Java technológiák – 4. előadás Menetkezelés. Eseményfigyelők. Szűrők.

ANTAL Margit

Sapientia - EMTE

2009

A 4. előadás célja

1. Menetkezelés
   - Munkamenetek és sütik
   - A HttpSession interfész
   - URL újraírás
2. Eseményfigyelők
3. Szűrők

HTTP és menetkövetés

- A HTTP egy állapotmentes protokoll: minden kérés-válasz kapcsolat egymástól független.
- A webkonténer feladata egy olyan mechanizmus létrehozása, amely tárolja a menetinformációkat ügyfelenként.

Meneti hatókör

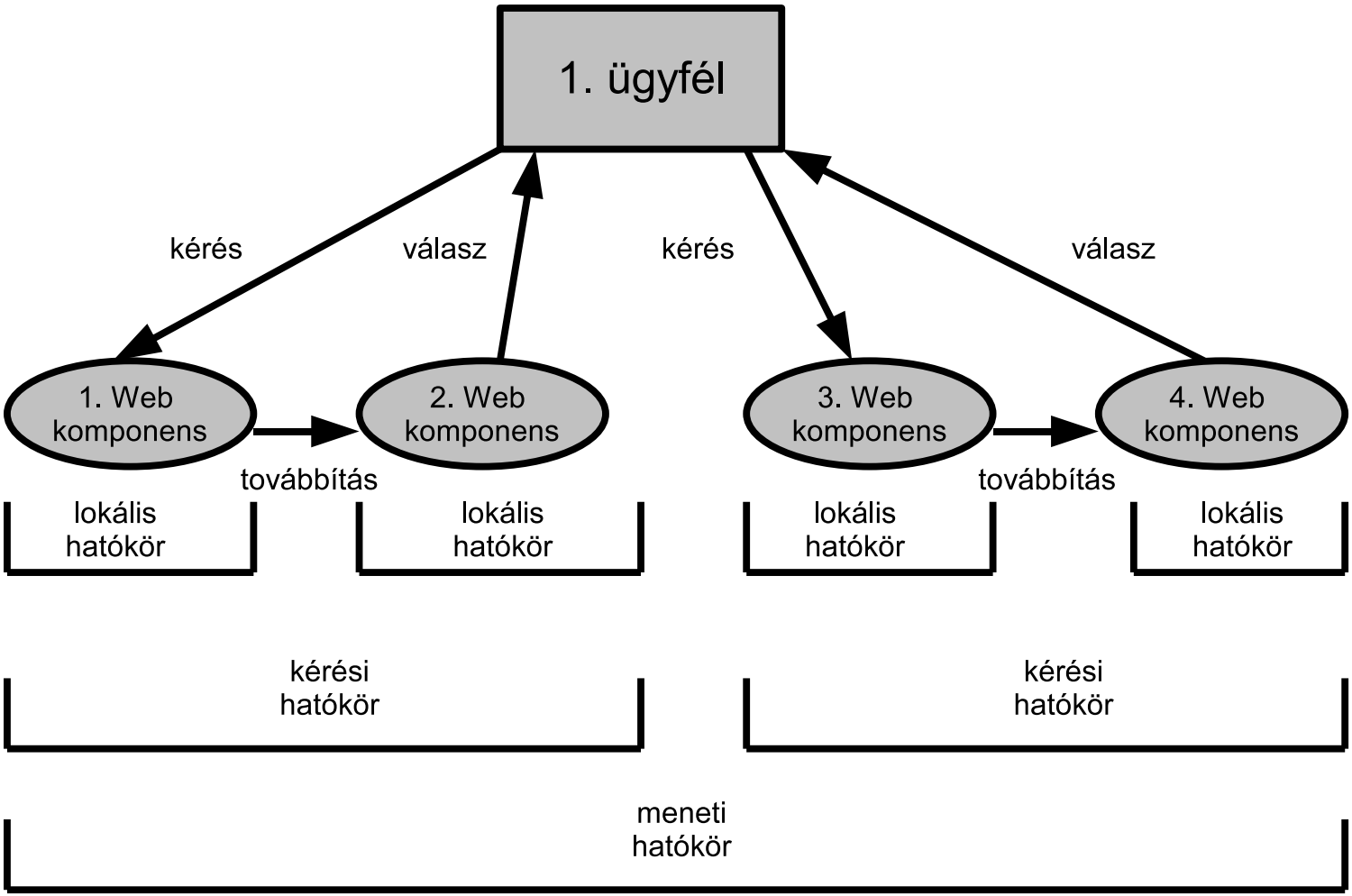


Figure: Meneti hatókör [IMRE]

