K. Fejezet

Servletek

Egy *servlet* valójában egy speciális Java program, amely egy webszerverrel szorosan együttműködve a szerveroldalon lehetővé teszi HTML oldalak dinamikus létrehozását és paraméterezését különböző átviteli (például HTTP) protokollokon keresztül. Azt is mondhatjuk, hogy egy servlet ugyanaz a webszervernek, mint egy applet a böngészőprogramnak, csak nem rendelkezik grafikus felhasználói felülettel. De erre nincs is szükség, mivel a servletek – mint a név is mutatja – csakis szerverfunkciókat képesek elvégezni. Pontosabban a webszerver funkcionalitását lehet velük kibővíteni, mert ha egy kliens olyan HTML oldalt kér a webszervertől, melyet egy servlet állít elő, akkor a webszerver tulajdonképp csak delegálja a kérést a servlet felé, majd a servlet által generált oldalt továbbítja a kliensnek.

> *Megjegyzés*: Ezen fejezet a HTML és a HTTP protokoll alapszintű ismeretét feltételezi.

A javax.servlet csomag tartalmazza az összes servletspecifikus osztályt és interfészt. Minden servlet vagy a javax.servlet.GenericServlet leszármazottja, vagy maga implementálja a javax.servlet.Servlet interfészt, amely az általános servlet életciklus metódusokat specifikálja. Mi a továbbiakban csakis olyan servletekkel foglalkozunk, melyek a WWW-n legjobban elterjedt *HTTP* (HyperText Transfer Protocol) átviteli protokollt használják a klienssel való kommunikáció megvalósítására. Az ilyen típusú servletek mindig a javax.servlet.http.HttpServlet leszármazottai. A javax.servlet.http alcsomag a HTTP átviteli protokollhoz kapcsolódó speciális szolgáltatások elérését biztosító típusokat tartalmazza.

A HTML oldalak dinamikus generálására használt CGI (Common Gateway Interface) megoldásokkal szemben a servletek a következő előnyökkel rendelkeznek:

- A Java nyelv miatt platformfüggetlen megoldást biztosítanak a rendszerint platformfüggő CGI-szkript megvalósításokkal szemben.
- A webszerveren belül állandóan futó virtuális gép sokkal gyorsabb kiszolgálást eredményez, mivel elmarad a minden egyes kérés kiszolgálásához szükséges külső programindítás.
- A webszerveren belül állandóan futó virtuális gép miatt leegyszerűsödik a kéréskiszolgálások szinkronizációja.
- Egy servlet átirányíthatja a kérést egy másik servlethez, ezáltal dinamikusan lehet alkalmazkodni a webszerver terheltségéhez.
- Egy servlet könnyen megőrizhet információkat az azonos helyről jövő egymás utáni kérések kiszolgálása között.

K.1. Servlet fejlesztőkörnyezet

A webszerver és a servlet közti kommunikáció a servlet API-n keresztül történik. Servleteket tehát csakis olyan webszerverekkel lehet használni, melyek támogatják ezen API-t, és természetesen képesek Java programok futtatására. A servlet API nem része a standard JDK-nak, de letölthető hozzá különálló fejlesztőkörnyezet (*JSDK* - Java Servlet Development Kit) a [SM-JSDK 01] címről. Ezen fejlesztőkörnyezet jelenlegi legfrissebb verzióját (2.1) fogjuk a továbbiakban ismertetni.

A fejlesztőkörnyezet tartalmazza a servlet API implementációját, HTML dokumentációját, néhány példaprogramot, valamint egy egyszerű Java alapú webszervert, melynek segítségével akkor is kipróbálhatjuk servleteinket, ha nincs a gépünkön más webszerver telepítve.

K.2. Fordítás

A servlet API implementációját a servlet.jar fájl tartalmazza. Servlet fordításakor ezen fájlnak elérhetőnek kell lennie az osztályelérési útvonalon keresztül. A lefordított servlet bájtkódját a webszerver előre megadott alkönyvtárába kell másolni. Ennek pontos elhelyezkedése a webszerver konfigurációjától függ, de rendszerint egy servlets nevű alkönyvtárat szoktak használni erre a célra.

K.3. Futtatás

Mivel egy servlet csak egy webszerverrel együtt képes működni, ezért a servlet futtatása a webszerver feladata. Erre akkor kerül sor, ha a webszerver olyan kérést kapott, melyet a servletnek kell feldolgoznia. Ez rendszerint a kért URL alapján könnyen megállapítható, mert az a servleteket tartalmazó útvonalat (általában /servlet), valamint a futtatandó servlet nevét tartalmazza. Ilyenkor a kért servlet szolgálja ki a kliens kérését, a webszerver pedig a servlet válaszát adja vissza HTML oldalként. Tehát a kliensoldal legfeljebb a kért URL alapján jöhet rá, hogy egy servlettől kapott választ, mivel a kliens szemszögéből semmi más különbség nincs egy statikus HTML oldal, illetve egy servletet megadó URL lehívása között.

A JSDK részét képező Java webszervert például a startserver szkripttel indíthatjuk el. Ezek után a http://localhost:8080/ címet megtekintve egy tetszőleges böngészőprogrammal elolvashatjuk a JSDK API leírását, valamint kipróbálhatunk egy pár példaservletet.

K.4. Egy servlet életciklusa

A servlet életciklusa alatt a következő három eseményt értjük:

- Servlet példányosítása.
- A servlet kiszolgál egy kliens kérést.
- Servlet megszüntetése.

Ezen eseményekhez tartozó metódusokat a Servlet interfész specifikálja és a webszerver hívja meg.

K.4.1. Servlet példányosítása

Miután a webszerver létrehozta a servlet egy példányát, meghívja annak public void init(ServletConfig) metódusát. A paraméterként kapott ServletConfig objektumot el kell menteni, mivel a Servlet interfész getServletConfig metódusa pontosan ezen objektum visszaadását követeli meg. A GenericServlet osztály emiatt bevezette az init metódus paraméter nélküli változatát, melyet a paraméteres változat automatikusan meghív a kapott paraméter elmentése után, így nekünk már nem kell foglalkoznunk a ServletConfig mentésével, ha az init metódus ezen paraméter nélküli kényelmi változatát definiáljuk felül.

Megjegyzendő, hogy teljesen a webszervertől függ, mikor és hány példányt hoz az létre egy servletből. Egyedül csak az a biztos, hogy adott servletnek szóló kérés kiszolgálása előtt legalább egy servletpéldány létre lett hozva, valamint az init példánymetódus

meghívása csak egyszer, de a két másik életciklus metódus előtt történik. A példányosítás történhet a webszerver indulásakor, vagy közvetlenül az első kérés kiszolgálásakor, vagy bármikor ezen két időpont között. Általában a webszerver egy servletből annyi példányt hoz létre, ahány különböző néven magát a webszervert el lehet érni (=virtuális hosztok száma). Tehát nagy valószínűséggel kijelenthető, hogy a webszerverek legtöbbje egy servletet csak egyszer példányosít. Arra is figyeljünk, hogy ha egy servlet osztályának kódja egyszer már betöltésre került, akkor annak változását rendszerint nem veszi észre a servlet futtatókörnyezet. Épp ezért egy servlet fejlesztése során a servlet egy újabb változatának kipróbálása előtt érdemes a webszervert újraindítani.

A servlet inicializálásakor érdemes a servlet teljes futási ideje alatt használt mezőket beállítani, például adatbázis-kezelő servlet esetén itt érdemes megnyitni az adatbázis-kapcsolatot. Ha már ekkor kiderül, hogy valami miatt nem fogja tudni a servlet az elvárt szolgáltatásokat nyújtani (például hiba az adatbázis-kapcsolat megnyitásakor), akkor ezt jelezhetjük a servlet futtató környezetének UnavailableException kivétel kiváltásával. Tehát ne a System.exit metódust használjuk!

