előzőleg lefordított és onnan letöltött bináris bájtkódot a kliensgép Java virtuális gépe interpretálja.

A program felépítése szempontjából a JavaScript utasítások és eseménykezelők halmaza, míg a Java alkalmazás osztályok definíciója.

A JavaScript objektumalpú nyelv, a Java pedig objektum orientált.

A JavaScript gyengén, az applet erősen típusos, tehát a JavaScriptben nincs szükség a változók deklarálására, és a típusukat sem kell explicit módon megadni, Java esetében pedig minden változót deklarálni kell és a deklarációban meg kell adni a változó típusát is, ami mendíg vagy a Java alaptípusainak egyike, vagy pedig egy osztály.

A típusellenőrzés a JavaScript esetében dinamikus, tehát objektumreferencia típusellenőrzés csak futási időben történik, Javaban pedig statikus, tehát az objektumreferencia típusellenőrzése már fordításkor megtörténik.[6]

4.3. JavaScript beágyazása

A JavaScriptet három helyen lehet használni egy HTML oldalon belül:

- A <SCRIPT> </SCRIPT> HTML kulcsszavak között utasítások végrehatására és függvények definiálására.
- A HTML elemeknél eseménykezelők megírására.
- HTML elemek paramétereinek megadásakor. Ahol paraméterértéket lehet megadni, ott állhat JavaScript kifejezés a következő formában:
  Paraméternév=&{*JavaScript kifejezés*};

Például a <HR WIDTH=”&{szazalek()};” ALIGN=”LEFT”> a HTML oldal balról számított annyi százalékánál húz egy vonalat, amennyit a **szazalek()**, egy a HTML oldalon már korábban definiált függvény JavaScript függvény visszaadott.

4.4. A <SCRIPT> HTML kulcsszó

A <SCRIPT> </SCRIPT> HTML kulcsszavak közé tett JavaScript a HTML oldal letöltése után, de még a megjelenítés előtt kerül kiértékelésre. A kiértékelés alatt a definiált függvények eltárolódnak, az utasítások pedig végrehajtódnak. A szintaxis a következő:

<SCRIPT
[LANGUAGE=szkriptnyelv]
[SRC=fájl]
>
JavaScript utasítások
[<NOSCRIPT>
HTML utasítások
</NOSCRIPT>]
</SCRIPT>

- **Language =** a szkript nyelvét adja meg. Néhány lehetséges érték:
  - **„JavaScript”** – a JavaScript alapverzióját jelöli. Ilyen szkripteket a Netscape 2.0-ás verziója is végre tud hajtani
  - **„JavaScript1.1”** – A JavaScript 1.1-es verzióját jelöli. Ilyen szkripteket a Netscape 3.0-ás, vagy annál újabb böngészők tudnak végrehajtani.
  - **„JavaScript1.2”** – A JavaScript 1.2-es verzióját jelenti. Az ilyen szkriptek végrehajtásához legalább 4.0-ás Netscape szükséges.
- **Src** = a szkriptet tartalmazó fájl URL címe. A fájl egyszerű ASCII formátumú, csak a szkript szövegét tartalmazhatja, rendszerint .js kiterjesztéssel. Ezen attribútum

megadása esetén a <SCRIPT> </SCRIPT> kulcsszavak között megadott minden JavaScript sor figyelmen kívül marad.

A <NOSCRIPT> </NOSCRIPT> kulcsszavak közötti HTML szöveg csak akkor fog látszani, ha a böngésző nem képes JavaScript értelmezésére, vagy az nincs engedélyezve.[7]

4.5 A webes könyváruház bemutatása

Az appletekről szóló részben említett könyvkatalógus példát továbbgondolva JavaScript segítségével készíthetünk akár olyan HTML oldalt is, ami egy egyszerű webes könyváruházat valósít meg. Az ilyen portálok az internet kereskedelmi használatában fontos szerepet játszanak, hálózaton keresztül lehet válogatni az árukból, berakni azokat a képzeletbeli bevásárlókocsiba, és az interneten keresztül meg is lehet rendelni azokat. Az alkalmazás **frame**-eket használ, a bevásárlókocsi kezelésére pedig JavaScriptet.

