Webfejlesztés alapjai

Óbudai Egyetem, Java Programozás
Mérnök-informatikai kar
Labor 7

Bedők Dávid
2017.10.30.
v1.0

Webfejlesztés

A mai világban szinte minden "programozás iránt érdeklődő" 14 éves "webprogramozó".

Általában ez PHP nyelven írt "weboldalakat" eredményez.

Ez egy látványos és közösségi eszköz, melyben gyorsan el lehet érni sikereket, azonban egy idő után nagyon hiányozni fog a "programozói alapozás" azoknak, azoknak, akik "csak" weben tudnak fejleszteni.

Alapfogalmak

Hypertext Transfer Protocol (HTTP)

Kliens és server közötti kérés-válaszok (HTTP request - HTTP response) kiszolgálására használható. A kliens (*user-agent*) nem feltétlenül böngésző, de legtöbbször.

TCP/IP réteg felett helyezkedik el, kizárólag TCP protokollt használ (adatvesztés nem megengedett).

Állapot néküli protokoll! Plain text (ASCII) kommunikáció.

1991: 0.9 (csak GET metódus)

1996: 1.0 (kapcsolat kérés után záródik)

1999: 1.1 (kapcsolat tartás megvalósítható, stream)

2005: 2.0

Tim Berners-Lee

HTTP Request line (GET [url] HTTP/1.1)
Request Header (key-value pairs)
Message body/payload (optional)

HTTP Response status line with status code (HTTP/1.1 200 OK)
Response Header (key-value pairs)
Message body/Payload (optional)

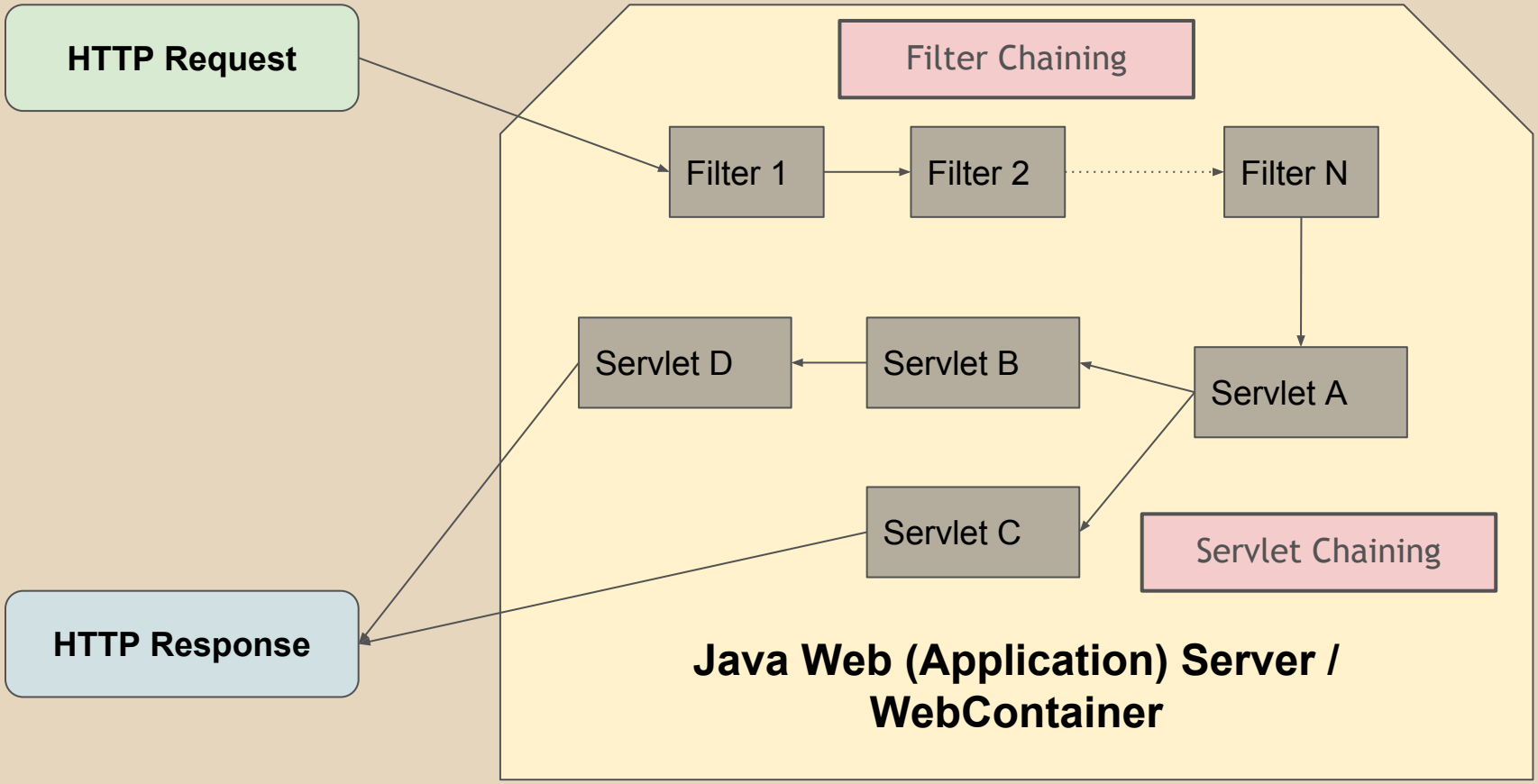
Webfejlesztés Java-ban

A Java egyik erőssége a gazdag webes framework-ök megléte. Ezek szinte mindegyike **Servlet**-eket generál a háttérben.