K.4.2. Servletparaméterek

Servletparamétereket a servletek konfigurálására, illetve inicializálására lehet felhasználni. A paraméterek megadási formája *paraméternév*=*paraméterérték* alakú (vesd össze: java.util.Properties.load), ezen beállításokat pedig rendszerint egy servlets.properties nevű fájl tartalmazza. A paraméterfájlt a webszerver általában csak az indításakor, egyszer olvassa be, ezért az esetleges változtatások csak a webszerver újraindításakor lépnek életbe.

Jelenleg két servletparamétert használ a servletfuttató környezet:

- Servlet neve : *servletnév*.code=*servlet osztályának teljes neve* - ennek beállítása után a megadott névvel lehet hivatkozni a servletre. Ezen név a webszerverhez küldött URL-ben, illetve további servletparaméterek megadásakor is használható. Ha nem adunk meg nevet egy servlethez, akkor alapértelmezés szerint annak neve meg fog egyezni a servlet osztályának nevével.
- Servlet inicializációs paraméterei : *servletnév*.initArgs=*paraméterek* - az így megadott paramétereket a servlet megkapja az inicializálásakor. A *paraméterek* megadása *paraméternév*=*paraméterérték* értékpárok vesszővel elválasztott listájával történik. Ilyenkor ügyeljünk arra, hogy logikailag mindezt egy sorba írva kell megtenni. Tehát ha több sorban szeretnénk az inicializációs paramétereket megadni, akkor minden közbülső sor végére tegyünk ki egy \ jelet.

A servlet inicializáláskor kapott ServletConfig paraméteren keresztül a servlet inicializációs paramétereit (a getInitParameter és getInitParameterNames metódusokkal), valamint a servletet futtató környezetet reprezentáló ServletContext objektumot lehet lekérdezni (a getServletContext metódussal). Mivel a GenericServlet automatikusan kezeli a ServletConfig objektumot, ezért ezen három metódus magára a servletre is meghívható. A ServletContext-en keresztül a servlet futtatókörnyezet szolgáltatásait lehet elérni, például annak log metódusával lehet a naplófájlba üzeneteket kiírni. A GenericServlet log metódusa alapértelmezés szerint a ServletContext azonos nevű metódusát hívja meg.

K.4.3. Kliens kiszolgálása

Miután a webszerver inicializálta a servlet egy példányát, az képessé válik a kliens kérések kiszolgálására. Ha a webszerver megállapította, hogy egy kérés egy servletet céloz meg, meghívja annak public void service(ServletRequest, ServletResponse) metódusát. Az első paraméterként kapott objektumon keresztül a kliens-servlet, míg a második paraméterként kapott objektumon keresztül a servlet-kliens irányú kommunikáció valósítható

meg. A servlet tehát úgy szolgál ki egy kliens kérést, hogy feldolgozza az esetleges kapott paramétereket, melyek rendszerint HTML űrlapok használata esetén az űrlap mezőinek értékeit tartalmazzák, majd dinamikusan legenerál (általában) egy HTML oldalt, melyet a kliens böngészőprogram megjelenít. Ilyenkor vegyük figyelembe, hogy servletünk rendszerint csak akkor lesz újra meghívva, ha a kliens böngészőprogram nem cache-eli be a válaszunkat, tehát érdemes a generált válasz cache-elését letiltani.

A ServletRequest interfész a klienskérés paramétereinek (getParameterNames, getParameter, getParameterValues) és jellemzőinek (getContentLength, getContentType, getCharacterEncoding, getProtocol, getScheme), a kliens gép címének (getRemoteAddr, getRemoteHost), a kérést kiszolgáló szerver gép címének (getServerName, getServerPort), valamint a kérés tartalmának olvasást lehetővé tevő (getInputStream, getReader) metódusokat definiál. A kérés tartalmát adatfolyamként egy ServletInputStream objektum reprezentálja. HttpServlet esetén a klienskérést egy HttpServletRequest objektum tartalmazza. Ezen interfész a ServletRequest interfészt a HTTP fejléc mezőinek olvasásához, valamint a HTTP specifikus szolgáltatások eléréséhez szükséges metódusokkal bővíti ki. Megjegyzendő, hogy a paraméterek feldolgozását vagy mi magunk végezzük el a kérés adatfolyamát értelmezve, vagy ezt bízzuk rá a servlet futtatókörnyezetre és csak a getParameter metódusokat (vagy HttpServletRequest esetén a getQueryString metódust) használjuk. Ugyanazon kliens kérés paramétereinek lekérdezésekor ezen két eljárás nem alkalmazható egyszerre!

HTML űrlapok tartalmát szintén paramétereken keresztül kapja meg a servlet. A paraméter neve a HTML űrlap beviteli mezőjének nevével (NAME attribútum) egyezik meg, a paraméter értéke pedig a beviteli mező tartamát veszi fel. Ebből következik, hogy csakis a névvel rendelkező beviteli mezők értékét tudja átvenni a servlet. Ha egy beviteli mező üres (vagy például checkbox és választós lista esetén nincs kiválasztva semmi sem), akkor előfordulhat, hogy az adott beviteli mező nem is adódik át paraméterként. Rádiógomb és checkbox esetén, ha az ki van választva, a megfelelő nevű paraméter értéke a mező VALUE attribútuma, vagy annak hiányában "on" lesz. Listák esetén a paraméter értéke(i) a kiválasztott elem(ek) VALUE attribútumával, vagy annak hiányában az elem(ek) feliratával fog(nak) megegyezni. Megjegyzendő, hogy a paraméterek értéke HTML űrlapok esetén automatikusan kódolódik, de ha mi magunk egy űrlapon kívül (például hyperlink paramétereként) akarunk paramétereket átadni, a kódolásról nekünk kell gondoskodni a java.net.URLEncoder.encode metódus segítségével.

A ServletResponse interfész a servlet válaszának jellemzőit beállító (setContentLength, setContentType), valamint a válasz kiírását lehetővé tevő metódusokat (getOutputStream, getWriter) definiál. A válasz tartalmát adatfolyamként egy ServletOutputStream objektum reprezentálja. HttpServlet esetén a servlet válaszát egy HttpServletResponse objektum tartalmazza. Ezen interfész a ServletResponse interfészt a HTTP fejléc mezőinek beállításához, valamint a HTTP specifikus szolgáltatások eléréséhez szükséges metódusokkal bővíti ki. A servlet válaszát csak azután küldi el a webszerver, miután lezártuk a választ reprezentáló adatfolyamot annak close metódusával. Ha még a teljes adatfolyam lezárása előtt szeretnénk, hogy a kliens böngészőprogram megkezdje az addig generált válasz megjelenítését, akkor használjuk a ServletOutputStream flush metódusát. Ekkor a böngésző a válasznak csak azon prefixét képes megjeleníteni, melynek grafikus megjelenítése már nem fog megváltozni, tehát például ha szöveg esetén lezártuk azt egy sorvége jellel. Megjegyzendő, hogy HttpServletResponse esetén a HTTP mező fejléceit csakis a válasz adatfolyamhoz történő első hozzáférés előtt szabad állítani. Ha a válasz hosszát előre tudjuk, akkor érdemes azt beállítani a setContentLength metódussal, mivel ilyenkor a webszerver és a kliens közti adatátvitel sokkal gyorsabb lesz.

A HttpServlet már maga implementálja a service metódust, és a HTTP klienskéréseket azok HTTP típusától függően a következő metódusokhoz továbbítja:

- doDelete(HttpServletRequest, HttpServletResponse) - DELETE típusú HTTP kérések

kezelésekor hívódik meg. Ilyen típusú kérésekkel dokumentumok törlését lehet kérni a webszerveren.