A főoldal forráskódja a következő:

```
<HTML>
<HEAD>
<TITLE>Könyváruház</TITLE>
<SCRIPT>
megvettlista = []

function vesz(neve, ara) {
   while (true) {
      db = prompt("Hány darab "+neve+"-t akar venni?",1)
      if (db == null) return
      dbszam = parseInt(db)
      if (isNaN(dbszam) || dbszam<1) alert("Érvénytelen szám: "+db)
      else break
   }
```

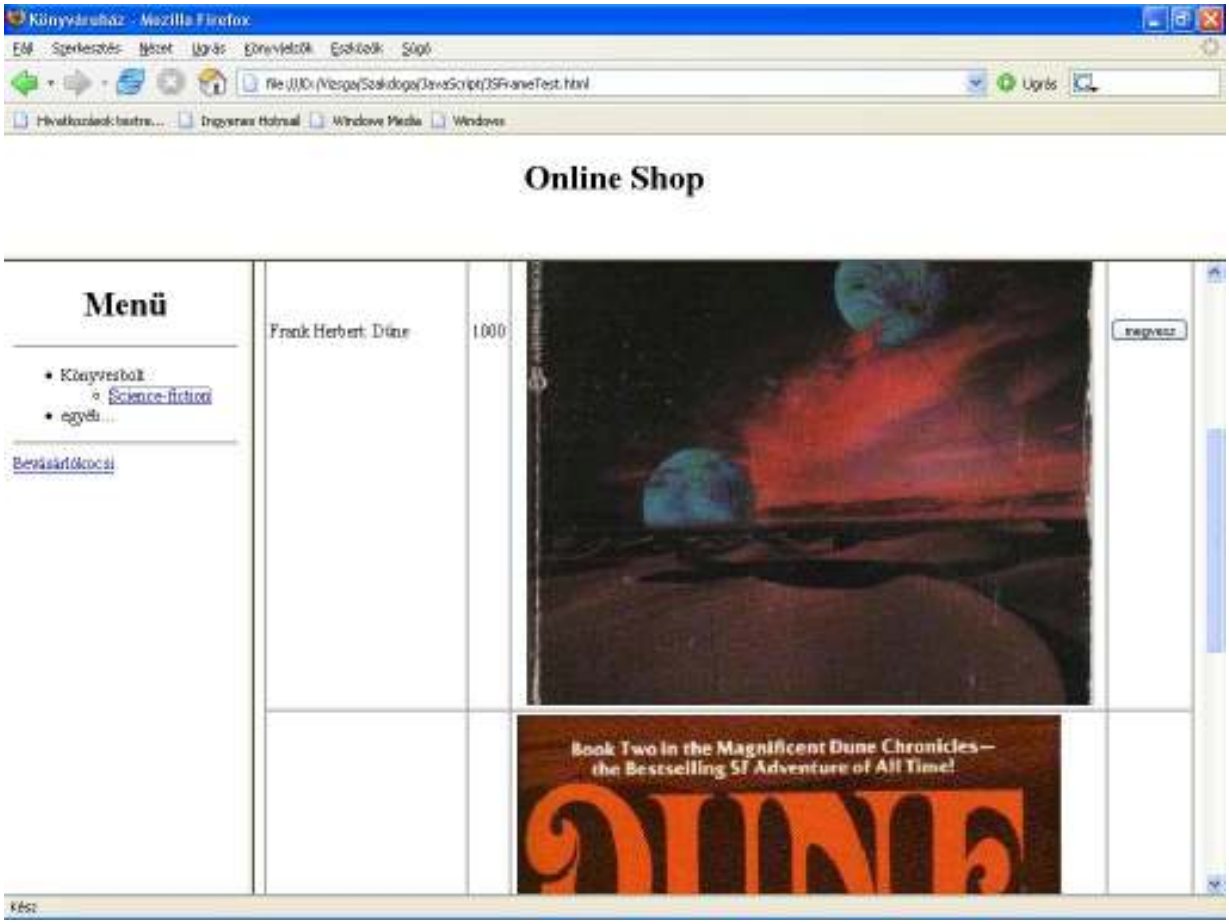
```
    var index = 0
    for (; index<megvettlista.length; index++) {
        adatok = megvettlista[index].split(":")
        if (adatok[0]==neve) {
            if (confirm("Már van "+adatok[2]+" db. "+neve+
                    " a bevásárlókocsiban.\n"+
                    "Akar még "+dbszam+" db.-ot venni hozzá?")) {
                dbszam += parseInt(adatok[2])
                break
            } else return
        }
    }
    megvettlista[index] = neve+":"+ara+":"+dbszam     // megvett áru eltárolása
    alert(dbszam+" db. "+neve+" bekerült a bevásárlókocsiba.")
}
function nemveszmeg(mit) {
    for (var i=0; i<megvettlista.length; i++)
        if (megvettlista[i] == mit) {
            megvettlista.splice(i, 1)
            break
        }
}
function kilistaz(hova) {  // bevásárlókocsi tartalmának listázása táblázatként
    hova.writeln("<TABLE BORDER=\"1\" CELLPADDING=\"3\" "+
            "CELLSPACING=\"0\" WIDTH=\"100%\">")
    hova.writeln("<TR><TD>Név<TD>Ár<TD>Darabszám<TD>Összeg")

    osszosszeg = 0
    osszdarabszam = 0
    megvettlista.sort()
```

```
   for (var i=0; i<megvettlista.length; i++) {
      adatok = megvettlista[i].split(":")
      osszeg = adatok[2] * adatok[1]
      osszosszeg += osszeg
      osszdarabszam += parseInt(adatok[2])
      hova.writeln("<TR><TD>"+adatok[0]+"<TD>"+adatok[1]+"<TD>"+adatok[2]+
            "<TD>"+osszeg+"<TD>")
      hova.writeln("<FORM><INPUT TYPE=\"button\" VALUE=\"töröl\" onclick=\""+
            "parent.nemveszmeg('"+megvettlista[i]+"\');"+
            "window.history.go(0)\"></FORM>")
   }
   hova.writeln("<TR><TD COLSPAN=2>Összesen<TD>"+
         osszdarabszam+"<TD>"+osszosszeg)
   hova.writeln("</TABLE>")
}
</SCRIPT>
</HEAD>
<FRAMESET rows="100,*">
<FRAME NAME="header" SRC="JSFrameTestHeader.html">
<FRAMESET cols="20%,80%">
<FRAME NAME="menu" SRC="JSFrameTestMenu.html">
<FRAME NAME="work" SRC="JSFrameTestHeader.html">
</FRAMESET>
</FRAMESET>
<NOFRAMES>
<BLINK>
Ez a böngésző nem ismeri a FRAME-eket!
</BLINK>
</NOFRAMES>
</HTML>
```