A Servlet egy egyszerű Java osztály, mely a `javax.servlet.GenericServlet` osztályból származik (mi ennek egy leszármazottját fogjuk használni ősként: `javax.servlet.http.HttpServlet`).

A Servlet osztályok képesek arra, hogy HTTP kérésekre HTTP választ adjanak. Ez általában (X)HTML tartalom.

HTTP kérések feldolgozása



GenericServlet

GenericServlet.java

```
package javax.servlet;

public abstract class GenericServlet implements Servlet,
ServletConfig, Serializable {
    [..]

    public void init() throws ServletException {}

    public void init(ServletConfig config) throws
ServletException {}

    public abstract void service(ServletRequest request,
ServletResponse response) throws ServletException, IOException;

    public void destroy() {}

}
```

A container miután példányosítja a Servlet-et, meghívja pontosan egyszer az `init()` metódusát.

A `service()` metódus **minden HTTP request esetén külön szálban fut le** (de ugyanazon a példányon).

A Servlet “halála” előtt pontosan egyszer lefut a `destroy()` metódus is (container hívja).

HttpServlet

GenericServlet.java

```
package javax.servlet.http;

public abstract class HttpServlet extends GenericServlet implements Serializable
{
    [..]

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
    protected void doHead(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
    protected void doPut(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
    protected void doDelete(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
    protected void doOptions(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
    protected void doTrace(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {}
}
```

Minden HTTP method-hoz megtalálható a megfelelő do..() metódus.

Filter

Filter.java

```
package javax.servlet;

public interface Filter {

    void destroy();

    void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain);

    void init(FilterConfig filterConfig);

}
```

```
@WebFilter("/LoremIpsum")
public class LoremFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)  throws IOException, ServletException {
        [..]
        chain.doFilter(request, response);
    }
}
```

Webfejlesztés menete Java-ban

- HTTP Request beérkezik a Container-be. Az URL alapján általunk vagy a használt keretrendszer által írt Filter-ek és Servlet-ek “aktiválódhatnak” (kijelölődik a feldolgozás workflow-ja (bár természesen ebben lehetnek elágazások, de akkor is egy részfa “kijelölődik).
- Bejövő HTTP Request opcionális feldologzása/előfeldolgozása/módosítása saját Filter implementáció (k)-on keresztül. A HTTP Response előkészítése is megtörténhet.
- Bejövő HTTP Request feldolgozása saját Servlet-en keresztül. A Servlet tipikusan elkészíti/módosítja a HTTP Response-t. Átirányítja a feldolgozás egy másik Servlet-nek avagy ha nincs ilyen elküldésre kerül a kliensnek az elkészült plain text HTTP Response.

Vagyis a Servlet Java nyelven egy plan/text-et állít elő, ami ha egy (X)HTML dokumentum, akkor a böngésző ezt weblapként értelmezi (de lehet akármi más is, pl. PDF, stb.).

Vagyis `StringBuilder append()`?

A különféle framework-ök azért léteznek, mert Servleteket készíteni elég körülményes, nincs alap támogatás ahhoz, hogy ne keveredjen a tartalmi megjelenítés (XHTML) és a dinamikus elemek.

A fejlődés egy fontos támpontja lesz majd a JSP alapjainak megismerése, mely bár ugyanúgy keveri általában az XHTML és Java kódot (mint egy "rosszul" megírt PHP script, DE ne állítsunk a kettő között párhuzamot!), azonban már gyorsabb munkát eredményez.

Web Frameworks

A különféle webes frameworkök általában saját szabályok szerinti API-val dolgoznak, és legtöbbjük XHTML állományokban tárolja a tartalmi részeket, mely így jobban leválasztható az üzleti logikától. A háttérben ezen rendszerek Servlet-eket készítenek el, de szerencsére ezeket már ember "jó esetben" nem elemezgeti.

Ilyen framework-ök megismerése nem scope.

Hello World servlet

File | New | Dynamic Web Project

Name: **HelloServlet**

helyi menü (project) - new Servlet:

- Java package: `hu.qwaevisz.hello.servlet`
- Class name: `GreatingServlet`

helyi menü (project) - Properties | Java build path | Libraries | Add Variable..| Add external JARs

- `servlet-api.jar` (a fordításhoz kell csak, runtime a container fogja biztosítani)

Greating servlet készítése

GreatingServlet.java

```
package hu.qwaevisz.hello.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/Greating")
public class GreatingServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
        out.close();
    }
}
```

`servlet-api.jar` tartalmazza ezeket az osztályokat

Bár ez nem valid XHTML tartalom, a böngészőben meg tud jelenni.

Első servlet

`@WebServlet("/Greating")` annotáció

Megadja hogy milyen URL-en fogjuk tudni megszólítani a Servlet-et.

http://localhost:8080/hello/Greating

Ha a webalkalmazás context root-ja `hello` (Tomcat esetén a WAR file neve) és a webserver a `localhost:8080`-on elérhető.

`doGet()` és `doPost()` metódusok

Itt lehet feldolgozni a kérést (ami kapcsán ide jutottunk), illetve legyártani a választ. A metódusok paraméterei segítenek nekünk ebben. A `doGet()` a GET-es HTTP kéréseket szolgálja ki (ez fut le akkor is, ha csak beírjuk a böngészőbe a Servlet pontos címét), míg a `doPost()` a POST-osokat.

Apache Tomcat

http://tomcat.apache.org/

The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies.