- doGet(HttpServletRequest, HttpServletResponse) - GET típusú HTTP kérések kezelésekor hívódik meg. Ilyen típusú kérésekkel a kért dokumentumnak paramétereket lehet megadni, melyek az URL végén levő paramétersztring formájában kerülnek átadásra.
- doOptions(HttpServletRequest, HttpServletResponse) - OPTIONS típusú HTTP kérések kezelésekor hívódik meg. Alapértelmezés szerint a servlet által támogatott HTTP kéréstípusokat adja vissza, így csak akkor kell felüldefiniálni, ha a HTTP1.1-es kéréstípusokon kívül más kéréseket is kezelni tud a servlet.
- doPost(HttpServletRequest, HttpServletResponse) - POST típusú HTTP kérések kezelésekor hívódik meg. Ilyen típusú kérésekkel a kért dokumentumnak paramétereket lehet megadni, melyek nem látható módon a HTTP fejlécben kerülnek átadásra.
- doPut(HttpServletRequest, HttpServletResponse) - PUT típusú HTTP kérések kezelésekor hívódik meg. Ilyen típusú kérésekkel dokumentumok feltöltését (FTP-hez hasonlóan) lehet kérni a webszerverre.
- doTrace(HttpServletRequest, HttpServletResponse) - TRACE típusú HTTP kérések kezelésekor hívódik meg. Alapértelmezés szerint a kérés fejlécében található adatokat adja vissza.

A HttpServlet osztály kibővítésekor tehát csak azon do*típus* metódusokat kell felüldefiniálni, amilyen HTTP típusú kéréseket szeretnénk kezelni. Ez rendszerint a doGet és doPost metódusok implementálását jelenti, ráadásul úgy, hogy az egyik metódus tartalmazza a tényleges megvalósítást, míg a másik metódus erre a metódusra irányítja át a kérés feldolgozását. A feldolgozás menete pedig általában a kapott paraméterek kiolvasását, a válasz HTTP fejlécének beállítását, a válasz legenerálását, annak kiküldését a kliens adatfolyamra, majd az adatfolyam lezárásakor annak elküldését jelenti.

Többszálúság

Mivel a webszerverhez egy időben ugyanazon servletet megcímző több kérés is befuthat, ezért a klienst kiszolgáló service metódust egyszerre több programszál is végrehajthatja. Saját servlet implementációjakor tehát mindig ügyeljünk arra, hogy a kiszolgáló kód ne okozzon gondot, ha azt esetleg egyszerre több programszál is futtatja. Ha a kliens kérések szinkronizációját a servlet futtatókörnyezetre akarjuk bízni, akkor implementálnunk kell a SingleThreadModel interfészt. Ezen interfész nem tartalmaz egyetlen metódust sem, de ha implementáljuk, akkor a servlet futtatókörnyezet biztosítja, hogy a kiszolgáló service metódust egy időben legfeljebb csak egy programszál fogja meghívni. Ez a legegyszerűbb módja a bejövő kliens kérések sorbaállításának.

Kérés átirányítása

HTTP protokoll használata esetén lehetőségünk van a kliens kérést egy másik URL-hez átirányítani. A HttpServletResponse sendRedirect metódusával egy abszolút URL-t megadva ugyanazt a hatást érhetjük el, mintha a kliens nem is a servletünktől kért volna választ, hanem egyből az általunk megadott URL-t tekintette volna meg. Természetesen a kérés átirányítása után már nem küldhetünk adatot a kliens felé.

Hiba jelzése

HTTP protokoll használata esetén lehetőségünk van előredefiniált HTTP státusz- és hibakódokat küldeni a kliensnek. A HttpServletResponse sendError metódusaival egy HTTP

hibakódot adhatunk meg, ezen hibakódhoz tartozó hibaüzenetet fogja látni a kliens. Ilyenkor a metódus meghívása után már nem küldhetünk adatot a kliens felé.

Ha csak a HTTP státuszt akarjuk beállítani, akkor használjuk a HttpServletResponse sendStatus metódusát.

HTTP fejléc direkt kezelése

Ha a HTTP fejléc mezőit valamilyen okból mi magunk szeretnénk kezelni, akkor a kliens-kérés HTTP fejléc mezőihez a HttpServletRequest interfész getHeader, getHeaderNames, getDateHeader és getIntHeader metódusaival férhetünk hozzá, míg a válasz fejlécmezőit a HttpServletResponse setHeader, setDateHeader és setIntHeader metódusaival állíthatjuk be. Megjegyzendő, hogy a válasz fejléceinek mezőit csak azelőtt szabad módosítani, mielőtt adatot küldenénk a kliens felé.

K.4.4. Servlet megszüntetése

Egy servlet inicializálása után a webszerver úgy dönthet, hogy nincs többé szükség az adott servlet szolgáltatásaira, és megszüntetheti a servlet példányt. Ilyenkor a servlet futtatókörnyezet megvárja, míg az adott servletet használó minden még futó klienski-szolgáló programszál véget ér (vagy egy beállított időtartam, általában 30 másodperc le nem telik), és meghívja a servlet public void destroy() metódusát. Ezen metódus meghívása után már nem fog egyik életciklus metódus sem meghívódni. Előfordulhat, hogy ezen metódus már akkor meghívódik, amikor a servlet még egyetlen klienst sem szolgált ki, de az is elképzelhető, hogy a destroy meghívásának pillanatában esetleg még több kliens kiszolgálása is folyamatban van.

A túl hosszan tartó kiszolgálásból eredő problémára megoldás lehet, ha mindig nyilvántartjuk, hogy egyszerre hány programszál tartózkodik a kiszolgáló metódusban. A destroy meghívásakor ezt egy logikai változóval jelezzük a programszálak felé, majd addig várunk, míg a számlálónk segítségével meg tudjuk állapítani, hogy minden programszál befejeződött. A kiszolgálás menetét pedig felosztjuk apróbb részfeladatokra, melyek befejezése után mindig megnézzük, hogy közben esetleg meghívták-e a destroy-t. Így elérhetjük, hogy minden futó programszál a leggyorsabban reagál a servlet megszüntetésére, valamint a servlet megszüntetése előtt, miután már egyetlen programszál sem tartózkodik a kiszolgáló metódusban, biztonságosan felszabadíthatjuk a servletünk által foglalt erőforrásokat (például adatbázis-kapcsolat zárása).

A servlet megszüntetése után a servletpéldány kikerül a servlet futtatókörnyezet ellenőrzése alól. Ha esetleg újra szükség lenne egy már megszüntetett servletre, akkor az a servlet egy új példányának létrehozását és inicializálását vonja maga után.