A szkript működése

A böngészőablak három részből áll. A felső részben az üzlet neve látszik, ahonnan vásárolni szeretnénk. Ez a rész még tartalmazhat további információt is a cégről, amelyik az online üzletet üzemelteti. A bal oldalon egy menü található, ahol az áruk kategóriái közül lehet választani. A jobb oldal pedig az a terület, ahol a nekünk tetsző árut tudjuk berakni a bevásárlókocsinkba, illetve megtudjuk tekinteni, hogy mi került már be a bevásárlókocsiba, vagy a rendelést elküldeni.

Ha kiválasztunk egy árut meg kell adni, hogy darabot szeretnénk belőle vásárolni. Ha már korábban raktunk ilyen terméket a bevásárlókocsiba, akkor a program figyelmeztető üzenetet küld. A vásárlás bejezése után a bevásárlókocsi hivatkozásra kattintva kilistázódnak a megvenni kívánt árúk. Itt lehetőségünk van a lista módosítására, vagy a rendelés elküldésére.



2. Online könyvesbolt képe

A JavaScript kezeli a bevásárlókocsit. A kocsiba került árukat a megvettlista nevű tömb reprezentálja. A tömb az árukat szöveges formában tartalmazza. Minden egyes tömbelem tartalmazza az áru nevét, árát, és a megrendelt darabszámot, egymástól kettősponttal elválasztva. Az áru bevásárlókocsiba helyezése a főframe vesz metódusának meghívásával történik. A metódus első paramétere az áru neve, a második pedig annak ára. A metódus meghívása pedig a megvesz feliratú gomb eseménykezelőjéből történik a következő képpen:

onclick="parent.vesz('Dune', 1000)”

A metódus a megvett árut felveszi a listába, vagy módosítja a darabszámot.