Sütik

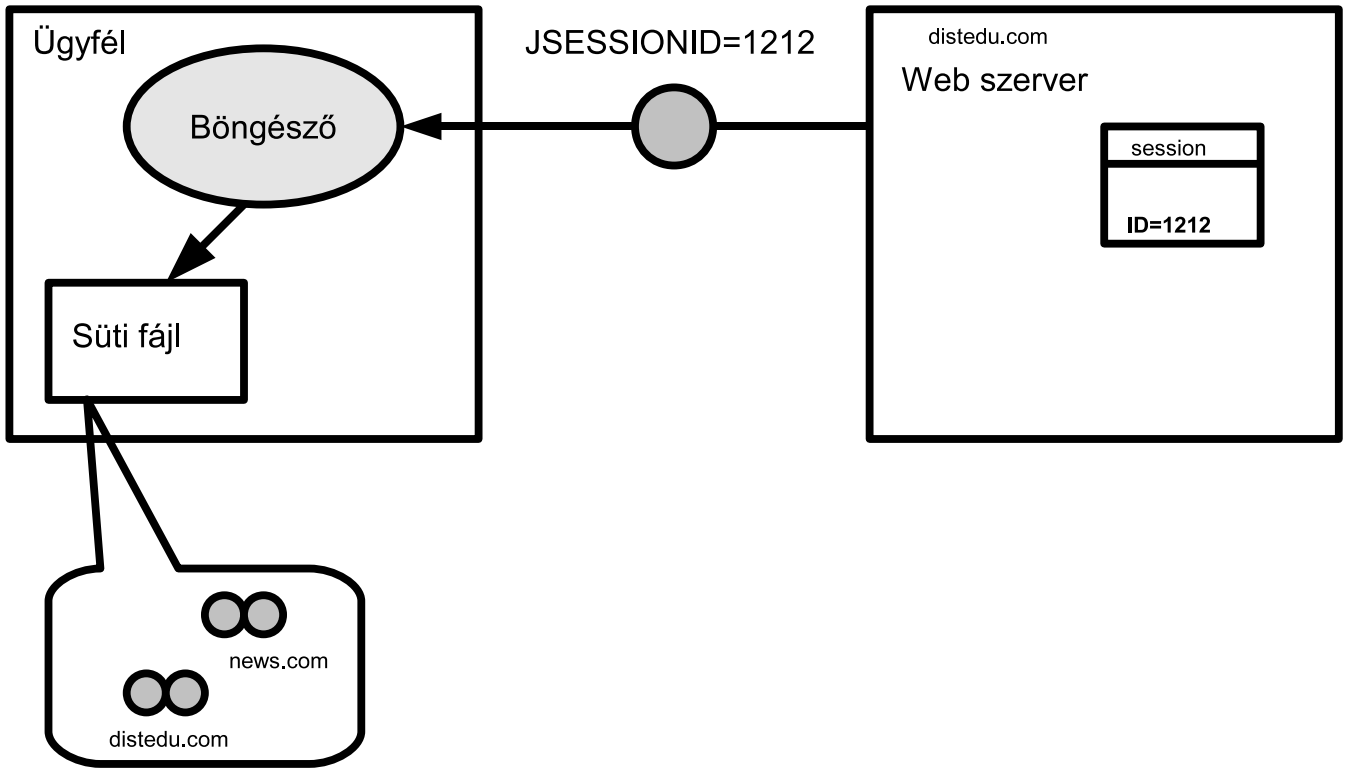
- A sütiket a HTTP protokoll specifikálja, segítségükkel kis, szöveges információkat lehet tárolni kliensoldalon.
- A sütiket a webszerver (webkonténer) küldi a kliensnek (böngésző). A sütit a Set Cookie fejlécben küldi:

  ```
  Set Cookie: NAME=VALUE; expires=DATE;
  path=PATH; domain=DOMAIN;secure
  ```

- Miután a böngésző egy adott tartományhoz eltárolt egy sütit, az ezt követő összes, az adott tartományhoz intézett kérésben ezt visszaküldi.
- A sütik tárolása kliensoldalon történik.

Süti elküldése a böngészőnek

```
Set Cookie:  NAME=VALUE; expires=DATE;
path=PATH; domain=DOMAIN;secure
```