K.4.5. 1. példa: a LifeTestServlet servlet

Ezen példaservlet az életciklus metódusainak meghívásakor a servlet futtatókörnyezet naplófájljába kiírja, hogy mely metódus lett meghívva, eddig hányszor lett a servlet példányosítva, valamint a klienskérésekre válaszképp a servlet visszaadja a kérés, illetve a servlet jellemzőit és paramétereit. A servlet azt is nyilvántartja, hogy egyszerre hány programszál tartózkodik a kiszolgáló metóduson belül. A kiszolgálást pedig egy tíz másodpercig tartó ciklussal direkt lassítjuk, hogy ki lehessen próbálni, hogy viselkedik a servlet, ha ezen időtartam alatt egy újabb kérést kell kiszolgálnia. A servlet megszüntetésekor a kiszolgáló metódus ezt érzékelvén előbb befejeződik, a destroy pedig addig vár, amíg minden esetleg még futó programszál be nem fejeződik.

```
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;

public class LifeTestServlet extends HttpServlet {

    static int példányszámláló;                                  //példányszámláló
    int példánysorszám = ++példányszámláló;                       //példánysorszám
    int futásszám;                                                     //futásszám

    public void init() {
           log(példánysorszám+": init");                   //elindulás naplózása
    }

    private int programszálszám;                         //programszálak számlálása
    private synchronized void programszálEleje() {                      //belépés
            programszálszám++;
    }
    private synchronized void programszálVége() {                       //kilépés
            programszálszám--;
            if (programszálszám == 0 && megszüntet()) notifyAll();
    }
    private synchronized int programszálSzám() {
            return programszálszám;
    }

    public void service(HttpServletRequest kérés,
                        HttpServletResponse válasz)
           throws ServletException, IOException {
       try {
           programszálEleje();
           log(példánysorszám+": service");
           válasz.setContentType("text/html");
           PrintWriter out = válasz.getWriter();
           out.println("<PRE>"+getClass().getName()+" adatai");
           out.println("Példányszámláló = "+példányszámláló);
           out.println("Példánysorszám = "+példánysorszám);
           out.println("Futásszám = "+(++futásszám));
           out.println("Programszálszám = "+programszálSzám());
           out.println();
           out.println("Inicializáló paraméterek");   //inicializáló paraméterek
           Enumeration e = getInitParameterNames();                  //listázása
           while (e.hasMoreElements()) {
                 String név = (String)e.nextElement();
                 String érték = getInitParameter(név);
                 out.println("   " + név + " = " + érték);
           }
           out.println(); out.flush();
           out.println("Kliens jellemzői");          //kliensjellemzők listázása
           out.println("Kliens címe = " + kérés.getRemoteAddr());
           out.println("Kliens gép = " + kérés.getRemoteHost());
           out.println(); out.flush();
           out.println("Szerver jellemzői");        //szerverjellemzők listázása
           out.println("Szerver neve = " + kérés.getServerName());
           out.println("Szerver portja = " + kérés.getServerPort());
           out.println(); out.flush();
```

```
        out.println("A kérés jellemzői");          //kérésjellemzők listázása
        out.println("Karakterkódolás = " + kérés.getCharacterEncoding());
        out.println("Kérés hossza = " + kérés.getContentLength());
        out.println("Kérés típusa = "+ kérés.getContentType());
        out.println("Kérés protokollja = "+ kérés.getScheme());
        out.println("Használt protokoll = "+ kérés.getProtocol());
        out.println(); out.flush();

        out.println("Kérés paraméterei");             //paraméterek listázása
        e = kérés.getParameterNames();
        while (e.hasMoreElements()) {
              String név = (String)e.nextElement();
              String[] érték = kérés.getParameterValues(név);
              out.print("   " + név + " = ");
              for (int i = 0; i < érték.length; i++)
                  out.print(érték[i] + " ");
              out.println();
        }
        out.println(); out.flush();

        out.println("HTTP fejléc");                   //HTTP fejléc listázása
        e = kérés.getHeaderNames();
        while (e.hasMoreElements()) {
              String név = (String)e.nextElement();
              String érték = kérés.getHeader(név);
              out.println("   " + név + " : " + érték);
        }
        out.println(); out.flush();

        out.println("HTTP jellemzők");             //HTTP jellemzők listázása
        out.println("Biztonság: " + kérés.getAuthType());
        out.println("HTTP típus: " + kérés.getMethod());
        out.println("Kérés útvonala: " + kérés.getPathInfo());
        out.println("Kérés útvonalának megfelelő útvonal: " +
                    kérés.getPathTranslated());
        out.println("Kérés paraméterei: " + kérés.getQueryString());
        out.println("Felhasználó: " + kérés.getRemoteUser());
        out.println("Kliensoldali kapcsolatazonosító: " +
                    kérés.getRequestedSessionId());
        out.println("Kérés URI: " + kérés.getRequestURI());
        out.println("Servlet útvonala: " + kérés.getServletPath());
        out.println(); out.flush();
        out.print("Időhúzás");                                    //időhúzás
        for (int i=0; i<10; i++) {
             if (megszüntet()) break;
             try {
                 Thread.currentThread().sleep(1000);
             } catch (InterruptedException ie) {}
             out.print("."); out.flush();
        }

        out.println("</PRE>"); out.flush();
    } finally {
      programszálVége();
    }
}
```

```
    private boolean megszüntetés;              //servlet megszüntetésre került-e
    private synchronized boolean megszüntet() {
            return megszüntetés;
    }
    private synchronized void megszüntet(boolean b) {   //servelet megszüntetése
            megszüntetés=b;
    }

    public synchronized void destroy() {
           megszüntet(true);
           while (programszálSzám() > 0) try {    //futó programszálak megvárása
                 wait();
           } catch (InterruptedException e) {}
           log(példánysorszám+": destroy");
    }
}
```

Fordítás

A fordításkor ügyeljünk arra, hogy a javax.servlet csomagot tartalmazó servlet.jar fájl elérhető legyen az osztálybetöltési útvonalon keresztül.

Futtatás

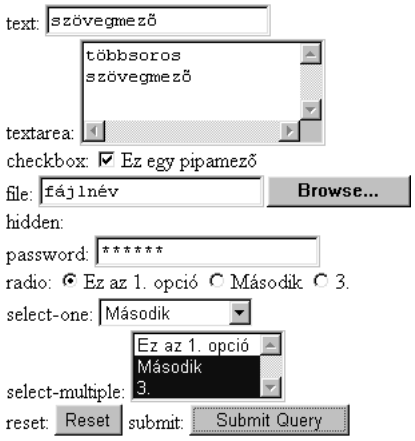
A következőkben a LifeTestServlet JSDK-val történő kipróbálásának menetét ismertetjük. Másoljuk be a lefordított bájtkód fájlt a JSDK webpages/WEB-INF/servlets alkönyvtárába. A webpages/WEB-INF alkönyvtárban levő servlets.properties fájlban adhatunk nevet és inicializációs paramétereket servletünknek. Ezután indítsuk el a webszervert a start-server szkripttel.

Servletünk teszteléséhez készítettünk egy HTML lapot, melyen HTML beviteli mezőket lehet kitölteni, majd azok tartalmát két különböző HTTP típussal (GET, POST) is el lehet küldeni.

```
<html>
<head>
<title>LifeTestServlet</title>
</head>
<body>
<h1>LifeTestServlet</h1>
<a href="LifeTestServlet.java">Íme a source</a><p>
<HR>
<h2>GET form</h2>
<FORM METHOD="GET" ACTION="http://localhost:8080/servlet/LifeTestServlet">
text:
<input type="text" name="text"><br>
textarea:
<textarea name="textarea" rows=3>
</textarea><br>
checkbox:
<input type="checkbox" name="checkbox" value="bubu">Ez egy pipamező<br>
file:
<input type="file" name="file"><br>
hidden:
<input type="hidden" name="hidden" value="hiddenvalue"><br>
password:
<input type="password" name="password"><br>
```

```
radio:
<input type="radio" name="radio" value="radio1">Ez az 1. opció
<input type="radio" name="radio" value="radio2">Második
<input type="radio" name="radio" value="radio3">3.
<br>
select-one:
<select name="select-one">
<option value=option1>Ez az 1. opció
<option value=option2>Második
<option value=option3>3.
</select><br>
select-multiple:
<select name="select-multiple" MULTIPLE>
<option value=option1>Ez az 1. opció
<option value=option2>Második
<option value=option3>3.
</select><br>
reset:
<input type="reset">
submit:
<input type="submit">
</FORM>
<HR>
<h2>POST form</h2>
<FORM METHOD="POST" ACTION="http://localhost:8080/servlet/LifeTestServlet">
text:
<input type="text" name="text"><br>
textarea:
<textarea name="textarea" rows=3>
</textarea><br>
checkbox:
<input type="checkbox" name="checkbox">Ez egy pipamező<br>
file:
<input type="file" name="file"><br>
hidden:
<input type="hidden" name="hidden" value="hiddenvalue"><br>
password:
<input type="password" name="password"><br>
radio:
<input type="radio" name="radio" value="radio1">Ez az 1. opció
<input type="radio" name="radio" value="radio2">Második
<input type="radio" name="radio" value="radio3">3.
<br>
select-one:
<select name="select-one">
<option value=option1>Ez az 1. opció
<option value=option2>Második
<option value=option3>3.
</select><br>
select-multiple:
<select name="select-multiple" MULTIPLE>
<option value=option1>Ez az 1. opció
<option value=option2>Második
<option value=option3>3.
</select><br>
reset:
<input type="reset">
```

```
submit:
<input type="submit">
</FORM>
</body>
</html>
```



