Интернет програмиране с Java – част 7

Автор: Светлин Наков,
СУ "Св. Климент Охридски"
Web-site: http://www.nakov.com

Последна промяна: 26.10.2002

В предходната част от курса се запознахме с основните концепции в Web-програмирането, обяснихме какво е Java сървлет, как се създават сървлети и как се изпълняват със сървър Tomcat. В тази част ще продължим изучаването на Java сървлетите, ще покажем как сървлетите могат да приемат данни въведени от потребителя във вид на параметри, ще обясним жизнения цикъл на сървлетите и как те могат да работят с много потребители едновременно като използват HTTP сесии.

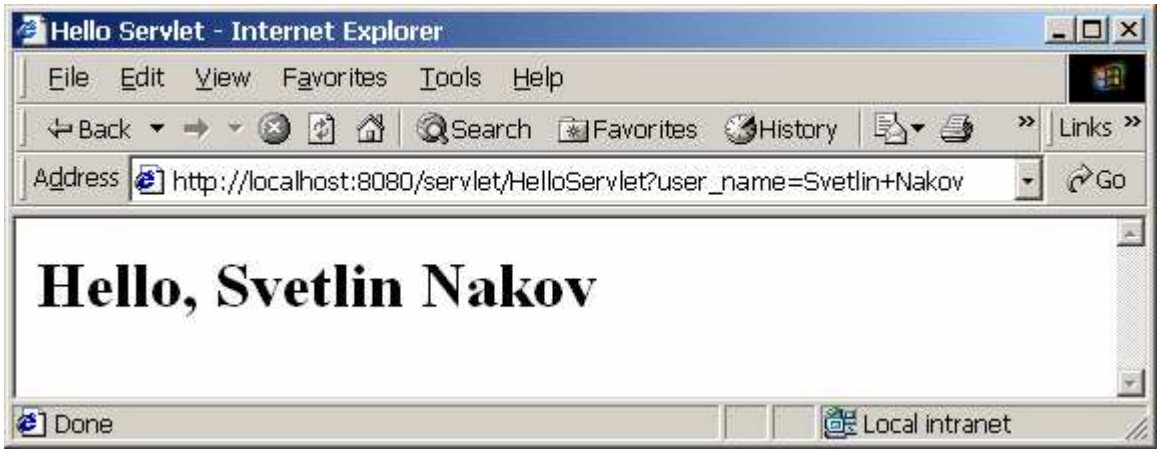
HTML форми и извличане на данните от тях

Всички сме използвали машини за търсене в Интернет като Google и AltaVista и знаем, че те представляват Web-приложения, които приемат от потребителя няколко ключови думи и му намират страниците, в които тези думи се срещат. Вероятно всеки е забелязал че след задаване на заявката за търсене в полето за адрес на браузъра се появява ново URL съдържащо въведената фраза за търсене замаскирана сред множество странни символи. Например ако в Google зададем търсене на фразата "Svetlin Nakov", ще получим URL подобно на това: http://www.google.com/search?q=Svetlin+Nakov&ie=windows-1251&hl=bg&lr=. Частта от URL-то след въпросителния знак съдържа данните, изпратени като параметри към това URL, кодирани по специален начин, наречен URL-encoding. Данните от HTML форма могат да бъдат предадени към сървъра по два начина – с GET или POST заявка. При GET HTTP заявки, те се предават след URL-то като се отделят от него с въпросителен знак, а при POST HTTP заявки се предават заедно със заявката, отделени от URL-то на отделен ред.

Java сървлетите имат вградена възможност за извличане на изпратените от потребителя данни. За целта се използва методът getParameter() на класа HttpServletRequest. Този метод връща стойността на параметър по зададено име или null ако такъв параметър не е изпратен от браузъра на потребителя. Парсването на параметрите и декодирането им от URL-encoded формат в чист текст става напълно автоматично, т.е. програмистът не е необходимо да се грижи за отделянето на параметрите един от друг, за отделянето им от URL-то и за декодирането на символите, които са били заменени с други съгласно с цел да се избегнат (escaped symbols). Ето един примерен сървлет, който демонстрира леснотата с която се получават параметрите. Той получава като вход име на потребител (параметър с име user_name) и му казва "здравей".