Sütik tulajdonságai

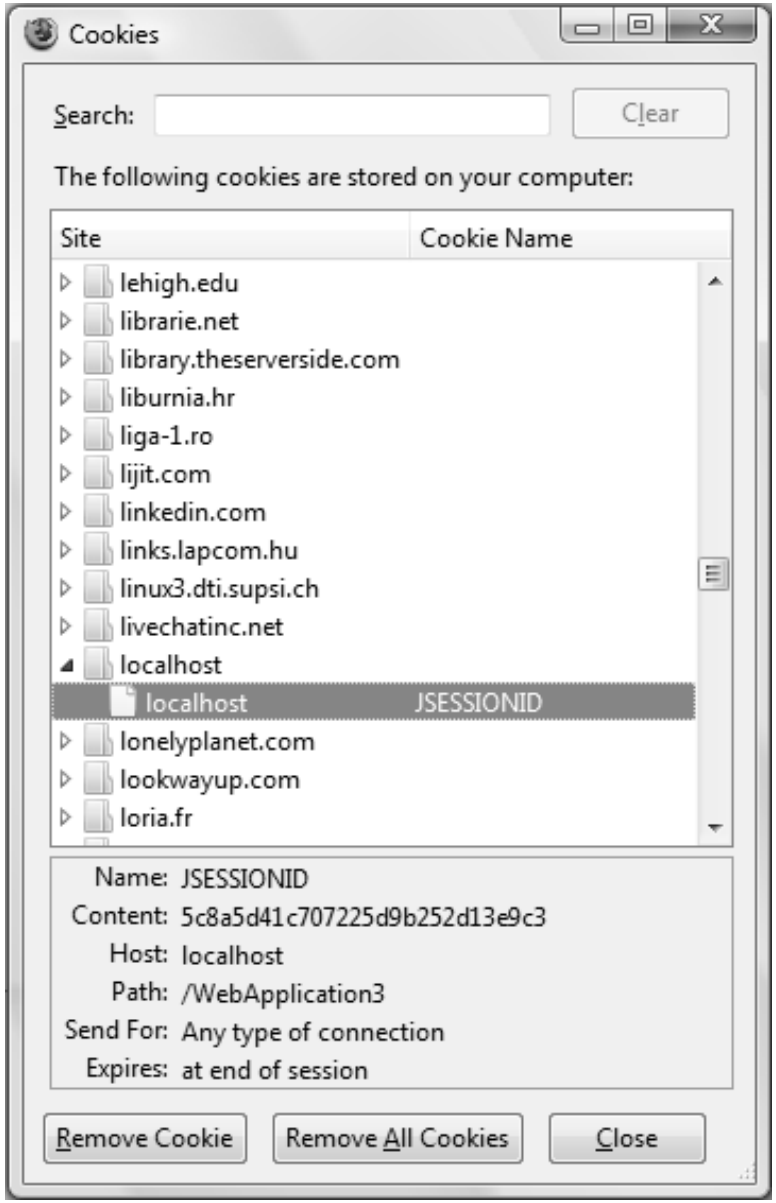
Perzisztencia

- perzisztens (maradandó): ha a süti tartalmazza az `expires` tulajdonságot
- nem perzisztens: másképpen – ebben az esetben a böngésző bezárásakor törlődik
- ha lejár a süti ideje, a böngésző letörli

Biztonság

- `secure`: csak HTTPS kapcsolaton keresztül küldhető
- másképpen: HTTP és HTTPS kapcsolaton is