A bevásárlókocsi tartalmának kilistázása dinamikus HTML generálással történik. A bevásárlókocsi hivatkozás céljaként megadott HTML oldal magát a listát a következő formában tartalmazza:

```
<SCRIPT>
parent.kilistaz(document)
</SCRIPT>
```

A táblázat utolsó oszlopába egy nyomógomb kerül, aminek a segítségével az adott nyomógomb törölhető a listáról. A törlés az adott tétel kitörlésését jelenti a listát reprezentáló tömbből, majd a list tartalma újra kiíratásra kerül.

3. ábra Bevásárlókocsi tartalma

5. Adatbázisok kezelése dinamikus weblapokon

Az adatbázisok használata nagy előnyt jelent az eddigi lehetőségekhez képest. Hiszen ha csak appletet, vagy a szkript nyelvek valamelyikét használjuk, és nincsen mögötte adatbázis, akkor azokat az adatokat, amikkel dolgozunk, explicit módon bele kell írni a kódba. Tehát ha módosulnak az adatok, amikkel dolgozunk, vagy újabb adatokkal szeretnénk dolgozni a már meglévők mellet, vagy pedig törölni szeretnénk közülük, akkor a kód módosítása, aktualizálása meglehetősen körülményes és lassú feladat. Ugyanakkor, ha az adatokat adatbázisban tároljuk, az adatbázis megváltozása automatikusan megváltoztatja a HTML dokumentumunk képét is, nincsen szükség a kód újraírására.

Egy appletből, mivel az a Java része, a szintén a Java részét képező JDBC-n (Java DataBase Connectivity – Java alpú adatbázis kapcsolat) keresztül lehet egy adatbázis-kezelő rendszerhez kapcsolódni.

A szkript nyelvek (a JavaScriptet kivéve) ODBC-n (Open DataBase Connectivity- nyílt adatbázis kapcsolat) keresztül kapcsolódhatnak egy adatbázis-kezelő rendszerhez.[8]

5.1. JDBC felépítése

A JDBC kommunikációt biztosít a Java programok és az adatbázis-kezelő rendszerek között.

A JDBC egy Java API (Application Programming Interface – felhasználó program interfész), amelynek segítségével SQL (Structured Query Language – strukturált lekérdezőnyelv) utasításokat lehet kiadni Java alkalmazásból, vagy appletből.[9]

A JDBC 2.0-ás verziója két részből áll:

- **JDBC alap API (Core):** az adatbázis eléréséhez szükséges alapvető típusokat írja le. Ezeket a java.sql standard Java csomag tartalmazza.
- **JDBC standard kiterjesztés API (Extension):** további haladó szintű JDBC típusokat ír le. Ezeket a javax.sql Java csomag tartalmazza. Ennek az API-nak a szolgáltatásait a következő csoportokba sorolhatjuk:
  - Lehetővé teszi a JNDI (Java Naming and Directory Interface – Java megnevezés és könyvtárszolgáltatás) használatát adatbázisok megnevezésére. Ezáltal adatbázisokhoz logikai nevet lehet rendelni, illetve meg lehet adni az eléréshez szükséges JDBC meghajtóprogramot, majd a Java program a továbbiakban ezen a néven hivatkozhat az adatbázisra. Így a program független lesz az aktuálisan használt adatbázis nevétől és pontos elérési útvonalától.
  - Lehetővé teszi adatbázis-kapcsolatok cache-elését. Ennek előnye, hogy nem kell mindig új adatbázis-kapcsolatot létrehozni, ha igény van rá, mert ennek felépítése hosszú ideig is eltarthat, elég azt a cache-ből kérni.
  - Lehetővé teszi a Java Tranzakció API (JTA) kétfázisú protokolljának használatát.
  - Lehetővé teszi adatbázistáblák kapcsolat nélkül történő kezelését is. Ilyenkor az elvégzett változtatások csak akkor kerülnek az adatbázisba, ha újra rákapcsolódunk az adatbázis szerverre.

A JDBC API szolgáltatásait három csoportba szokás sorolni:

- Összekapcsolódás relációs adatbázissal.
- SQL utasítások végrehajtása.
- SQL lekérdezések eredményeinek feldolgozása.

A JDBC használatával a Java adatbázis-kezelő programok platform- és adatbázis-kezelő függetlenek lehetnek.