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

public class HelloServlet extends HttpServlet {
    public void doGet (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        resp.setContentType("text/html");
        ServletOutputStream out = resp.getOutputStream();
        String userName = req.getParameter("user_name");
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello Servlet</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Hello, " + userName + "</H1>");
        out.println("</BODY></HTML>");
    }
}
```

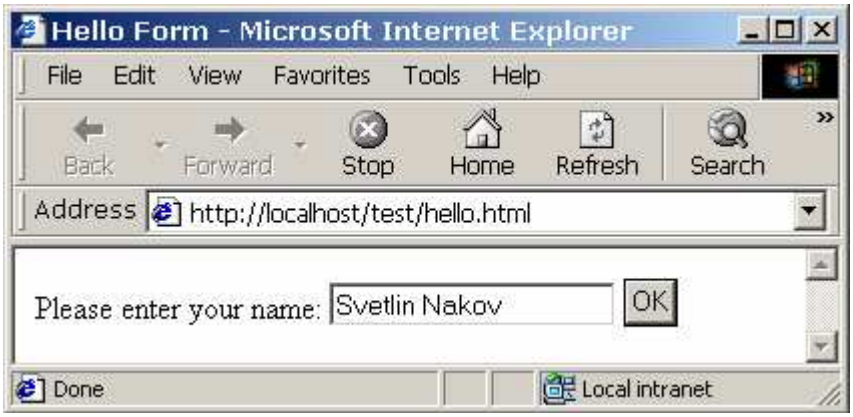
Ето и резултатът от изпълнението на този сървлет с параметър "Svetlin Nakov" :



Можем да направим HTML форма, в която потребителя си пише името и след натискане на "submit" бутона това име се подава като параметър на сървлета HelloServlet. Ето един пример:

```
<html>
<head> <title> Hello Form </title> </head>
<body>

<form method="GET" action="/servlet/HelloServlet">
Please enter your name:
<input type="text" name="user_name">
<input type="submit" value="OK">
</form>