K.1. ábra: A LifeTestServlet-et meghívó HTML oldal képe

Eredményül az alábbihoz hasonló tartalmú lapot kapunk:

```
LifeTestServlet adatai
Példányszámláló = 1
Példánysorszám = 1
Futásszám = 2
Programszálszám = 2

Inicializáló paraméterek

Kliens jellemzői
Kliens címe = 127.0.0.1
Kliens gép = localhost

Szerver jellemzői
Szerver neve = localhost
Szerver portja = 8080

A kérés jellemzői
Karakterkódolás = iso-8859-1,*,utf-8
Kérés hossza = -1
Kérés típusa = null
Kérés protokollja = http
Használt protokoll = HTTP/1.0

Kérés paraméterei
   checkbox = on
   password = jelszó
   select-one = option2
   text = szövegmező
   select-multiple = option2 option3
```

```
   textarea = többsoros
szövegmező
   hidden = hiddenvalue
   file = fájlnév
   radio = radio1

HTTP fejléc
   Connection : Keep-Alive
   User-Agent : Mozilla/4.6 [en] (WinNT; I)
   Pragma : no-cache
   Host : localhost:8080
   Accept : image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
   Accept-Encoding : gzip
   Accept-Language : en
   Accept-Charset : iso-8859-1,*,utf-8
   Cookie : SESSIONID=To1010mC34383544136087085At

HTTP jellemzők
Biztonság: null
HTTP típus: GET
Kérés útvonala: null
Kérés útvonalának megfelelő útvonal: null
Kérés paraméterei: text=sz%F6vegmez%F5&textarea=t%F6bbsoros%0D%0Asz%F6vegmez%F5
&checkbox=on&file=f%E1jln%E9v&hidden=hiddenvalue&password=jelsz%F3&radio=radio1
&select-one=option2&select-multiple=option2&select-multiple=option3
Felhasználó: null
Felhasználó: null
Kliensoldali kapcsolatazonosító: null
Kérés URI: /servlet/LifeTestServlet
Servlet útvonala: /servlet/LifeTestServlet

Időhúzás.........
```

Érdemes kipróbálni mindkét HTTP típusú beviteli űrlapot, valamint azt, hogy milyen paraméterértékek adódnak át, ha valamely mezőt üresen hagyjuk. Egy időben több kliensкérést elküldve pedig tesztelhetjük servletünk viselkedését, ha több programszál is futtatja annak kiszolgáló metódusát.

K.5. Információ a servletről

A Servlet interfész getServletInfo metódusának felüldefiniálásával a felhasználók számára információt lehet adni magáról az servletről, annak szerzőjéről és verziójáról. Ezen metódus eredményét webszerver-adminisztrációs programok használják az elérhető servletek leírására.

K.6. Információ megőrzése több klienskapcsolat alatt

Egy servlet csak az alatt az idő alatt van kapcsolatban a klienssel, amíg kiszolgálja annak kérését és vissza nem adja a generált válaszoldalt. Előfordulhat azonban, hogy adott klienstől jövő kérések egymástól nem függetlenek, azaz információt szeretnénk átadni a klienskiszolgálások között. Például ha egy servlet csak jelszó megadása esetén végezhet el egy adott feladatot, a jelszót csak akkor kellene bekérni és feldolgozni, amikor a kliens először veszi igénybe a servlet szolgáltatásait. Természetesen arra is ügyelni kell, hogy míg egy servletből rendszerint csak egy példány szolgál ki minden kérést, addig a kliensek több

példányban léteznek, teljesen különböző gépeken, azaz a servletnek valahogy különbséget kell tudni tennie a különböző kliensekhez tartozó éppen aktuális paraméterek közt. A servlet futtatókörnyezet ezen probléma megoldására két megoldást is ad.

K.6.1. Cookie-k használata

A legegyszerűbb megoldás, ha a klienshez tartozó adatokat maga a kliens tárolja, és azokat minden kérésekor elküldi a servletnek. Ehhez tehát az szükséges, hogy a servlet információt tudjon tárolni a kliens gépen. Erre valók az úgynevezett *cookie*-k. Ezek tulajdonképp szöveges információk, amelyeket a kliens böngészőprogram tárol el (általában egy cookies.txt fájlban), majd a szervernek elküldi azok tartalmát a klienskéréssel együtt. Egy cookie tehát mindig a kliensoldalon tárolódik, egy adott webszerverhez tartozik (így ha egy webszerveren több servlet is fut, akkor mindegyik olvashatja a másik által létrehozott cookie-kat) és a kliens küldi őket automatikusan a webszervernek (a HTTP fejlécben új mezőként). Általában a kliens programok egy adott webszerverhez legalább 20 darab, minimum 4 kilobájt hosszú cookie tárolását engedélyezik.

A cookie tartalmára annak nevével lehet hivatkozni. Mivel a név mint HTTP fejléc mező kerül felhasználásra, a cookie neve csak az alap ASCII karakterekből állhat, nem tartalmazhat szóköz karaktert, valamint a következő jeleket sem tartalmazhatja: `[ ] ( ) = , " / ? @ : ;`.

Mivel a cookie-k csak a HTTP protokoll esetén használhatóak, ezért minden cookie-val kapcsolatos típus és metódus a javax.servlet.http csomagban található. Egy cookie-t a Cookie osztály reprezentál, amelynek főbb jellemzői a következők:

- Név - a cookie neve. A kliens egyszerre több, azonos nevű cookie-t is küldhet. A nevet lekérdezni a getName metódussal lehet, beállítása pedig új cookie létrehozásakor a konstruktor segítségével történik.
- Érték - a cookie értéke. Egy cookie értéke tetszőleges szöveg lehet. Lekérdezése a getValue metódussal, beállítása pedig új cookie létrehozásakor a konstruktor segítségével, vagy később a setValue metódussal történik.
- Élettartam - a cookie élettartama másodpercekben megadva. Ha a beállított élettartam lejár, akkor a cookie megszűnik. Ezért egy cookie-t úgy lehet törölni, ha beállítjuk élettartama értékét 0-ra. Negatív élettartam pedig azt jelzi, hogy a cookie csak addig él, amíg az azt tároló böngészőprogram fut. Tehát egy negatív élettartamú cookie nem kerül perzisztens tárolásra a kliensoldalon. Az élettartam lekérdezése a getMaxAge, beállítása pedig a setMaxAge metódussal történik.
- Megjegyzés - a cookie-hoz fűzött megjegyzés. A cookie értékével együtt kerül tárolásra. Lekérdezése a getComment, beállítása pedig a setComment metódussal történik.
- Verzió - a cookie verzióját megadó szám. 0 értéke jelzi, hogy a cookie teljesen kompatibilis a Netscape eredeti cookie specifikációjával. 1 értéke pedig a cookie RFC 2109 kompatibilitását jelzi. Lekérdezése a getVersion, beállítása pedig a setVersion metódussal történik.
- Biztonság - biztonságos cookie-t csakis biztonságos kapcsolaton (https, ssl) keresztül lehet küldeni. Lekérdezése a getSecure, beállítása pedig a setSecure metódussal történik.

Ezen jellemzők közül egyedül a név és érték az, amit minden jelenleg elterjedt webszerver és kliensprogram támogat.

A klienstől kapott cookie-kat a HttpServletRequest getCookies metódusával lehet lekérdezni, amely azokat egy tömbben adja vissza.

Ha új cookie-t szeretnénk az aktuális kliensnél eltároltatni, akkor hozzuk létre a cookie-t a konstruktorral (ügyeljünk arra, hogy nem megengedett vagy a HTTP protokoll számára fenntartott név esetén a konstruktor kivételt vált ki), állítsuk be a cookie paramétereit, majd küldjük azt el a kliensnek a HttpServletResponse addCookie metódusával. Megjegyzendő, hogy mivel mint már említettük, a cookie-k a HTTP fejlécben kerülnek átadásra, ezért azokat elküldeni még azelőtt kell, mielőtt bármi más adatot küldtünk volna a kliens fele. Már létező cookie-t újra el kell küldeni a kliensnek, ha annak valamely jellemzőjét megváltoztattuk.