5.1.1. Két- és háromrétegű adatbázis-elérési modell

A JDBC a következő adatbázis-elérési modelleket támogatja:

- **Kétrétegű modell:** a program közvetlenül az adatbázis-kezelő rendszerrel kommunikál. Maga az adatbázis akár másik gépen is elhelyezkedhet, mint ahol a program fut, az adatforgalom pedig hálózaton keresztül folyik. Ezt az esetet kliens-szerver konfigurációnak nevezik, ahol az adatbázist tároló gép a szerever, a programot futtató gép pedig a kliens.
- **Háromrétegű modell:** a program adott protokollon keresztül egy olyan szolgáltató réteggel kommunikál, ami az adatbázis-kezelő rendszer és a futó program között helyezkedik el. Ez a réteg a programtól kapott parancsokat értelmezi, átalakítja, majd továbbítja azokat az adatbázis-kezelő rendszerhez. A lekérdezési eredményeket a program szintén a szolgáltató rétegen keresztül kapja meg. Ezen közbülső réteg bevezetése lehetővé teszi az adatbázis hozzáférések könnyű ellenőrzését és optimalizálását is. A szolgáltató réteg Java implementációja esetén az adatbázissal JDBC-n keresztül történik a kommunikáció.

5.1.2. JDBC meghajtóprogramok

A JDBC hívások végrehajtásakor mindig fizikailag is fel kell venni a kapcsolatot a felhasznált adatbázissal. Mivel ezen adatbázis akármilyen típusú, azaz az adatbázis-kezelő bármilyen szoftver lehet, ezért minden adatbázis-kezelő esetén külön biztosítani kell a JDBC hívások megfelelő értelmezését és kiszolgálását. Ezt a feladatot a JDBC meghajtóprogramok végzik el.

Egy JDBC meghajtóprogram valósítja meg a JDBC hívásokat egy adott adatbázis típushoz a Driver interfészt implementálva. A meghajtóprogramok a következő négy csoportba sorolhatók be:

1. **JDBC-ODBC áthidalóprogram és ODBC meghajtóprogram:** már létező ODBC meghajtóprogram használatát teszi lehetővé JDBC hívások kiszolgálására.
2. **JDBC-saját kliens-API áthidaló/meghajtóprogram:** a meghajtóprogram a JDBC hívásokat közvetlenül átalakítja a megfelelő adatbázis kliens-API hívásaira. Ebben az esetben minden kliens gépen ott kell lennie a megfelelő adatbázis kliens-API-t megvalósító bináris programnak.
3. **JDBC-hálózati protokoll Java meghajtóprogram:** a Javaban írt hordozható meghajtóprogram a JDBC hívásokat adatbázis független hálózati protokoll hívásokká (például RMI, vagy CORBA) alakítja, melyeket egy megfelelő szerverprogram értelmez és alakít át az adott adatbázis-kezelő API-jának hívásaivá. Ebben az esetben tehát a JDBC kliens nem közvetlenül az adatbázissal, hanem egy külön szerverprogrammal kommunikál, és csak ezen szerverprogram tart fenn direkt kapcsolatot az adatbázissal. Ez megfelel a háromrétegű adatbázis-elérési modellnek.
4. **JDBC-saját protokoll Java meghajtóprogram:** szintén Javaban írt meghajtóprogram, amely a JDBC hívásokat közvetlenül a megfelelő adatbázis-kezelő adatmanipulációs protokolljának hívásaivá alakítja át. Ebben az esetben nincs szükség közbülső szerverprogramra.

Az első két meghajtó típus alkalmazásakor a felhasznált bináris kisegítő programok miatt a Java program elveszti hordozhatóságát. Mivel a 3. és 4. típusú meghajtóprogramok Javaban íródtak, ezért az azokat használó Java programok platformfüggetlenek maradnak, de a 4. típusú meghajtó program csak egy adott típusú adatbázis-kezelővel tud kommunikálni. Ezért platform- és adatbázis-kezelőfüggetlen megoldás esetén csak a 3. típusú meghajtó programok és a háromrétegű adatbázis-elérési modell jöhet szóba, ahol a kiszolgáló réteg adatbázis-kapcsolatát a kétszintű adatelérési modellnek megfelelő 1., 2., vagy 4. típusú meghajtó programokkal lehet megvalósítani.