Jelenleg 9.x-es verziónál tart, a Labor során 8.0.x-es verziókat használunk. Javasolt 8.5.x használata (ebben a **Servlet API 3.1** és a **JSP API 2.3** van benne).

Install: unzip (pl.: `apache-tomcat-8.5.8-windows-x64.zip`)

Test: http://localhost:8080/

```
>cd /apache-tomcat-8.5.8/bin
>./startup.sh
```

```
>cd c:\apache-tomcat-8.5.8\bin
>startup.bat
```

Konfiguráció és információ

Avalon project name → Avalon (city), Catalina Island (California, USA) → Catalina → Apache Tomcat HOME dir name: CATALINA_HOME, Default Service name: Catalina

`\conf` → Konfigurációs állományok

`\lib` → 3rd party library-k (`servlet-api.jar` és a `jsp-api.jar` innen használható)

`\webapps` → Webalkalmazások default “virtual directory”-ja (WAR állományokat ide kell másolni/deployolni).

`\logs` → napló állományok (pl. `localhost.YYYY-MM-DD.log`).

\conf\server.xml

```
[..]
<Service name="Catalina">
    [..]
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    [..]
</Service>
[..]
```

Catalina service HTTP/1.1 connector’s port: 8080

Hello World servlet

helyi menü (project) - Export.. | WAR file

- Start Apache Tomcat 8.0.x

  `.\apache-tomcat-8.0.28\bin\startup.bat`

  ha hiányolja a JAVA-t, írjuk be a `startup.bat` első sorába (értelemszerű elérési úttal):

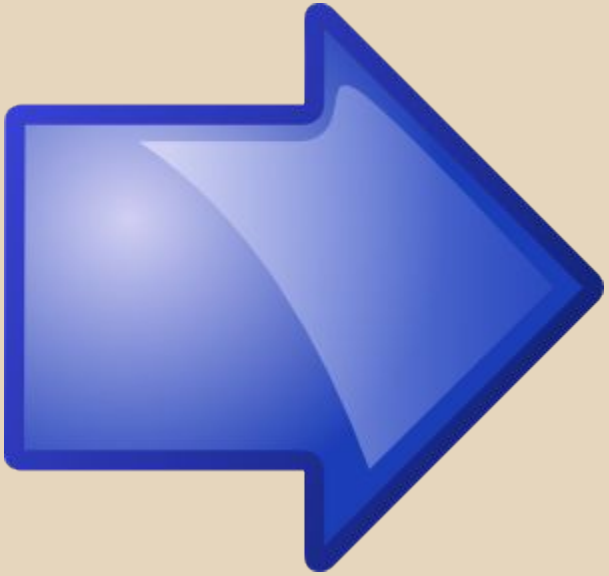
```
SET JAVA_HOME=c:\apps\Java\jdk1.7.0_06\
```

- WAR állomány másolása (**hello.war**):

  `.\apache-tomcat-8.0.28\webapps\`

- http://localhost:8080/**hello**/Greating

Üdvözlő Servlet (proj: HelloServlet)



```
GreatingServlet.java
 • doGet()
 • responseContent()
```

http://localhost:8080/hello/Greating

Servlet életciklusa

Fontos! A Servlet-ek példányai nem jönnek létre minden kérés esetén! Egyszer létrehozza őket a webcontainer (pl. Apache Tomcat), majd ezt követően ha valaki megszólítja a Servlet-et, akkor a már létrehozott példányt kapja meg (azonban minden kérés külön szálon fog futni)!

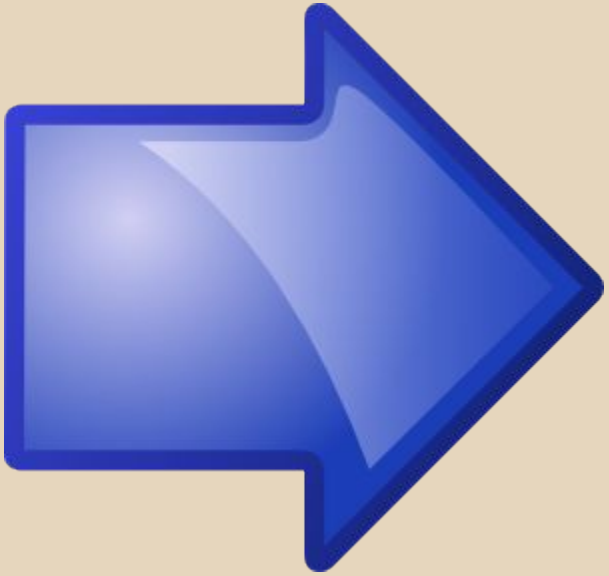
Ennek egyenes következménye, hogy “klasszikus” példány változókat gyakorlatilag TILOS a Servlet-ben használni. Példányok majd pl. egy Spring/JEE környezetben kizárólag inject-ált field-ek lesznek, ahol a container biztosítja ezek szálbiztosságát (proxy-k, stb.). *Nálunk ez a téma nem scope*.

Test servlet készítése

TestServlet.java

```
package hu.qwaevisz.hello.servlet;
[..]
@WebServlet("/Test")
public class TestServlet extends HttpServlet {

    private int counter;

    public TestServlet() {
        this.counter = 0;
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.counter++;

        PrintWriter out = response.getWriter();
        out.println(Integer.valueOf( this.counter).toString());
        out.close();
    }

}
```

Teszt Servlet (proj: HelloServlet)



```
TestServlet.java
 • counter
 • doGet()
```

http://localhost:8080/hello/Test

HttpServletRequest

```
void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    [..]
    Enumeration<String> names = request.getParameterNames();
    String userName = request.getParameter("name");

    User user = service.getUser(userName);

    request.setAttribute("user", User);
    [..]
    Enumeration<String> names = request.getAttributeNames();
    User user = (User) request.getAttribute("user");
}
```

Minta kódok..