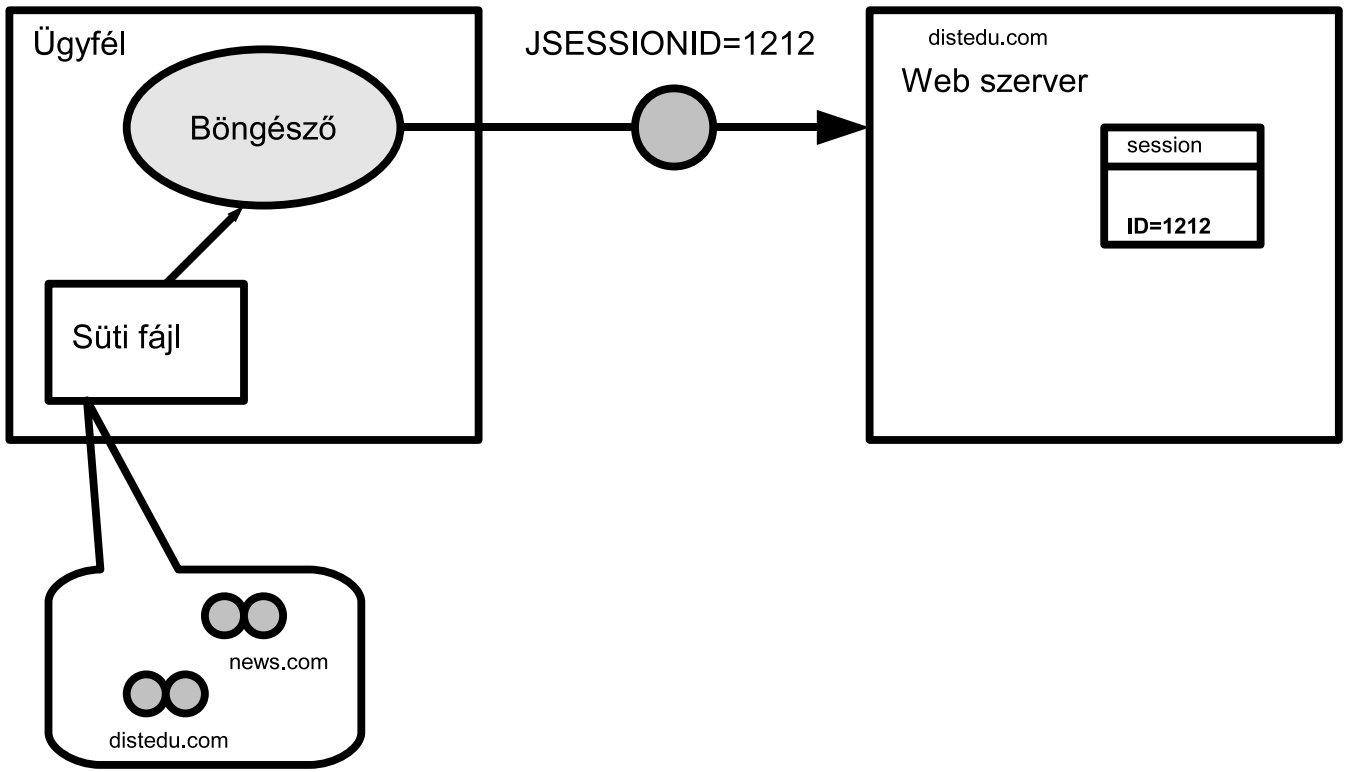
Sütik a böngészőben



Süti visszaküldése böngészőből

Cookie fejlécben

```
Cookie:  NAME1=VALUE1; NAME2=VALUE2;
NAME3=VALUE3;
```



HttpSession API

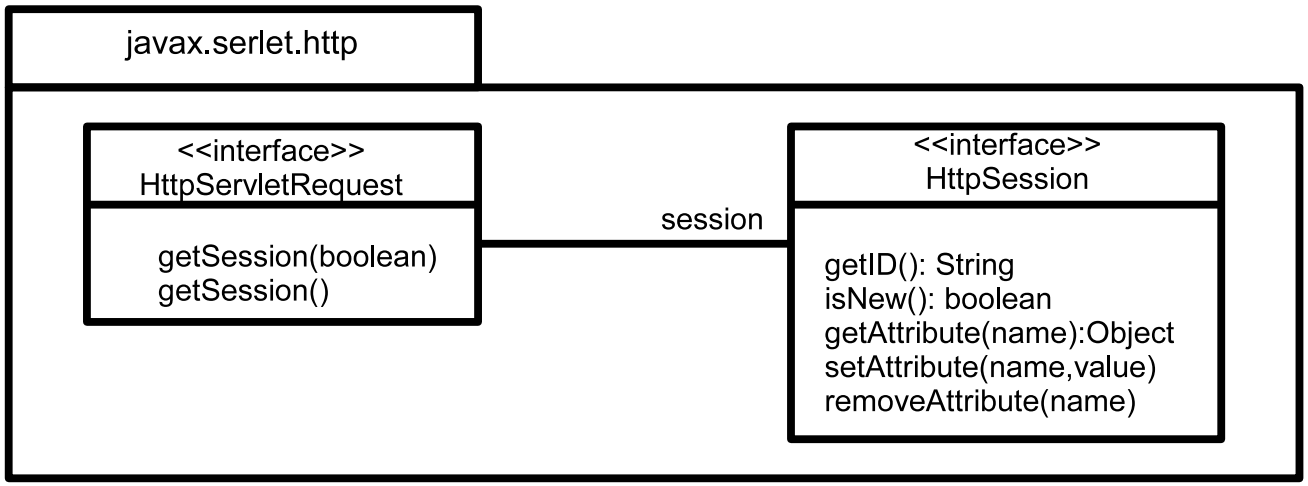


Figure: A HttpSession API [FJ310]

Servlet API - menetkövetés

- A webkonténer minden egyes ügyfeléhez hozzá van rendelve egy `HttpSession` objektum.
- Ez az objektum információkat tárol az ügyfélről, illetve lehetővé teszi az illető ügyfélhez tartozó menetinformációk mentését.
- A menet élettartama lejárhat:
  - automatikusan, egy előre beállított tétlenségi idő elteltével,
  - kézi vezérléssel.
- Ha egy menet érvénytelenné válik, akkor a menetobjektumból az összes információ elvesztődik
- Ha az információkat nem akarjuk elveszíteni $\Rightarrow$ ügyfél bejelentkezése+mentés adatbazisba.

Menet lejárati ideje – deklaratív módszer

web.xml telepítésleíró – az idő percben kifejezve

```
<web-app>
  <session-config>
    <session-timeout>
      60
    </session-timeout>
  </session-config>
</web-app>
```

Menet lejárati ideje – programozott módszer

az idő másodpercben van kifejezve

```
public interface HttpSession{
  void invalidate();
  long getCreationTime();
  long getLastAccessedTime();
  int getMaxInactiveInterval();
  void setMaxInactiveInterval(int);
}
```

Menet lejárati idejének helyes megválasztása

- Banki alkalmazásoknál a lejárati idő nagyon rövid – biztonsági okok.
- Adatbázis kapcsolatokat tároló menetek is minimálisra állítják a lejárati időt.
- Ha a menet nem tárol költséges objektumkat, a menet lejárati idő lehet a szokásosnál hosszabb.
- Bevásárló kocsi tartalmát tároló menetek hosszú lejáratúak.

Szervlet – doGet metódus – Mit csinál?

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
HttpSession session=request.getSession();
Integer counter =
  (Integer)session.getAttribute("counter");
if( counter == null)
  counter=new Integer(1);
else
  counter=new Integer(counter.intValue()+1);
session.setAttribute("counter",counter);
out.println("<HTML><HEAD><TITLE>
          Session Counter</TITLE></HEAD>");
out.println("<BODY>");
out.println("</H4>You have visited this page "+
  counter+" times</H4>");
out.println("</BODY>");
out.println("</HTML>");
```

Sütik – összefoglalás

- A sütiket a webszerverek küldik.
- A sütiket az ügyfél gépe tárolja.
- A sütiket az ügyfél böngészője a webszerver tartománynevéhez asszociálva tárolja.
- Minden egyes kérésben, amely egy bejegyzett webszerver irányába történik, a hozzá tartozó összes süti is elküldésre kerül.
- A sütik élettartamát a szerveroldal állítja be a süti küldésekor, és az ügyfél böngészője kezeli.

A menetobjektum használata – szerveroldal

- tárolja a menetazonosítót – JSESSIONID,
- más objektumokat is tárolhat.

Objektum regisztrálása meneti hatókörbe