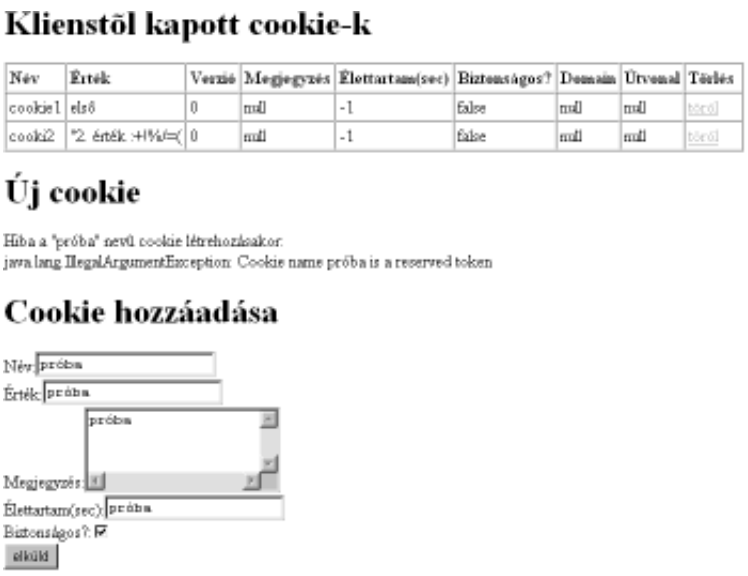
K.6.2. 2. példa: a CookieTestServlet servlet

Ezen példaservlet kilistázza a böngészőtől kapott cookie-k jellemzőit egy táblázatban, majd lehetővé teszi azok törlését egy hyperlink segítségével, illetve új cookie-t hoz létre a kliensparaméterek alapján. A létrehozandó cookie adatait egy HTML űrlapon lehet megadni, az ott megadott értékek alapján kerülnek beállításra az új cookie jellemzői.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CookieTestServlet extends HttpServlet {

    public void doGet(HttpServletRequest kérés, HttpServletResponse válasz)
           throws IOException, ServletException {

        Cookie[] cookies = kérés.getCookies();                 //kapott cookie-k
        Cookie újcookie = null;                                      //új cookie
        Exception cookiex = null;     //új cookie felvételekor fellépett kivétel
        String újnév = kérés.getParameter("cookiename");               //újnév
        String újérték = kérés.getParameter("cookievalue");           //újérték
        String újmegjegyzés = kérés.getParameter("cookiecomment");//újmegjegyzés
        int újlejár = -1;                                             //újlejár
        boolean újbiztonságos = kérés.getParameter("cookiesecure")!=null;
        if (újnév!=null && újnév.length()>0 && újérték!=null) {      //új cookie
           try {
               újlejár = Integer.parseInt(kérés.getParameter("cookiemaxage"));
           } catch (Exception e) {}
           try {                                          //új cookie felvétele
               újcookie = new Cookie(újnév, újérték);
               if (újmegjegyzés!=null) újcookie.setComment(újmegjegyzés);
               újcookie.setMaxAge(újlejár);
               újcookie.setSecure(újbiztonságos);
               válasz.addCookie(újcookie);
           } catch (Exception e) {                         //kivétel eltárolása
             cookiex = e;
           }
        } else {
          újnév = null;
        }

        String törölnév = kérés.getParameter("deletename");            //törlés
        if (törölnév!=null) for (int i=0; i<cookies.length; i++)
           if (cookies[i].getName().equals(törölnév)) {
                cookies[i].setMaxAge(0);
                válasz.addCookie(cookies[i]);
           }
```

```
válasz.setContentType("text/html");

PrintWriter out = válasz.getWriter();
out.println("<html><head><title>" + getClass().getName());
out.println("</title></head><body>");
out.println("<H1>Klienstől kapott cookie-k</H1>");

if (cookies.length > 0) {                          //táblázat felépítése
   out.println("<TABLE BORDER=\"1\" CELLPADDING=\"3\" "+
               "CELLSPACING=\"0\">");
   out.println("<TR><TD><B>Név</B><TD><B>Érték</B><TD><B>Verzió</B>"+
               "<TD><B>Megjegyzés</B><TD><B>Élettartam(sec)</B>"+
               "<TD><B>Biztonságos?</B><TD><B>Domain</B><TD>"+
               "<B>Útvonal</B><TD><B>Törlés</B>");
   for (int i = 0; i < cookies.length; i++) {       //cookie-k listázása
        Cookie cookie = cookies[i];
        out.print("<TR><TD>"+cookie.getName()+"<TD>"+cookie.getValue()+
                  "<TD>"+cookie.getVersion()+"<TD>"+cookie.getComment()+
                  "<TD>"+cookie.getMaxAge()+"<TD>"+cookie.getSecure()+
             "<TD>"+cookie.getDomain()+"<TD>"+cookie.getPath()+"<TD>");
        if (cookie.getName().equals(törölnév)) out.println("Törölve");
        else out.println("<A HREF="+válasz.encodeURL(        //törölhető
             getClass().getName()+"?deletename="+
             java.net.URLEncoder.encode(cookie.getName()))+
             ">töröl</A>");
    }
    out.println("</TABLE>");
} else out.println("Nem küldött a kliens cookie-kat.");

if (újnév!=null) {                                          //új cookie
   out.println("<H1>Új cookie</H1>");
   if (újcookie==null || cookiex!=null) out.println("Hiba a \""+újnév+
      "\" nevű cookie létrehozásakor: <BR>"+cookiex);
   else {
        out.println("<TABLE BORDER=\"1\" CELLPADDING=\"3\" "+
                "CELLSPACING=\"0\"");
      out.println("<TR><B><TD>Név<TD>Érték<TD>Verzió<TD>Megjegyzés<TD>"+
            "Élettartam(sec)<TD>Biztonságos?<TD>Domain<TD>Útvonal</B>");
     out.print("<TR><TD>"+újcookie.getName()+"<TD>"+újcookie.getValue()+
              "<TD>"+újcookie.getVersion()+"<TD>"+újcookie.getComment()+
                "<TD>"+újcookie.getMaxAge()+"<TD>"+újcookie.getSecure()+
                  "<TD>"+újcookie.getDomain()+"<TD>"+újcookie.getPath());
        out.println("</TABLE>");
   }
}

out.println("<H1>Cookie hozzáadása</H1>");           //cookie hozzáadása
out.println("<form action="+getClass().getName()+" method=POST>");
out.println("Név:<input type=text length=20 name=cookiename><br>");
out.println("Érték:<input type=text length=20 name=cookievalue><br>");
out.println("Megjegyzés:<textarea rows=3 name=cookiecomment>");
out.println("</textarea><br>");
out.print("Élettartam(sec):");
out.println("<input type=text length=20 name=cookiemaxage><br>");
out.println("Biztonságos?:<input type=checkbox name=cookiesecure>");
out.println("<br><input type=submit value=\"elküld\"></form>");
```

```
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    public void doPost(HttpServletRequest kérés, HttpServletResponse válasz)
           throws IOException, ServletException {
           doGet(kérés, válasz);
    }