Metódusok

```
String getParameter( String parameterName ) {...}
```

- A HTTP request-ben küldött paraméterek értékei (pl. HTML FORM mezői és azoknak értékei).
- `String` kulcs, `String` érték.

```
void setAttribute( String attributeName, Object  ) {...}
Object getAttribute( String attributeName ) {...}
```

- Bármely Servlet írhat a request-be attribute-okat, melyeknek String kulcsuk, és tetszőleges Object értékük lehet.
- A Servlet chain az elsődleges felhasználási terület: egyik Servlet eltárol a request során valamit, amit ugyanezen feldolgozás egy későbbi Servlet-e elvár, kiolvas és feldolgoz.
- `String` kulcs, `Object` érték

Új feladat - Személy regisztráló

Hozzunk létre egy weboldalt, melyen keresztül személyeket lehet regisztrálni. A következő adatokat kérjük be egy regisztrációs űrlapon:

- név
- jelszó
- jelszó ismét
- e-mail cím

A regisztráció akkor legyen sikeres, ha:

- a név nem túl rövid (min. 5 karakter legyen)
- a megadott két jelszó megegyezik
- az e-mail mező tartalmaz egy @ jelet
- egyik mező sem üres

Eredmény oldal

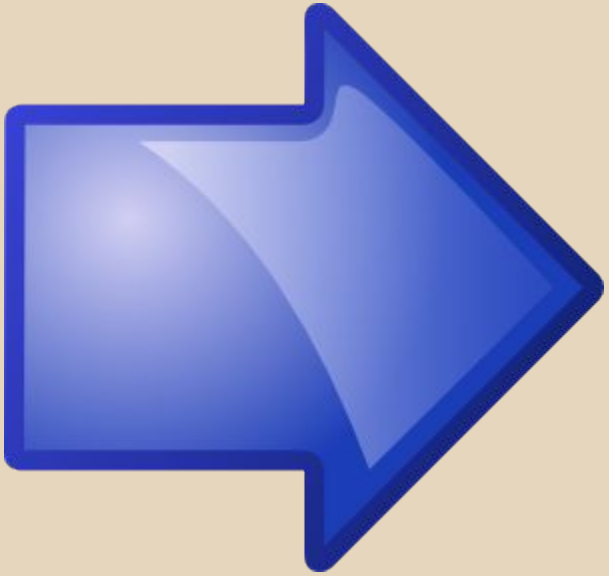
Sikeres regisztráció után egy eredmény oldalt készítsünk el, melyben a már regisztrált személyek láthatóak táblázat formájában!

Sikertelen esetben jelenítsük meg a talált hibákat a regisztrációs oldalon.

Új ismeretek

- A regisztrációs **űrlapot XHTML állományban** készítsük el (ne Java kód rakja össze).
- Az eredmény oldalt egyelőre Java kódban fogjuk összerakni.
- Annotációk helyett ismerjük meg a `web.xml` állomány-t!
- Új ismeret lesz az **űrlap paraméterek feldolgozása** Java-ban.
- Hogyan tároljuk el a már regisztrált személyeket? Adatbázis hiányában itt **Singleton pattern**-t fogunk alkalmazni.

proj: PersonRegister (pr.war)



- registerpage.html
  - action="perreg"
- web.xml
- Register.java
  - doPost()
  - checkParams()
  - responseContent()
  - errorsHtml()
  - peopleHtml()
- Person.java
- PersonCatalog.java

http://localhost:8080/pr/registerpage.html

Új feladat - Webes számológép

Készítsünk alapműveletes számológépet! Két egész érték bevitele és a köztük definiált művelet (+, -, *, /) meghatározását követően a webalkalmazás az eredményt kiszámolja az űrlap elküldése után. Nullával való osztás esetén 0 legyen az eredmény.. (ne haljon meg a program..)

Részletek

Az eredmény oldal jelenleg ugyanaz az oldal legyen, ahol az űrlapot elküldjük. Mivel az eredmény oldalon is meg kellene jeleníteni mind a két operandust és az operátort is, valamint így azonnal indíthatunk új kalkulációt!

Ehhez pár dolgot meg kell oldani:

- az elküldött űrlap paraméterek legyenek az alapértelmezettek az űrlapon a visszatöltést követően
- az eredmény oldal mivel megegyezik a kiindulási oldalal (egy weboldalunk lesz), ezért "mindegyiket" XHTML alakban adjuk meg!
  - E kapcsán bevezetjük a **JSP lapokat**!

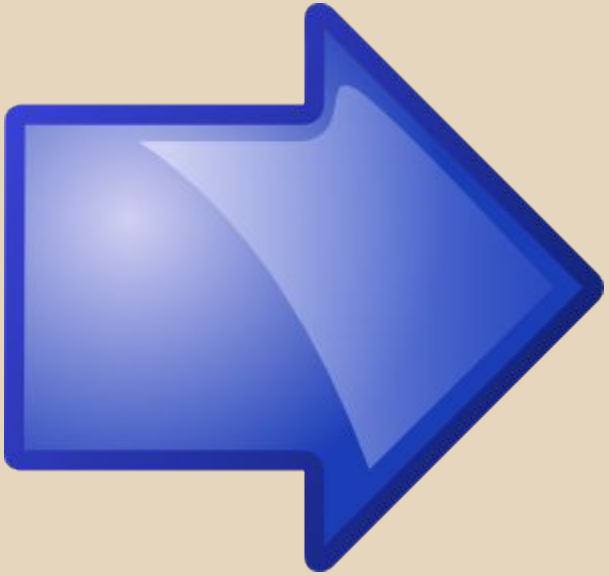
JSP

Nagyon röviden a JSP egy kényelmesebb módja a Servlet készítésnek. A webkonténer egy Servlet-et generál belőle, majd ezt követően a "jsp"-nek már nem sok szerepe lesz. A Servlet URL-je a jsp neve lesz alapértelmezés szerint.