</body></html>
```

Примерната HTML форма задава за метод на HTTP зявката GET. Така при натискане на "submit" бутона въведения в полето user_name текст се предава като долепен до URL-то параметър с име "user_name" и стойност въведената в текстовото поле. Ето как изглежда формата в браузъра:

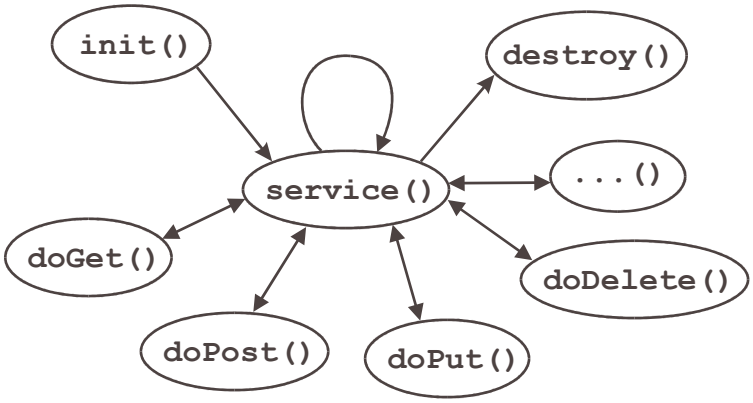


По идея HTML формите служат за автоматизация на процеса на предаване на параметри между потребителския браузър към сървърски скриптове, които обработват тези параметри. Името на скрипта, който браузърът извиква при submit-ване на формата, се задава в атрибута "action", а методът на HTTP заявката – в атрибута "method" на тага <form>. Във формата се задават различни текстови и други полета, като им се задават имена. Зададените имена съвпадат с имената на параметрите, които се генерират при създаване на HTTP заявката към сървъра. Специалният бутон "submit" служи за изпращане на данните, въведени във формата, към посочения скрипт. При изпращането на попълнените във формата данни браузерът се грижи да ги кодира по стандарта "URL-encoding" и да ги изпрати през URL-то или като част от заявката в зависимост дали методър на формата е GET или POST. При използването на GET метод всички параметри се долепват в URL-то, поради което обемът им не може да бъде много голям. При използването на POST метод всички параметри се предават скрито, не през URL-то и потребителят вижда само името на скрипва, който е обработил данните, но не и самите данни. Кой от двата метода да се използва е въпрос на преценка от страна на Web-разработчика.

За удобство на програмиста класът `HttpServletRequest` има и метод за изброяване на всички изпратени параметри `getParameterNames()`. Трябва да се знае, че при имената на параметрите малките и главните букви се различават.

Жизнен цикъл на сървлетите

Жизненият цикъл на сървлетите описва тяхното поведение от момента, в който те бъдат създадени като обекти на сървъра (инстанцирани) до момента на тяхното премахване от него. На картинката са показани основните методи, който реализират жизнения цикъл на сървлетите и последователността на тяхното извикване:



При първо извикване на сървлета сървърът, който изпълнява сървлетите и JSP скриптовете (така наречения Web-контейнер), извиква метода init(), дефиниран в класа HttpServet. Сървлетите, които имат нужда от еднократна първоначална инициализация преди започване на работата си, трябва да припокрият този метод и да реализират тази своя инициализация. Например един сървлет може да прочете от сървъра или от някакъв файл конфигурационна информация, която да използва по-нататък.

При настъпване на заявка за достъп до сървлета, подадена от Web-браузъра на някой клиент, сървърът извиква метода service() от класа HttpServlet. Този метод анализира типа на заявката и взависимост от това дали заявката е GET, POST, PUT, DELETE или друга, извиква съответно един от методите doGet(), doPost(), doPut(), doDelete() и т.н. Понеже HTTP методите PUT, DELETE, HEAD, TRACE, OPTIONS и т.н. се използват много рядко и не са типични за повечето сървлети, няма да ги разглеждаме. Методът doGet(), трябва да бъде реализиран от сървлетите, които обработват заявки подадени по метод GET, например ако обработват данните, получени от HTML форми, за които е зададено method="GET". Аналогично doPost() методът трябва да бъде реализиран, когато трябва да се обработят данни получени от HTML форми, за които методът е POST. Ако искаме да направим сървлет, който обработва едновременно и GET и POST заявки, не е хубаво да припокриваме директно метода service(), защото той се грижи за правилната обработка на HEAD заявките и за още други важни неща. Вместо това можем да имплементиране обработката на данните в метода doGet(), а от doPost() просто да извикваме doGet(). Това е препоръчителният начин за обработка на данни, които се очаква да пристигат и чрез GET и чрез POST заявки.

След като сървлетът бъде изпълнен веднъж, той остава като обект в паметта на виртуалната машина на Java и при следващи извиквания се изпълнява веднага, без да се зарежда от .class файла отново. В рамките на едно Web-приложение един сървлет се инстанцира само веднъж. Когато няколко клиента поискат един сървлет едновременно, Web-контейнерът стартира едновременно няколко нишки (threads) и извиква от всяка от тях сървлета в един и същ момент. Понеже всеки сървлет има само една инстанция в сървъра, то класът, който реализира този сървлет, заедно с член-променливите, които са дефинирани в него, се инстанцират само веднъж в рамките на Web-приложението. Следователно работата с тези член-променливи не е thread-safe, т.е. не е обезопасена от проблеми с конкурентния достъп при заявки от няколко потребителя едновременно. Затова е необходимо програмистът да има предвид, че е възможно кодът на единствената инстанция на написания от него сървлет да се изпълнява едновременно от няколко

нишки (threads) и затова трябва да се грижи за синхронизация на достъпа до член-променливите на сървлета, както и другите ресурси, които използва.

Когато сървърът по някаква причина реши да премахне от паметта един сървлет (например при намеса на администратора), се извиква метода destroy() на класа HttpServet. В реализацията на този метод сървлетите трябва да освободят заетите от тях ресурси и да финализират работата си. Типичен пример за използване на init() и destroy() методите при прости приложения е за отваряне и затваряне на връзката към базата данни, когато се използва такава. При по-сложни приложения връзката към базата данни се управлява от специална компонента на системата известна като “connection pool”. Методът destroy() се използва по-рядко от метода init(), защото Java освобождава автоматично някои типове ресурси, като например паметта, при унищожаването на обект. Ето един пример за сървлет, който използва методите init(), destroy(), doGet() и doPost() и същевременно демонстрира как от сървлет може да се генерира динамично JPEG изображение и да се върне като отговор на клиентската заявка:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.awt.*;
import java.awt.image.*;
import com.sun.image.codec.jpeg.*;
import java.io.*;
import java.util.Date;

public class ImageServlet extends HttpServlet {
    private int mVisitCounter;
    private String mStartDate;

    public void init() {
        mStartDate = new Date().toString();