```
HttpSession session = request.getSession();
Course course = new Course("Java SE",
        "Java Standard Edition", 1000);
session.setAttribute("course", course);
```

Objektum előkeresése meneti hatókörből

```
HttpSession session = request.getSession();
Course c =
    (Course)session.getAttribute("course");
```

Adatok küldése sütikben

Ha a menetazonosítón (JSESSIONID) kívül más adatot is szeretnénk küldeni ...

Küldés

```
String name = request.getParameter("name");
Cookie c = new Cookie("yourname",name);
response.addCookie(c);
```

Fogadás

```
String name=null;
Cookie[] cookies = request.getCookies();
for( int i=0; i<cookies.length; ++i ){
  if(cookies[i].getName().equals("yourname"))
    name = cookies[i].getValue();
}
```

Megkötések a sütikkel kapcsolatosan

- A sütik alkotják az alapértelmezett menetkezelési mechanizmust.
- Bizonyos böngészők nem engedélyezik a sütiket.
- Sütik engedélyezettségének lekérdezése:
  `HttpServletRequest -->`
  `isRequestedSessionIdFromCookie`
- A menetazonosítót tartalmazó süti neve:
  `JSESSIONID`

URL újraírás

- A szervletek specifikációja előírja, hogy azoknál a böngészőknél is kell támogatni a menetkövetést, amelyek nem fogadnak sütiket.
- Egyik sütihelyettesítő eljárás az URL újraírás.
- Az URL újraírás kevésbé elegáns és átlátható, mint a sütik.
- Még a statikus HTML lapokat is dinamikusan kell előállítani, mert URL módosítás lesz minden egyes `A-HREF` és `FORM` tagban

URL újraírása: encodeURL

```
out.println("<form action='" +
    response.encodeURL("add_course.do") +
    "' method='POST'>");
```

Mihez lehet eseményfigyelőt rendelni?

- kérés,
- kérés szintű attribútum,
- munkamenet,
- munkamenet szintű attribútum,
- munkamenet aktiválása,
- webalkalmazás,
- webalkalmazás szintű attribútum.

Interfészek

- javax.servlet.ServletRequestListener
- javax.servlet.ServletRequestAttributeListener
- javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSessionAttributeListener
- javax.servlet.http.HttpSessionActivationListener
- javax.servlet.ServletContextListener
- javax.servlet.ServletContextAttributeListener

Weblkalmazás-figyelő készítése

ServletContextListener

Webalkalmazás életciklusát a webkonténer szabályozza:

- Webkonténer indítása $\Rightarrow$ minden telepített webalkalmazás inicializálódik.
- Webkonténer leállítása $\Rightarrow$ Minden webalkalmazás megsemmisül.
- Készíthetünk olyan figyelőt, amely lekezeli a fenti eseményeket.

ServletContextListener

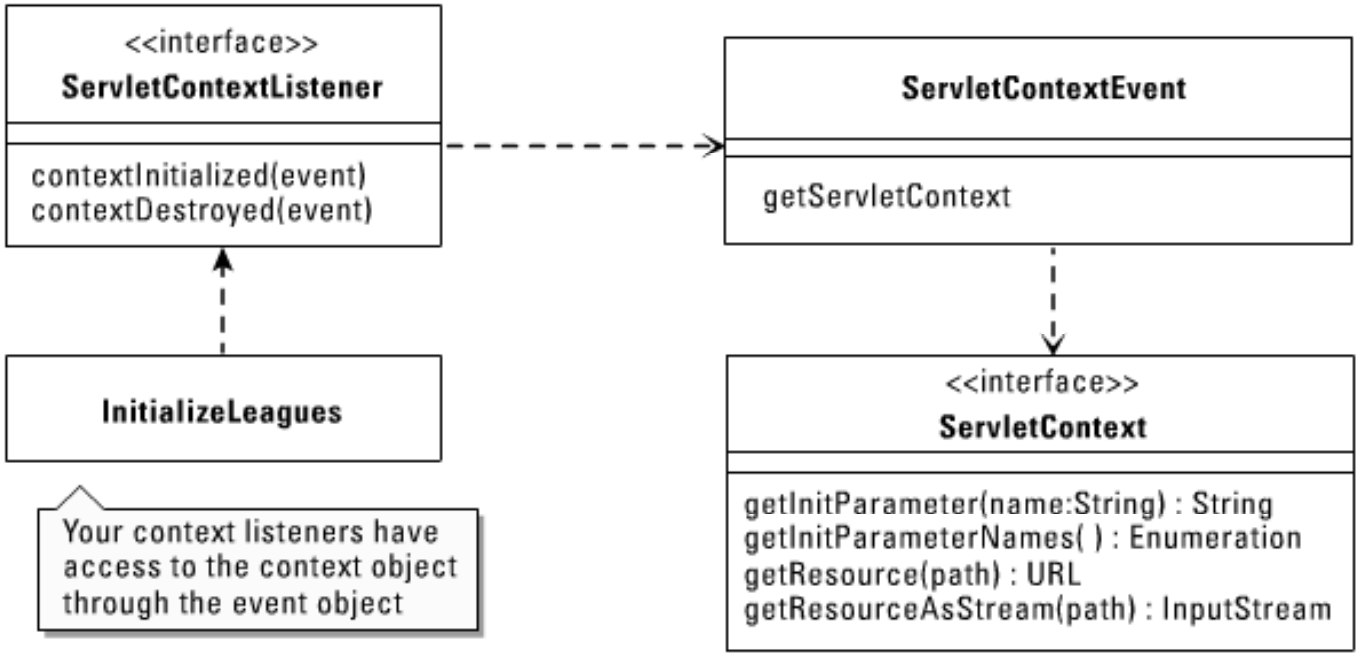


Figure: ServletContextListener [SL314]

ServletContextListener interfész

Műveletek

- `contextInitialized(ServletContextEvent)`
- `contextDestroyed(ServletContextEvent)`

Webalkalmazás paraméterei

web.xml