A JSP nem ad különösebb támogatást az XHTML és a Java kód szétválasztására, de legalább mindegyiket a neki megfelelő kényelmes formába implementálhatjuk!

Ritkán használjuk a JSP-t önmagában. Általában a Servlet chain legvégén van, előtte egy Java Servlet végzi az előfeldolgozást, a JSP-ből generált Servlet-nek már csak a megjelenítés a dolga (servlet oldali üzleti kód, az ún. controller felelősség nem).

proj: CalculatorWeb (calc.war)



- Build path in eclipse: jsp-api.jar
- index.jsp

http://localhost:8080/calc/index.jsp

BookStore

Készítsünk egy webalkalmazást, melyen keresztül könyvek adatait lehessen böngészni.

Könyv: ISBN szám, szerző, cím, ár, oldalak száma, kategória (Történelem, Irodalom, Sci-Fi, Filozófia, stb.)

Listázó oldalon legyen látható a rendszerben tárolt összes könyv. Ez a lista szűrhető legyen kategóriára nézve.

A listában a könyv címére kattintva a könyv részletező oldalára tudjunk navigálni.

Ismeretszerzés

A feladat kapcsán megismerjük a **Model-View-Controller** alapjait a Java webalkalmazás fejlesztés kapcsán, felhasználva a már ismert technológiákat (ami jelen esetben a JSP-t jelenti).

Megismerjük a Servlet-ek közti **átirányítás** lehetőségeit.

A könyveket adatbázis hiányában Singleton pattern szerint fogjuk elérni, de itt is törekszünk arra, hogy az adatok elkérését egy **“service” rétegen** keresztül oldjuk meg.

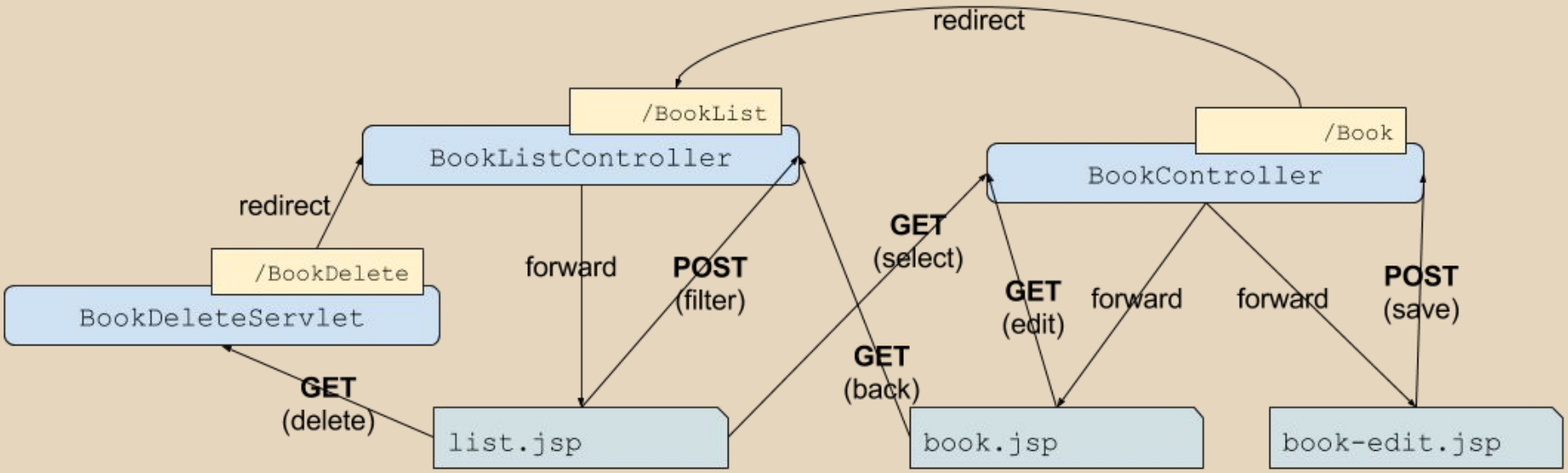
Átirányítás

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)  throws ServletException, IOException {

    response.sendRedirect("BookList");