5.1.3. A JDBC előnyei az ODBC-vel szemben

Az ODBC jelenleg az egyik legelterjedtebb adatbázis hozzáférési API, és Javaból is lehet használni a hívásait, de JDBC használata a következő előnyökkel jár:

- Az ODBC C stílusú interfésszel rendelkezik, az ezt megvalósító natív C/C++ metódusok pedig csökkentik a Java programok megbízhatóságát, hordozhatóságát és biztonságosságát.
- A JDBC-interfész az ODBC-vel szemben teljesen objektumorientált.
- Az ODBC meghajtó programok platformfüggőek, azokat minden kliens gépre manuálisan kell telepíteni, míg a JDBC, mivel Javaban íródott, teljesen platformfüggetlen és hordozható meghajtó programokat használ, melyek a dinamikus osztálybetöltés miatt telepítés nélkül, automatikusan kerülnek felhasználásra.

Ha egy adatbázishoz még nem létezik JDBC meghajtó program, de ODBC már igen, akkor használni lehet a JDBC-ODBC áthidaló programot, amely lehetővé teszi a JDBC API használatát ODBC-meghajtóprogram esetén is. Mivel mid az ODBC, mind a JDBC az X/Open SQL CLI (Call Level Interface – hívási szintű interfész) SQL-interfészszabványra épül, ezért könnyű az áttérés ODBC-ről JDBC használatára. A JDBC tekinthető az ODBC objektumorientált megfelelőjének.

5.1.4. JDBC használata appletekben/servletekben

A JDBC egyik felhasználási területe a böngészőprogramokkal történő adatlekérdezés és módosítás appletek/servletek segítségével. Servletek esetén csak maga a servlet használja a JDBC-t adatbázis-elérésre a szerveroldalon, a böngészőprogram dinamikusan generált HTML oldalak, illetve HTML űrlapok formájában olvashatja, illetve módosíthatja az adatokat. Appletek

használatakor a felhasználó közvetlenül hozzáférhet az adatbázishoz JDBC-n keresztül. Ennek egyetlen feltétele, hogy a kliens oldalról elérhető legyen a használt adatbázis-meghajtóprogram kódja. Mivel egy hálózati applet alapértelmezés szerint csak a kódját tartalmazó géppel hozhat létre hálózati kapcsolatot, ezért csak olyan adatbázissal kommunikálhat, amely ugyanazon szerveren fut, ahonnan az applet kódja is letöltésre került. Többrétegű elérési modell esetén csak a kiszolgáló rétegnek kell ugyanazon a gépen lennie, mint az applet kódjának, maga az adatbázis, akár más gépen is lehet, az applet mégis képes lesz azt a kiszolgáló rétegen keresztül elérni.

Applet futtatásakor a felhasznált adatbázis-meghajtóprogram osztályainak letölthetőeknek kell lenniük ugyanonnan, ahonnan az applet is származik.

5.1.5. Kapcsolat felvétele az adatbázissal

Az adatbázis-kapcsolat felvételének a menete a DriverManager getConnection metódusának meghívása, melynek paramétere az elérni kívánt adatbázis-URL cím (és opcionálisan egy felhasználói azonosító és egy jelszó). Ekkor a DriverManager sorban megnézi, hogy a regisztrált meghajtó programok közül melyik tudja a kapott adatbázis-URL-t feldolgozni, majd az első ilyen meghajtó programnak meghívja a connect metódusát. A keresés sorrendje a meghajtó programok regisztrációs sorrendjével egyezik meg, ahol a jdbc.drivers rendszerparaméterben megadott meghajtó programok megelőznek minden közvetlenül regisztrált meghajtó programot. A DriverManager biztonsági okokból egy adott programnak csak olyan meghajtó program használatát engedélyezi, amely a lokális gépen helyezkedik el, vagy ugyanarról a címről került letöltésre, mint ahonnan maga a program.

A DriverManager megkerülhető, ha közvetlenül a kívánt meghajtó program connect metódusának direkt meghívásával vesszük fel a kapcsolatot.

5.1.6. Tranzakciókezelés

Egy tranzakció SQL utasítások végrehajtásából áll, melyek eredményét vagy véglegesítjük az adatbázisban a commit metódus meghívásával, vagy visszavonunk minden változtatást a rollback metódussal, visszaállítva ezzel az adatbázis eredeti állapotát. Így egy tranzakció addig tart, amíg le nem zárjuk azt egy commit, vagy egy rollback utasítással. Ezután automatikusan megkezdődik a következő tranzakció.