```
<context-param>
    <param-name>course-file</param-name>
    <param-value>
        /WEB-INF/courses.txt
    </param-value>
</context-param>
    ...
<listener>
    <listener-class>
        InitializeCourses
    </listener-class>
</listener>
```

ServletContextListener I

```
public class ApplicationContextListener
  implements ServletContextListener {
  public void contextInitialized(
      ServletContextEvent arg0) {

   ServletContext ctx=
      arg0.getServletContext();
   List courseList= new LinkedList();
   String courseFile=ctx.getInitParameter(
      "course-file");
   //Fajl beolvasasa
   InputStream is = null;
   BufferedReader reader = null;
   try{
      is = ctx.
         getResourceAsStream(courseFile);
```

ServletContextListener II

```
  reader = new BufferedReader(
           new InputStreamReader( is ));
  String record;
  while((record = reader.readLine())!=null ){
     //A sor feldolgozasa: name, price
     Course course = new Course(name, price);
     courseList.add(course);
  }
  ctx.setAttribute("courseList",
                              courseList);
  }
  catch( Exception e ){
    context.log("Exception
      while processing the courses file.",e);
  }
  finally{//Lezarasok}
}
```

Menetszámláló – HttpSessionListener

```
package listeners;
import javax.servlet.http.*;

public class SessionCounterListener
        implements HttpSessionListener{
 private static int sessionCounter = 0;
 public void sessionCreated(
                    HttpSessionEvent e){
    sessionCounter++;
 }
 public void sessionDestroyed(
                    HttpSessionEvent e){
    sessionCounter--;
 }
 public static int getSessionNumber(){
    return sessionCounter;
 }
}
```

Menetszámláló – HttpSessionListener

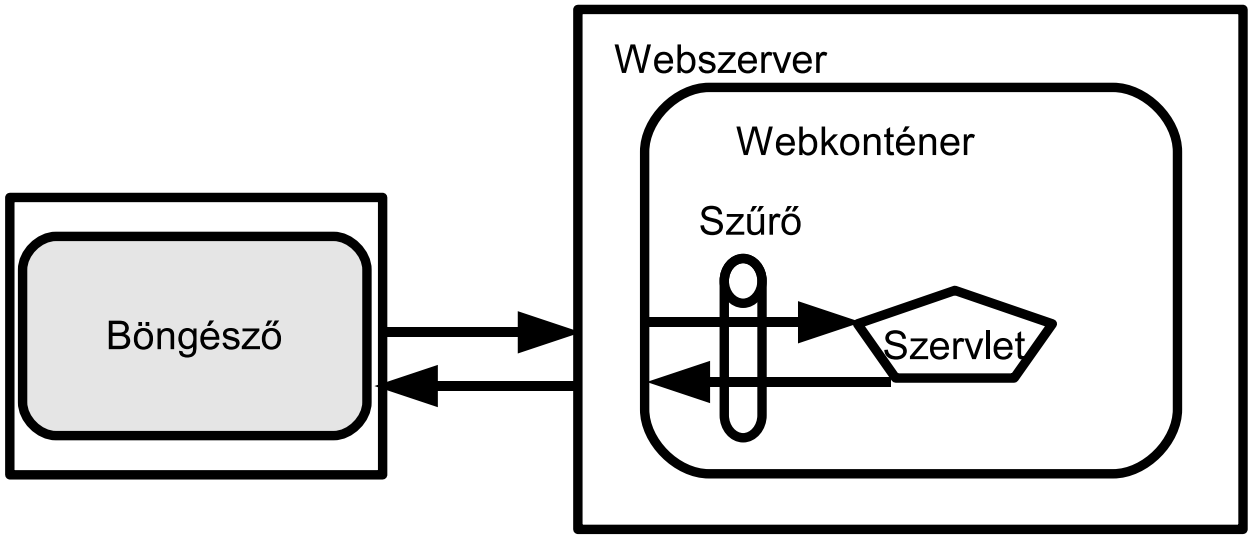
Kérdés

Helyesen viselkedik-e az előző figyelő egyidejű kérések esetén?

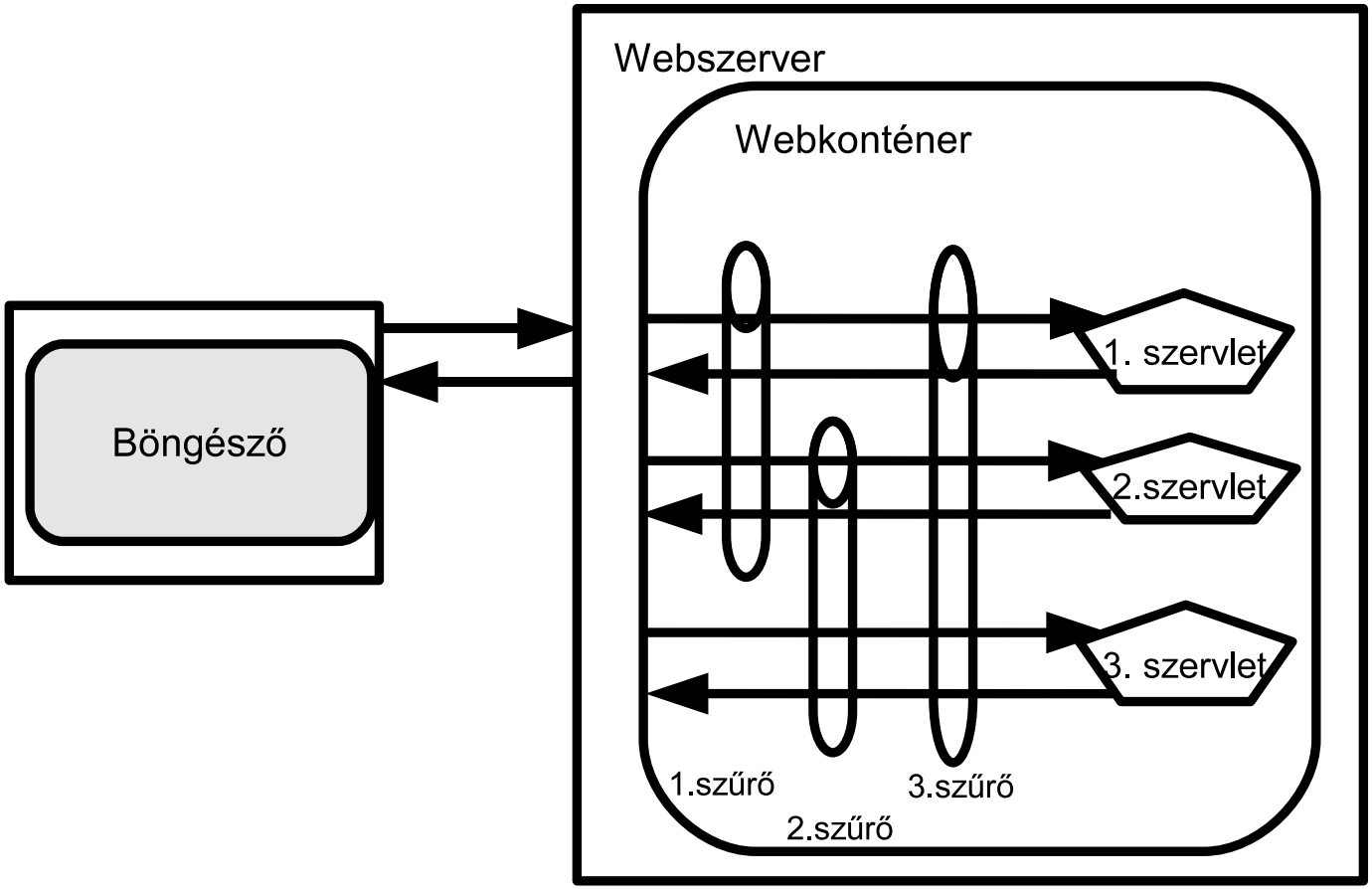
Szűrővel végezhető tevékenységek

- jogosultságellenőrzés
- szervletek hatékonyságának mérése és naplózása
- a válasz tömörítése
- lokalizáció

Kérésfeldolgozás szűrővel



Kérésfeldolgozás szűrővel



A Filter API

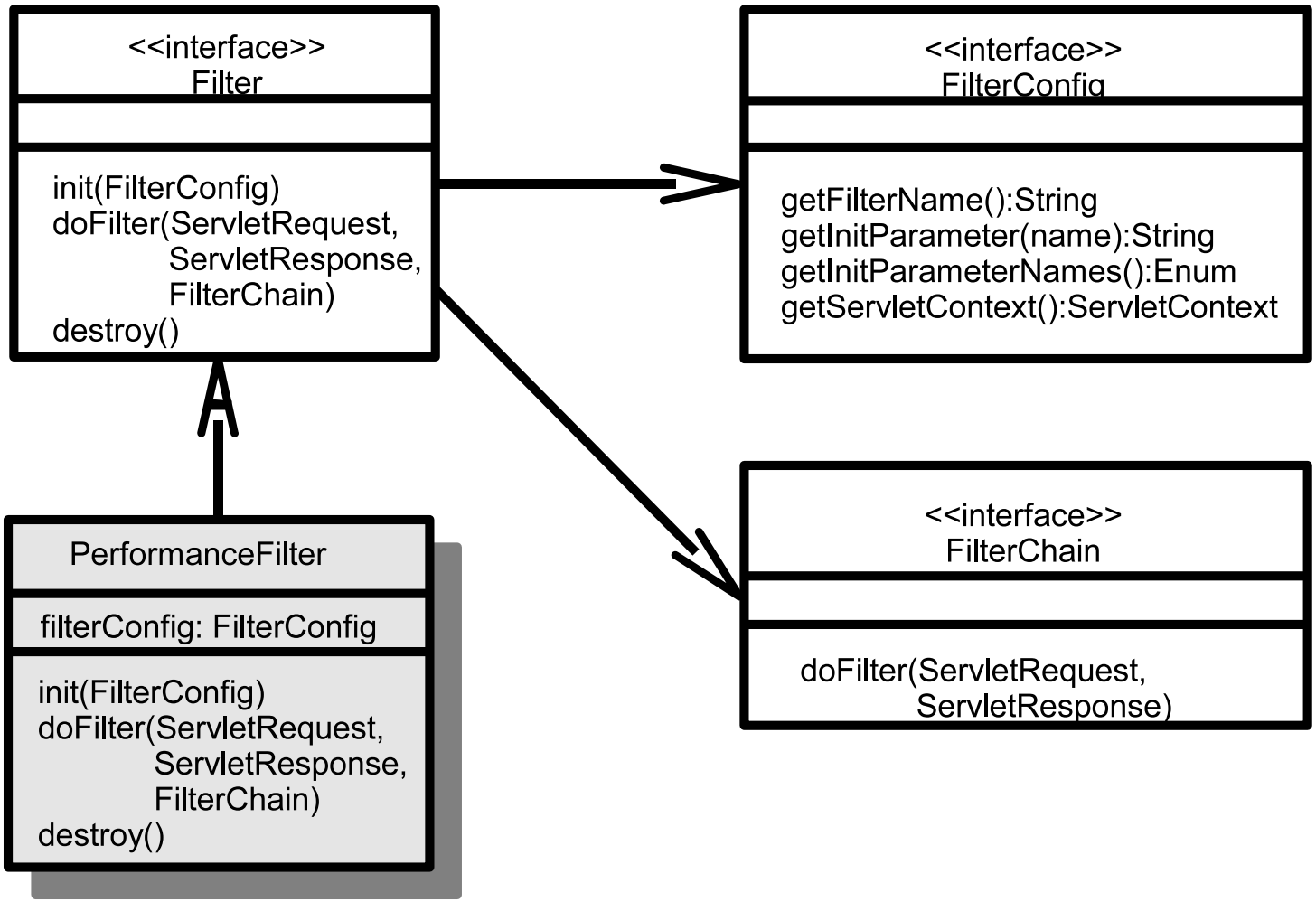


Figure: A Filter API [SL314]

Kérés paramétereinek naplózása