    RequestDispatcher view = request.getRequestDispatcher("book.jsp");
    view.forward(request, response);
}
```

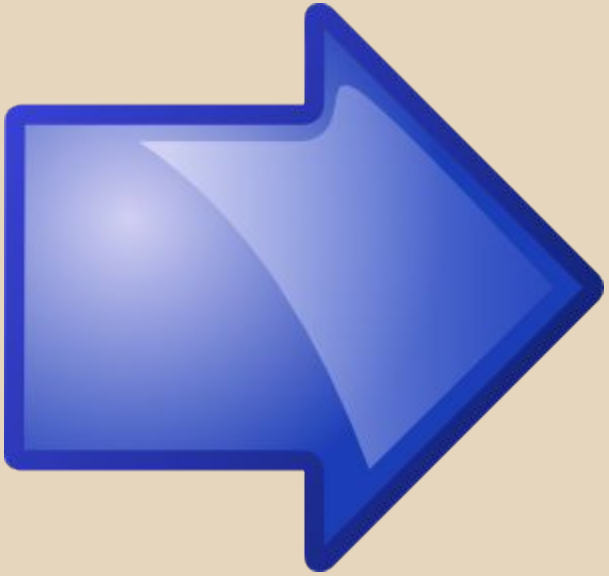
A megadott Servlet URI-t meghívja. A Redirect egy **új HTTP Request**-et jelent!

A megadott Servlet URI-nak továbbítja az **aktuális HTTP Request**-et (servlet chain)!

Áttekintés


redirect
/BookList
BookListController
/Book
BookController
redirect
/BookDelete
BookDeleteServlet
forward
POST
(filter)
GET
(select)
GET
(edit)
forward
forward
POST
(save)
GET
(delete)
GET
(back)
list.jsp
book.jsp
book-edit.jsp

proj: BookStore (bookstore.war)



- MVC
- Service layer

http://localhost:8080/bookstore/BookList
http://localhost:8080/bookstore/Book?isbn=978-0684824710

Új feladat - Egyszerű fórum

Készítsünk egy egyszerű fórum alkalmazást! Böngészőbe beírva listázzuk ki a memóriában tárolt (singleton) bejegyzéseket!

Amennyiben nincs bejelentkezve senki, egy link segítségével irányítsuk át egy bejelentkező oldalra! Itt egy felhasználó név és jelszó megadását követően léptessük be a figurát. Hiba esetén ne tudjon belépni! A felhasználói adatokat szintén memóráiban tároljuk el (másik singleton példány).

Sikeres bejelentkezés után a lista oldal visszajön, azonban már a "login" hivatkozás helyett egy "logout" szerepel, és egy űrlap, melyen keresztül új hozzászólás vihető fel!

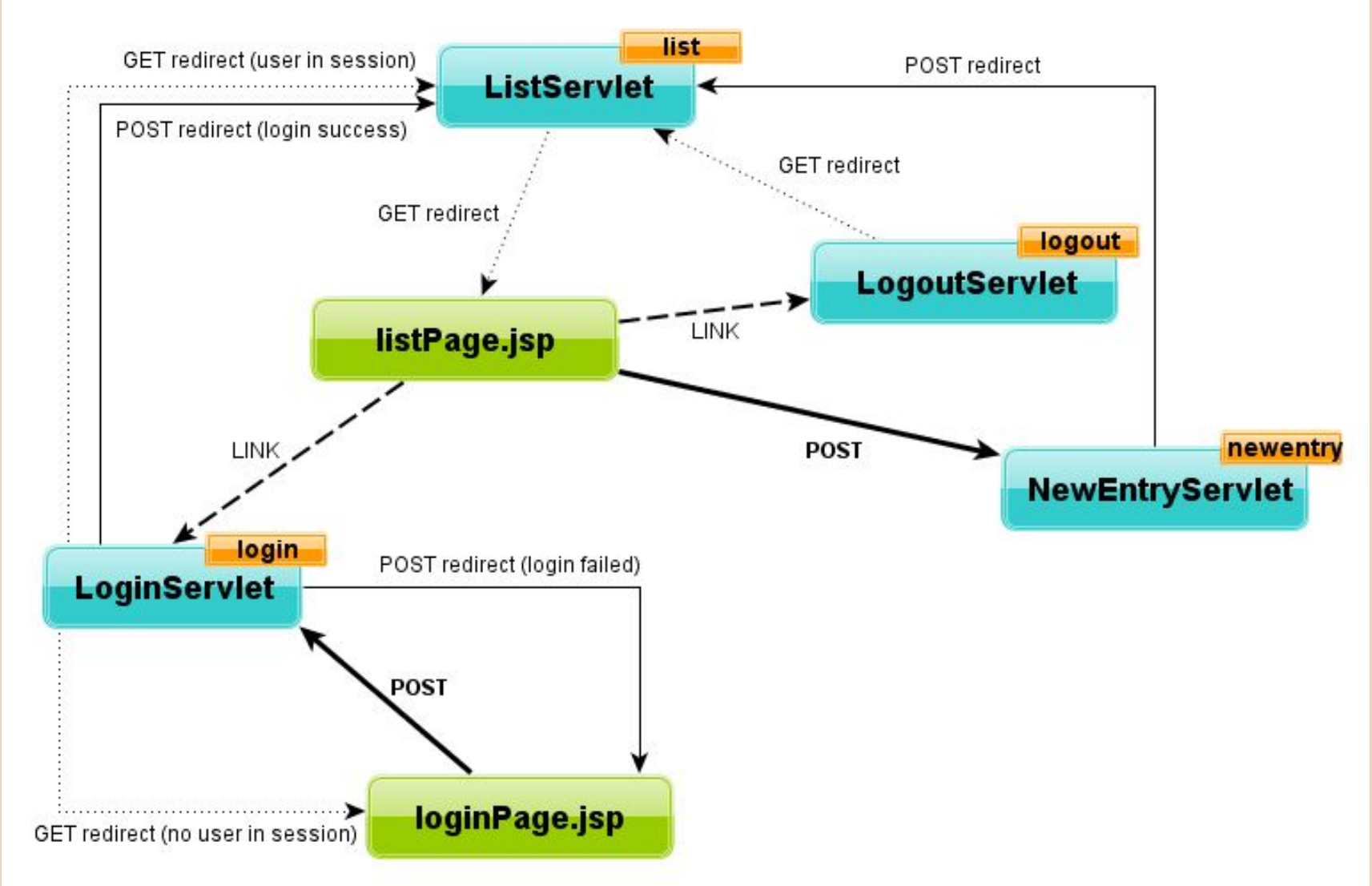
Új ismeretek

A HTTP állapotmentes protokoll, most mégis meg kell oldalunk valahogyan azt, hogy egy személy bejelentkezés után máshogy lásson egy weboldalt (bejegyzések listája), mint a nélkül!

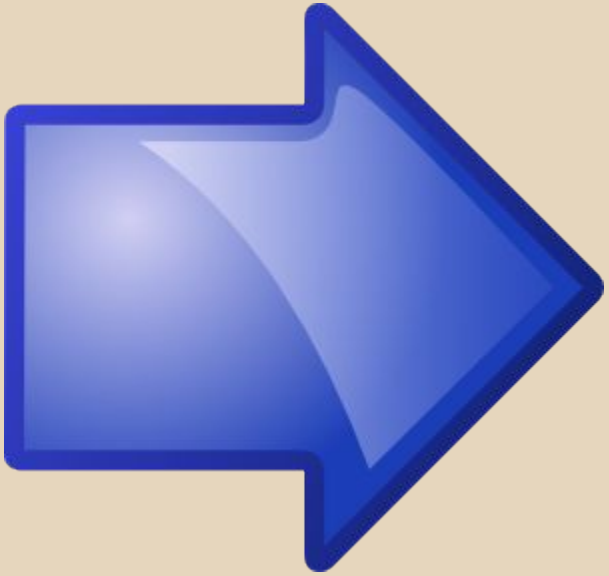
Erre megoldás az úgynevezett menetkövetés, melynek mi egyetlen formáját ismerjük meg ezen program kapcsán: **session kezelés**.

Szintén egy apró újdonság lesz egy Servlet **init paraméter**einek megadása!

Áttekintés


list
ListServlet
GET redirect (user in session)
POST redirect (login success)
POST redirect
GET redirect
GET redirect
logout
LogoutServlet
listPage.jsp
LINK
LINK
POST
newentry
NewEntryServlet
login
LoginServlet
POST redirect (login failed)
POST
loginPage.jsp
GET redirect (no user in session)

proj: SimpleForum (forum.war)