}
```



K.2. ábra: A CookieTestServlet servlet képe

K.6.3. Klienskapcsolat követése a szerveren

A cookie-k használatával tehát tetszőleges szöveges információt tároltathatunk a kliensoldalon. Viszont az teljesen a klienstől függ, hogy hogyan kezeli a hozzá küldött információkat. Előfordulhat, hogy a kliens egyáltalán nem foglalkozik a szerver olyan kéréseivel, melyek cookie tárolását kezdeményezik. Ekkor tehát a cookie-kon alapuló információmentés nem működik.

Egy másik megoldás, ha nem a kliens, hanem a szerver tárolja magát az információt. Ekkor minden klienskapcsolat, amely ugyanazt a klienskapcsolat környezetet használja, ugyanazon adatokat is fogja látni. Ezen klienskapcsolat környezetet tehát a szerver tartja nyílván, majd eljuttatja annak azonosítóját a klienshez, ezután a kliens a kapott azonosítóra hivatkozva lehetővé teszi, hogy a servlet adatokat rendeljen a klienshez, melyek annak többszöri futása alatt is elérhetőek lesznek.

Egy klienskapcsolat környezetet a HttpSession osztály reprezentál, amelynek a következő főbb jellemzői vannak:

- Azonosító - a klienskapcsolat környezet szöveges azonosítója. Lekérdezni a getId metódussal lehet, beállítása pedig annak létrehozásakor automatikusan történik.
- Létrehozás időpontja - a klienskapcsolat környezet létrehozásának időpontja. Lekérdezni a getCreationTime metódussal lehet, beállítása pedig annak létrehozásakor automatikusan történik.
- Legutolsó hozzáférés időpontja - a klienskapcsolat környezet használatának legutolsó időpontja. Lekérdezni a getLastAccessedTime metódussal lehet, beállítása pedig annak használatakor automatikusan történik.
- Élettartam - a klienskapcsolat környezet érvényességi időtartama másodpercekben. Ha ezen időtartam alatt nem használja senki sem az adott környezet adatait, akkor a szerver megszünteti a környezetet. Lekérdezni a getMaxInactiveInterval, beállítani pedig a setMaxInactiveInterval metódussal lehet.

Az aktuális klienskapcsolat környezetet a HttpServletRequest getSession metódusával lehet lekérdezni. Ez a metódus, ha még nem létezik az aktuális klienskapcsolathoz környezet objektum, akkor kérés esetén automatikusan létrehoz egy új objektumot. Ha már ismert a környezet azonosító a szerveroldal számára, akkor az azt reprezentáló objektumot fogjuk visszakapni. Egy klienskapcsolat környezetről annak isNew metódusával lehet megállapítani, hogy újonnan jött-e létre. Ezen metódus egy hasznos felhasználási területe lehet annak ellenőrzése, hogy ismert-e már a kliens a servlet számára. Ha nem, akkor például átirányíthatjuk a kliensкérést egy üdvözlő, vagy éppen bejelentkezési adatokat kérő oldalhoz.

Miután megszereztük a HttpSession referenciát, objektumokat tárolhatunk benne a putValue metódussal, elérhetjük az abba már korábban berakott objektumokat a getValue metódussal, illetve törölhetjük azokat a removeValue meghívásával. A tárolandó objektumokra egy név alapján lehet hivatkozni. A kapcsolat környezetben tárolt összes objektum nevét pedig a getValueNames metódussal lehet lekérdezni.

Ha a klienskapcsolat környezetben tárolt objektum szeretne értesülni arról, hogy mikor kerül be a környezet adatai közé, illetve mikor kerül ki onnan, akkor implementálnia kell a HttpSessionBindingListener interfészt. Ekkor az adott objektum értesítést kap egy HttpSessionBindingEvent formájában, ha a putValue, vagy a removeValue metódus paramétereként tárolásra, illetve törlésre került.

Ha már nincs többé szükség egy klienskapcsolat környezetre, akkor mi magunk megszüntethetjük azt annak invalidate metódusával. A kapcsolat környezet automatikusan megszűnik, ha annak érvényességi időtartama alatt nem történik hivatkozás rá.

Mint már említettük, a klienskapcsolat környezetre annak azonosítójával hivatkozik a kliens. Ezt az azonosítót tehát a szervernek valahogy el kell juttatnia a klienshez, majd a kliensnek minden kérésével együtt el kell küldenie azt. Mivel a cookie-k pontosan erre használhatók, ezért alapértelmezés szerint a környezet azonosítója cookie-k segítségével kerül automatikusan átvitelre kliens és servlet között. Ezért a cookie-knál leírtaknak megfelelően a klienskapcsolat környezetet mindig még minden más adat küldése előtt nyissuk meg, mivel az ahhoz tartozó azonosítót reprezentáló cookie a HTTP fejlécben kerül átadásra. Abban az esetben, ha a kliensoldalon nem engedélyezett a cookie-k használata, magunknak kell gondoskodnunk a környezet azonosító átviteléről. Ezt például úgy tehetjük meg, hogy a servletünk által generált HTML oldal minden hyperlinkjéhez és űrlapjához megadjuk a környezet azonosítót, így amikor a kliens ezen hyperlinkeket vagy űrlapokat használva újra el akarja érni a servletet, akkor az azonosítót mint kliensкérés paramétert fogjuk megkapni. A környezet azonosító automatikusan belekerül minden célURL-be, ha azt a HttpServletResponse encodeURL metódusával kódoltatjuk el.

A HttpServletRequest interfész isRequestedSessionIdFromCookie, valamint isRequestedSessionIdFromURL metódusaival lehet megtudni, hogy a klienstől hogyan kapta vissza (cookie-val vagy URL paramétereként) a servlet a kért klienskapcsolat környezet azonosítóját.

K.6.4. 3. példa: a SessionTestServlet servlet

Ezen példaservlet kiírja az aktuális klienskapcsolat környezet jellemzőit, kilistázza az abban található objektumok jellemzőit egy táblázatban, majd lehetővé teszi azok törlését egy hyperlink segítségével, illetve új szövegobjektumot tárol el a kapott kliensparaméterek alapján. Az újonnan eltárolandó adatokat egy HTML űrlapon lehet megadni.

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionTestServlet extends HttpServlet {

    public void doGet(HttpServletRequest kérés, HttpServletResponse válasz)
           throws IOException, ServletException {

        HttpSession session = kérés.getSession();          //új környezet kérése
        if (kérés.getParameter("invalidate")!=null && !session.isNew()) {
           session.invalidate();
           session = kérés.getSession(true);
        }

        String újnév = kérés.getParameter("paramname");
        String újérték = kérés.getParameter("paramvalue");
                                                    //új attribútum felvétele
        if (újnév!=null && újnév.length()>0 && újérték!=null)
           session.putValue(újnév, újérték);
        else újnév = null;

        String[] adatok = session.getValueNames();

        String törölnév = kérés.getParameter("deletename");

        válasz.setContentType("text/html");

        PrintWriter out = válasz.getWriter();
        out.println("<html><head><title>" + getClass().getName());
        out.println("</title></head><body>");
        out.println("<H1>Session jellemzői</H1>");         //környezet jellemzői

        out.println("Azonosító = "+session.getId()+"<br>");
        out.println("Új? = "+session.isNew()+"<br>");
        out.println("Létrehozás időpontja = "+
                    new Date(session.getCreationTime())+"<br>");
        out.println("Utolsó hozzáférés időpontja = "+
                    new Date(session.getLastAccessedTime())+"<br>");
        out.println("Timeout(sec) = "+session.getMaxInactiveInterval()+"<br>");

        out.println("<H1>Session adatai</H1>");             //környezet tartalma
        if (adatok.length > 0) {                           //táblázatban lista
           out.println("<TABLE BORDER=\"1\" CELLPADDING=\"3\" "+
                       "CELLSPACING=\"0\">");
        out.println("<TR><TD><B>Név</B><TD><B>Érték</B><TD><B>Érték osztálya"+
                    "</B><TD><B>Törlés</B>");
           for (int i = 0; i < adatok.length; i++) {
                Object adat = session.getValue(adatok[i]);
```

```
                out.print("<TR><TD>"+adatok[i]+"<TD>"+adat+"<TD>"+
                          (adat==null ? "-" : adat.getClass().getName())+"<TD>");
                if (adatok[i].equals(törölnév)) {
                   out.println("Törölve");
                   session.removeValue(törölnév);
                } else if (adatok[i].equals(újnév)) out.println("Új");
                else out.println("<A HREF="+válasz.encodeURL(
                     getClass().getName()+"?deletename="+
                     java.net.URLEncoder.encode(adatok[i]))+
                     ">töröl</A>");
             }
             out.println("</TABLE>");
             out.println("<form action="+getClass().getName()+" method=POST>");
             out.println("<input type=hidden name=invalidate value=now><br>");
             out.println("<input type=submit value=\"Mindent töröl\"></form>");
          } else out.println("Nincsenek adatok a session-ban.");

          out.println("<H1>Paraméter hozzáadása</H1>");     //paraméter hozzáadása
          out.println("<form action="+getClass().getName()+" method=POST>");
          out.println("Név:<input type=text length=20 name=paramname><br>");
          out.println("Érték:<input type=text length=20 name=paramvalue><br>");
          out.println("<input type=submit value=\"elküld\"></form>");

          out.println("</body>");
          out.println("</html>");
          out.close();
       }

       public void doPost(HttpServletRequest kérés, HttpServletResponse válasz)
              throws IOException, ServletException {
              doGet(kérés, válasz);
       }
}
```