Egy új adatbázis-kapcsolat alapértelmezés szerint automatikus nyugtázási móddal jön létre, azaz minden SQL utasítás befejeződése után automatikusan meghívódik a commit metódus. Egy SQL utasítás akkor fejeződik be, ha teljesen végrehajtódott és nem ad vissza eredményt, vagy ha az SQL utasítást tartalmazó SQL objektumot újra végrehajtjuk. Eredménytáblát visszaadó SQL utasítás pedig akkor fejeződik be, ha már az eredmény utolsó sorát is feldolgoztuk, vagy ha az eredménytáblát lezárjuk. Ha az automatikus nyugtázási módot kikapcsoljuk, akkor a programnak magának kell vezérelnie a tranzakciókezelést a kommit és a rollback metódusok segítségével.

Egy többfelhasználós adatbázis-kezelő rendszer esetén előfordulhat, hogy az egyidejűleg futó tranzakciók valamilyen módon zavarják egymást. Például, ha az egyik tranzakció olyan adatot akar olvasni, amit egy másik tranzakció már megváltoztatott, de még nem véglegesített. Az ilyen konfliktusok feloldására szolgálnak a tranzakció-izolációs szintek, melyek az adatbázis viselkedését szabályozzák hasonló problémák fellépésekor.

A Connection interfész öt tranzakció-izolációs szintet definiál:

- **TRANSACTION_NONE:** Nincs tranzakciókezelés.
- **TRANSACTION_READ_UNCOMMITED:** Olvasáskor mindig az aktuális érték kerül felhasználásra.
- **TRANSACTION_READ_COMMITED:** Olvasáskor csak már véglegesített érték kerül felhasználásra.
- **TRANSACTION_REPEATABLE_READ:** A tranzakció ideje alatt az általa olvasott értékek más tranzakciók esetleges módosító hatása ellenére is mindig megegyeznek a tranzakció kezdetekor érvényben lévő értékekkel.

- **TRANSACTION_SERIALIZABLE:** A tranzakció ideje alatt az általa olvasott értékeket más tranzakciók nem írhatják felül.

A fenti sorrend megegyezik a szintek sorrendjével, ahol az első a legalacsonyabb tranzakció-izolációs szint. Minél magasabb ez a szint, annál lassabb lesz az SQL végrehajtás, hiszen az adatbázis-kezelő rendszernek annál több adminisztrációt kell elvégeznie minden egyes művelet végrehajtása során. A tranzakció-izolációs szintet a setTransactionIsolation metódussal lehet beállítani. Ha tranzakció közben módosítjuk a tranzakció-izolációs szintet, akkor a változtatás előbb automatikusan véglegesíti az aktuális tranzakciót, és csak azután változik meg az izolációs szint.

5.1.7. Információ a kapcsolatról

Egy adott adatbázis minden jellemzőjéről információt a DatabaseMetaData interfész felhasználásával lehet lekérdezni. Az adatbázis-kapcsolat adatbázisát leíró objektumot a getMetaData metódussal lehet megkapni. A lekérdezett információk egy része táblázatszerű, ezeket egy, az SQL lekérdező utasítások eredményeit is reprezentáló eredménytábla fogja tartalmazni. Az információk nagy része adott SQL fogalmak adatbázisspecifikus megvalósítását/korlátait adaja vissza. Az ezeket lekérdező metódusok nevei mindig get-tel kezdődnek. Az információk másik része pedig azt adja meg, hogy az adott adatbázis képes-e valamilyen művelet végrehajtására. Ezen metódusok nevei mindig supports-szal kezdődnek.

A lekérdező metódusok nagy részében szöveges keresési paramétereket is meg lehet adni. Ha itt null-t adunk meg, akkor a megfelelő paraméter nem fog szerepelni a keresés kritériumában. Üres szöveg megadásával pedig olyan keresést írunk elő, ahol az adott paraméter nem rendelkezhet semmilyen értékkel sem. Egyéb szöveges kritérium keresési mintaként szolgál, ahol a ’%’ jel helyén akármilyen, akár nulla hosszúságú szöveg, míg az ’_’ karakter helyén egy darab tetszőleges karakter állhat.