```
web.xml (ListServlet init
params)
ListServlet.java init()
ListServlet.java doGet()
listPage.jsp
LoginServlet.java doGet()
loginPage.jsp
LoginServlet.java doPost()
listPage.jsp
NewEntryServlet.java doPost()
listPage.jsp
LogoutServlet.java doGet()
```

http://localhost:8080/forum/list

Gyakorló feladat - Social network

Készítsünk egy egyszerű “social network” alkalmazást Disney figurák számára!

Az alkalmazás egy kezdőképernyőn induljon, ahol egy táblázatban láthatjuk, hogy mely rajzfilmfigurák vannak az “adatbázisban” (singleton), és melyik filmből ismerhetjük őket.

A táblázat egyes soraira kattintva egy részletek oldal töltődjön be, ahol a figura profil képe, telefonszámai illetve barátai láthatóak. A barátai hasonló táblázatban jelenjenek meg, mint a kezdőképernyőn, és hasonlóan kattinthatóak legyenek a sorai!

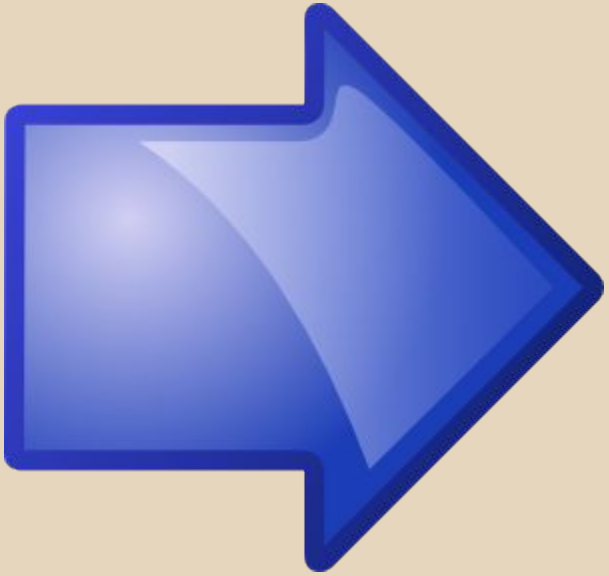
Megjegyzések, új ismeretek

Az adatokat a server oldalon “file”-ból olvassuk be a Singleton példányba.

Tisztán JSP-s megoldást adjunk a feladatra.

A listázó jsp állományt használjuk újra a kezdőképernyőn és a részleteket bemutató képernyőn (kerüljük el a redundanciát).

proj: DisneySocialNetwork (social.war)



```
index.jsp
→ list.jsp
member.jsp
```

http://localhost:8080/social/index.jsp

JSP Actions
Advanced JSP

- include

```
<jsp:include page="list.jsp">
    <jsp:param name="memberUnid" value="-1" />
</jsp:include>
```

- useBean, getProperty, setProperty