K.7. Servletközi kommunikáció

Azonos webszerveren futó servletek kommunikálhatnak egymással a servletfuttató környezeten keresztül, amelyet egy ServletContext objektum reprezentál. Minden servlethez tartozik egy ilyen objektum, melyet a servlet inicializációjakor kapott ServletConfig objektum getContext metódusával lehet lekérdezni. A kommunikáció egyik formája azon alapszik, hogy minden servlet névvel ellátott objektumokat tárolhat a futtató környezetben annak setAttribute metódusával, illetve ezen név alapján lekérdezheti a hozzá tartozó objektumot a getAttribute metódus segítségével. A tárolt objektumok nevének listáját a getAttributeNames metódus adja vissza, objektumot törölni pedig a removeAttribute metódussal lehet. Az így tárolt objektumok függetlenek a klienskapcsolat élettartamától, viszont csak a servletfuttató környezet futása alatt érhetőek el, nem kerülnek perzisztensen is tárolásra. Ugyanazon servletfuttató környezet objektum több servlethez is tartozhat (ennek szabályozása webszerverfüggő), ekkor ezen servletek látják egymás objektumait.

A kommunikáció másik formája, hogy adott servlet teljesen vagy csak ideiglenesen átirányíthatja a klienskérést egy másik servlethez. Ehhez azonban valahogy el kell érni a másik servletet. Erre használható a getRequestDispathcer metódus, amely a paramétereként kapott webszerveren belüli lokális erőforrás URL-hez (tehát ez nem csak servlet, hanem akár csak egy egyszerű HTML oldal is lehet) ad egy reprezentáló RequestDispathcer objektumot. Ezen objektum forward metódusát meghívva a klienskérés kiszolgálásáért a

K.3. ábra: A SessionTestServlet servlet képe

megcímzett objektum lesz a felelős. Ilyenkor a hívó servlet egyáltalán nem küldhet adatot a kliens felé, különben kivétel fog fellépni.

Ha a válasz generálása közben annak adott pontjától kezdve egy másik servlet eredményét (vagy tetszőleges URL tartalmát) szeretnénk megjeleníteni, akkor használjuk a RequestDispathcer include metódusát. Ilyenkor a külső erőforrás tartalma egyszerűen csak belekerül a mi servletünk válaszába, azaz előtte és utána is küldhetünk még adatot a kliens felé. Ha a beszúrt tartalmat egy másik servlet állítja elő, akkor ügyeljünk arra, hogy a meghívott servlet már nem módosíthatja a HTTP fejlécet (azaz nem küldhet például cookie-kat), és hogy az nehogy lezárja a kliens felé az adatátviteli csatornát.

K.7.1. 4. példa: a ContextTestServlet servlet

Ezen példaservlet először meghívja a LifeTestServlet servletet, majd az ott generált HTML oldalhoz hozzáfűzi a servlet futtatókörnyezet jellemzőinek listáját és az abban található attribútum objektumok jellemzőinek táblázatát, s lehetővé teszi azok törlését egy hyperlink segítségével, illetve új szövegobjektumot tárol el a kapott kliensparaméterek alapján. Az újonnan eltárolandó adatokat egy HTML űrlapon lehet megadni.

```
import java.io.*;
import java.util.Enumeration;

import javax.servlet.*;
import javax.servlet.http.*;

public class ContextTestServlet extends HttpServlet {
```

```
public void doGet(HttpServletRequest kérés, HttpServletResponse válasz)
       throws IOException, ServletException {

    válasz.setContentType("text/html");
    PrintWriter out = válasz.getWriter();
    out.println("<html><head><title>" + getClass().getName());
    out.println("</title></head><body><H1>"+ getClass().getName());
    out.println("</H1>LifeTestServlet meghívása...<P>");
    RequestDispatcher lifetest = null;           //LifeTestServlet meghívása
    try {
         lifetest =
      getServletContext().getRequestDispatcher("/servlet/LifeTestServlet");
    } catch (Exception e) {                                          //hiba
      lifetest = null;
      out.println(e.toString());
    }
    if (lifetest != null) try {         //LifeTestServlet kimenetének kérése
       lifetest.include(kérés, válasz);
    } catch (Exception e) {
      out.println("<BR><HR>Hiba a LifeTestServlet futása közben: "+e);
    } else out.println("<BR>A LifeTestServlet nem elérhető!");
    ServletContext context = getServletContext();          //érték felvétele
    String újnév = kérés.getParameter("paramname");
    String újérték = kérés.getParameter("paramvalue");
    if (újnév!=null && újnév.length()>0 && újérték!=null)
       context.setAttribute(újnév, újérték);
    else újnév = null;
    String törölnév = kérés.getParameter("deletename");      //érték törlése
    out.println("<H1>ServletContext jellemzői</H1>");  //környezet jellemzői
    out.println("<PRE>Fő verziószám = "+
                    getServletContext().getMajorVersion());
    out.println("Alverziószám = "+
                    getServletContext().getMinorVersion());
    out.println("Szerver információ = "+
                    getServletContext().getServerInfo());
    out.println("</PRE><H1>Attribútumok</H1>");     //attribútumok listázása
    Enumeration attribs = context.getAttributeNames();
    if (attribs.hasMoreElements()) {                    //táblázat felépítése
       out.println("<TABLE BORDER=\"1\" CELLPADDING=\"3\" "+
                   "CELLSPACING=\"0\">");
       out.println("<TR><TD><B>Név</B><TD><B>Érték</B>"+
                   "<TD><B>Érték osztálya</B><TD><B>Törlés</B>");
       for (; attribs.hasMoreElements();) {       //attribútum megjelenítése
           String attribname = (String)attribs.nextElement();
           Object attribvalue = context.getAttribute(attribname);
           out.print("<TR><TD>"+attribname+"<TD>"+attribvalue+"<TD>");
           out.print((attribvalue==null ? "-" :
                       attribvalue.getClass().getName())+"<TD>");
           if (attribname.equals(törölnév)) {
              out.println("Törölve");
              context.removeAttribute(törölnév);
           } else if (attribname.equals(újnév)) out.println("Új");
           else out.println("<A HREF="+válasz.encodeURL(     //lehet törölni
                getClass().getName()+"?deletename="+
                java.net.URLEncoder.encode(attribname))+
                ">töröl</A>");
       }
```

```
            out.println("</TABLE>");                                  //táblázat vége
        } else out.println("Nincsenek attribútumok");
        out.println("<H1>Attribútum hozzáadása</H1>");  //új attribútum megadása
        out.println("<form action="+getClass().getName()+" method=POST>");
        out.println("Név:<input type=text length=20 name=paramname><br>");
        out.println("Érték:<input type=text length=20 name=paramvalue><br>");
        out.println("<input type=submit value=\"elküld\"></form>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    public void doPost(HttpServletRequest kérés, HttpServletResponse válasz)
            throws IOException, ServletException {
            doGet(kérés, válasz);
    }
}
```

K.4. ábra: A ContextTestServlet servlet képe

Megjegyzés: Amikor adatokhoz azok nevén keresztül lehet hozzáférni (cookiekon, klienskapcsolat környezeten vagy a servletfuttató környezeten keresztül) mindig ügyeljünk a nevek egyértelműségére és vegyük figyelembe, hogy ugyanazon adatot esetleg egyszerre több servlet is láthatja. Ezért ajánlott az egyértelműség kedvéért minden névbe a servlet teljes nevét is belevenni.