```
public void doFilter(
 ServletRequest request,
 ServletResponse response,
 FilterChain chain)
 throws IOException, ServletException {
 Logger log=Logger.getLogger("Req. par.:");
 Enumeration parameters=
            request.getParameterNames();
 while(parameters.hasMoreElements()){
   String p=(String)parameters.nextElement();
   log.info(p+": ");
   String[] ps=request.getParameterValues(p);
   for( int i=0; i<ps.length; ++i )
     log.info("\t"+ps[i]);
 }
 chain.doFilter( request,response);
}
```

Szűrő konfigurálása

```
<filter>
    <filter-name>LogFilter</filter-name>
    <filter-class>
        LogRequestParametersFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Szűrők leképzése

- Szervlet-leképzés

```
<filter-mapping>
    <filter-name>PerfFilter</filter-name>
    <servlet-name>
        FrontController
    </servlet-name>
</filter-mapping>
```

- URL-leképzés

```
<filter-mapping>
    <filter-name>PerfFilter</filter-name>
    <url-pattern>/*.do</url-pattern>
</filter-mapping>
```

PerformanceFilter szűrő

```
public void doFilter(ServletRequest request,
ServletResponse response,FilterChain chain)
throws IOException, ServletException {

long begin = System.currentTimeMillis();
chain.doFilter(request, response);
long end = System.currentTimeMillis();

filterConfig.getServletContext().log(
"Ellapsed time: "+(end-begin)+" ms");
}
```

A PerformanceFilter szűrő konfigurálása

```
<filter>
    <filter-name>
        PerformanceFilter
    </filter-name>
    <filter-class>
        filters.PerformanceFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>
        PerformanceFilter
    </filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Feladat

```
<servlet-mapping>
<servlet-name>FrontController</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
<filter-mapping>
<filter-name>perfFilter</filter-name>
<servlet-name>FrontController</servlet-name>
</filter-mapping>
<filter-mapping>
<filter-name>auditFilter</filter-name>
<url-pattern>*.do</url-patern>
</filter-mapping>
<filter-mapping>
<filter-name>transFilter</filter-name>
<url-pattern>*.do</url-patern>
</filter-mapping>
```

Kérdés

A fenti szűrőkonfiguráció egy app nevű webalkalmazáshoz tartozik, amely az ms.sapientia.ro szerver 8080-as portján fogadja a kéréseket. Legyen a kérés URL a következő:

Kérés

```
http://server:8080/app/admin/add_league.do
```

Mely szűrők és milyen sorrendben hívódnak meg?