```
<jsp:useBean id="book" class="hu.qwaevisz.BookStub" scope="request" />
<jsp:getProperty name="book" property="author" />
<jsp:setProperty name="book" property="author" value="lorem ipsum" />
```

- További információ (pl.:)
  - http://www.tutorialspoint.com/jsp/jsp_actions.htm

JSP Expression Language
Advanced JSP

Szintaxis: **${expr}**

Legtöbb arithmetikai/logikai művelet használható.

Előre definiált objektumok:

- pageScope (local variables)
- **requestScope** (request attributes - any object)
  - ${requestScope.books.isEmpty()}
  - ${requestScope.books[0].price}
- **sessionScope** (session attributes - any object)
- applicationScope
- **param** (request parameters - string values)
  - ${param["username"]}
- header (HTTP request header)
  - ${header["user-agent"]}
- cookie (array of Cokkie(s))
- pageContext
  - ${pageContext.request.queryString}

Java Standard Tag Library
Advanced JSP

A JSP lapokban alkalmazható különféle Java blokkok sokszor nagyon zavaróak, amolyan “PHP hatást” keltenek, miközben bár van hasonlóság, a háttér természetesen teljesen más (a JSP lapból egy Servlet osztály készül, nem futás időben dolgozódik fel a script).

Elegánsabb ha TAG-eket használunk a különféle vezérlési szerkezetek megvalósítására. Erre szolgál többek között a JSTL.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

JSTL Core

- <c:out value=".."> használható <%= ... > helyett
- <c:set var="dummy" scope="session" value="${21*2}"/>
  - ha a változó object, akkor a property attribútum használható field elérésre
- <c:remove var="salary" scope="session"/>
- <c:import var="data" url="http://www.testline.hu"/>
  - <c:out value="${data}"/>
  - hasonló az <include> JSP Action-höz, de itt van lehetőség külső URL-ről is betölteni content-et

JSTL Core - Vezérlési szerkezetek I

- if

```
<c:if test="${dummy > 2000}">
    ..
</c:if>
```

else ág nem létezik a c:if esetén!

```
<% if (dummy > 2000) { %>
    ..
<% } %>
```

- choose

```
<c:choose>
    <c:when test="${dummy <= 0}">
        ..
    </c:when>
    <c:when test="${dummy > 1000}">
        ..
    </c:when>
    <c:otherwise>
        ..
    </c:otherwise>
</c:choose>
```

A **switch-case** esete, azonban így valósíthatjuk meg az **if-else** szerkezetet is.

JSTL Core - Vezérlési szerkezetek II

- forEach

```
<c:forEach var="i" begin="1" end="5" step="2">
   <c:out value="${i}"/>
</c:forEach>
```

számlálós ciklus megvalósítása

```
<c:forEach items="${requestScope.books}" var="book">
   <c:out value="${book.author}" />
</c:forEach>
```

foreach ciklus megvalósítása

```
<% for ( BookStub book : books) { %>
   <% out.print(book.getAuthor()); %>
<% } %>
```

Tag Library Descriptors (TLD)

Felmerülhet az igény arra, hogy saját tag-eket készítsünk annak érdekében, hogy ne kelljen Java kódokat elhelyeznünk feleslegesen a JSP lapokba, illetve ezek komoly redundanciáját is szeretnénk csökkenteni.

Pl. szeretnénk a könyvek árát megjeleníteni mindig úgy, hogy a valós érték egészre legyen kerekítve, illetve legyen egy “Ft” postfix-e.

Custom Tag készítése

PriceFormatterTag.java

```
public class PriceFormatterTag extends SimpleTagSupport {
    private static final String CURRENCY_SIGN = "Ft";
    private Double value;
    public void setValue(Double value) {
        this.value = value;
    }
    @Override
    public void doTag() throws JspException, IOException {
        NumberFormat formatter = new DecimalFormat("#");

this.getJspContext().getOut().write(formatter.format(value) +
" " + CURRENCY_SIGN);
    }
}
```

A **value** egy attribútuma lesz az általánunk használt tag-nek (a setValue() metódus miatt fogja ezt elvárni a tag).
A **doTag()** metódus feladata a content előállítása.

Tag leíró készítése (TLD)

WEB-INF/bookstore.tld

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
        version="2.0">
    <description>BookStore Custom Tags</description>
    <tlib-version>2.1</tlib-version>
    <short-name>booktag</short-name>
    <uri>http://qwaevisz.hu/jsp/tlds/booktag</uri>
    <tag>
        <name>formatPrice</name>

<tag-class>hu.qwaevisz.bookstore.weblayer.tag.PriceFormatterTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>value</name>
            <type>java.util.Double</type>
            <rtexprvalue>true</rtexprvalue>
            <required>true</required>
        </attribute>
    </tag>
</taglib>
```

A leíró több `<tag>`-et is tartalmazhat.

A tag neve **formatPrice** lesz (ezt nem kell hardcode-olnunk a Java osztályban).

Az `<rtexprvalue>` értéke teszi lehetővé hogy a value értéke lehessen-e avagy sem EL kifejezés.

Tag leíró bekötése

WEB-INF/web.xml

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <jsp-config>
        <taglib>

<taglib-uri>http://qwaevisz.hu/jsp/tlds/booktag</taglib-uri>
            <taglib-location>/WEB-INF/bookstore.tld</taglib-location>
        </taglib>
    </jsp-config>

</web-app>
```

Custom Tag használata

list.jsp

```
<%@ taglib uri="http://qwaevisz.hu/jsp/tlds/booktag" prefix="bt"%>
..
<c:forEach items="${requestScope.books}" var="book">
    ..
    <bt:formatPrice value="${book.price}"/>
    ..
</c:forEach>
..
```

Kérdések

?