5.1.8. Hibakezelés

Ha az adatbázis-kapcsolat során valamilyen hiba lép fel, akkor egy SQLException kivétel váltódik ki. Ez a kivétel a fellépett hibákról a következő információkat tartalmazza:

- A hiba szövegét, melyet a getMessage metódussal lehet lekérdezni.
- Az X/Open SQLstate konvencióban megadott SQLstate szöveget.
- A hiba kódját, ami meghajtóprogram-függő, és általában az adatbázis által visszaadott hibakódnak felel meg.
- Hivatkozást a következő SQLExceptionra. Ezzel lehet az aktuális hibaüzenethez hozzárendelt esetleges kiegészítő információkat kezelni.

Az előforduló figyelmeztetések kezelését az SQLWarning kivételosztály végzi. Ilyen kivételek nem szakítják meg a program futását, mert a megfelelő metódusok maguk elkapják, és az aktuálisan végrehajtás alatt lévő SQL objektumhoz láncolják őket. A fellépő figyelmeztetéseket a getWarnings metódussal lehet lekérdezni, míg a clearWarnings metódussal lehet törölni őket. Lekérdezéskor mindig az első figyelmeztető üzenetet kapjuk, ehhez láncban van felfűzve a többi hibaüzenet.

5.1.9. Kapcsolat lezárása

A kapcsolatot lezáró close metódus felszabadítja az adatbázis-kapcsolat által lefoglalt JDBC erőforrásokat. Ez a metódus a kapcsolatobjektum megsemmisítésekor és bizonyos fatális hibák fellépésekor automatikusan meghívódik.[10]

Összegzés

A webáruház jelenleg működőképes. Adatokkal való feltöltés után weben publikálható.

Az appletre vonatkozó tervem az volt, hogy bemutassam a dokumentumok dinamikus betöltését frame-ekbe. A könyvkatalógus applet ezt maradéktalanul megvalósítja. Könyvek borítóját lehet megtekinteni a segítségével, és a könyvekre vonatkozó információk mennyisége is könnyen bővíthető. Az applet további fejlesztéssel esztétikailag még kellemesebb benyomást tudna kelteni a felhasználóban.

A szkripteket arra használjuk, hogy a segítségükkel honlapunkat eseményvezérelt programozási eszközökkel tegyük gazdagabbá, így reagálni tudunk a felhasználók interakcióira. Minden vásárlás tulajdonképpen négy fázisból áll. Egy kiszemelt terméket beteszünk a kosarunkba, ha esetleg meggondolnánk magunkat (mert nem tetszik, vagy esetleg nincs elég pénzünk rá), akkor visszatehetjük a polcra. Bármikor megnézhetjük mit tettünk bele a kosárba. Végül ha végeztünk nem marad más dolgunk, mint fizetni. Az általam készített on-line könyváruházban mindezt meg lehet tenni.

A webáruháznak készül az adatbázis kapcsolattal való kiegészítése. Ezzel tetszőleges adatbázishoz kapcsolódhatunk. Ott a termékek adatait kezelhetjük, és nincs szükség arra, hogy ezeket a honlapba „bedrótozva” tartsuk nyilván.

A további fejlesztés nyomvonala függhet még a jövőbeni felhasználói szükségletektől.

Irodalomjegyzék

[1] Matt Zandstra: Tanuljuk mag a PHP használatát 24 óra alatt
[2] Marshall McLuhan: The Gutenberg Galaxy: The Making of Typographic Man, University of Toronto Press, Toronto, 1962
[3] Virginia deBolt: HTML és CSS, webszerkesztés stílusosan, Kiskapu kiadó, Debrecen, 2005
[4] Java documentation, http://java.sun.com/j2se/1.5.0/docs/api
[5] Nyékiné G. Judit: Java 2 utikalauz programozóknak, ELTE TTK Hallgatói Alapítvány, Budapest, 2001, 16. fejezet
[6] JavaScript dokumentáció, http://www.intermedia.c3.hu/javascript/main.html
[7] Nyékiné G. Judit: Java 2 utikalauz programozóknak, ELTE TTK Hallgatói Alapítvány, Budapest, 2001, G. fejezet
[8] ODBC leírás, http://pcforum.hu/szotar/?term=ODBC&tm=miaz
[9] Adatbázis elérése Javaból, http://www.prog.hu/cikkek/9/Adatbazis+elerese+Java-bol+JDBC.html
[10] Nyékiné G. Judit: Java 2 utikalauz programozóknak, ELTE TTK Hallgatói Alapítvány, Budapest, 2001, 25. fejezet