        mVisitCounter = 0;
    }

    public BufferedImage createImage(String msg) {
        Font font = new Font("Serif", Font.BOLD, 24);
        FontMetrics fm =
            new Canvas().getFontMetrics(font);
        int width = fm.stringWidth(msg) + 20;
        int height = fm.getHeight();
        BufferedImage image = new BufferedImage(
            width, height, BufferedImage.TYPE_INT_RGB);
        Graphics g = image.getGraphics();
        g.setColor(Color.red);
        g.fillRect(0, 0, width, height);
        g.setFont(font);
        g.setColor(Color.black);
        g.drawString(msg, 12, fm.getAscent()+2);
        g.setColor(Color.yellow);
        g.drawString(msg, 10, fm.getAscent());
        return image;
    }

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    throws IOException, ServletException {
        String msg;
        synchronized(mStartDate) {
            mVisitCounter++;
            msg = "Page visited " + mVisitCounter +
                " times since " + mStartDate;
        }
        BufferedImage image = createImage(msg);
        response.setContentType("image/jpeg");
        OutputStream out = response.getOutputStream();
        JPEGImageEncoder encoder =
            JPEGCodec.createJPEGEncoder(out);
        JPEGEncodeParam jpegParams =
            encoder.getDefaultJPEGEncodeParam(image);
        jpegParams.setQuality(1, false);
        encoder.setJPEGEncodeParam(jpegParams);
        encoder.encode(image);
        out.close();
```

```
    }

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

При инициализация, в метода init(), сървлетът запомня във вътрешна член-променлива датата и часа, в който е инициализиран. В друга вътрешна член-променлива той помни и броя пъти, които е бил извикван чрез GET или POST заявка. При извикване на doGet() метода, сървлетът генерира текстово съобщение, което съобщава колко пъти е била посетена страницата от първото извикване на сървлета. Заради възможността няколко потребителя едновременно да поискат страницата, е необходимо достъпът до член-променливите на сървлета да бъде синхронизиран. Както знаем от темата за многонишково програмиране и синхронизация, в Java синхронизацията може да се прави по монитора на произволен обект. В нашия случай синхронизираме по обекта, съдържащ датата и часа на инициализация на сървлета. Понеже, както споменахме по-горе, сървлетите се инстанцират само веднъж в рамките на едно Web-приложение, член-променливите им също се инстанцират само веднъж и затова броячът на посетители има само едно копие в паметта, въпреки, че не е обявен като static. Благодарение на синхронизирания достъп до него, той отчита посетителите правилно, дори при конкуретно извикване от много потребители едновременно. За динамичното генериране на изображението се използват стандартните средства на jawa.awt. Изчисляват се размерите на текста при шрифта, който ще бъде използван, след това се създава изображение от класа BufferedImage, взема се неговия обект за графична манипулация Graphics и се изобразява текста. Първо изображението се запълва с червен цвят, след това се печата текста, малко отместен с черен цвят, а след това се печата с жълто същия текст на истинската му позиция. Така се получава текст със сянка. След като изображението е изготвено, то се конвертира в JPEG поток от данни като се използва кодекът на Sun за JPEG кодиране и му се задава да работи с максимално ниво на качество. Полученият поток от данни се изпраща като изход от сървлета. За да може Web-браузърът да разпознае изпратения поток от данни като JPEG картинка, а не като текст, в header-а на HTTP отговора на заявката се слага “Content-type: image/jpeg”. По подразбиране, ако сървлетът не укаже друго, за тип на изпратените данни се слага “Content-type: text/html”. Реализацията на метода doPost() просто извиква метода doGet(), което позволява на сървлета да отговаря както на GET, така и на POST заявки по протокола HTTP. Ето и примерен резултат от извикването на сървлета:

Поддръжка на потребителски сесии

Когато говорихме за Web-приложения, споменахме, че те имат възможност да обслужват едновременно много потребители, независимо един от друг. Как знаем, HTTP протоколът има несесиен характер, т.е. не ни предоставя възможност да различаваме потребителите един от друг и да проследяваме коя заявка от кой потребител идва. Ето защо за проследяване и разграничаване на потребителите един от друг са необходими допълнителни усилия, които Web-приложенията

трябва да полагат. Потребителска сесия наричаме периода, в който един потребител си взаимодейства с едно Web-приложение. Проследяването на последователността от заявки, извършени от един потребител се нарича проследяване на неговата сесия. Ако два потребителя работат едновременно с една Web-система, те имат две различни сесии. Пример за Web-приложение, което проследява потребителската сесия, е Web-базираната система за електронна поща на Yahoo. Всички знаем, че е възможно докато един потребител си чете пощата от mail.yahoo.com, друг потребител, напълно независимо от него също да си чете пощата от същия сайт. Web-приложението за електронна поща, работещо на машината с име mail.yahoo.com разпознава различните потребители и проследява техните сесии. Взависимост от това кой потребител е дал HTTP заявка към Web-приложението, сървърът разпознава неговата сесия и дава достъп до неговите email-и, а не до тези на останалите потребители, работещи в същия момент. Възможно е от един и същ компютър да се осъществят няколко независими сесии към едно и също Web-приложение. Например потребителят може да отвори два различни Web-браузъра – един Netscape и един Opera и да влезе в едно Web-приложение като два различни потребителя. В рамките на браузъра Netscape, той ще има създадена една сесия със сървъра, а в рамките на браузъра Opera той ще има създадена още една, независима от първата сесия със същия сървър. Това означава, че за сървъра потребителите са различими един от друг, дори когато идват от един и същ компютър. Това се обяснява с механизма, по който сървърът различава потребителите един от друг. Има два основни начина за проследяване на потребителската сесия – с cookies и с добавяне на допълнителен параметър към URL-то. Cookies е възможност едно Web-приложение да чете и записва информация на машината на клиента. Информацията от cookies може да се чете само от приложението, което я е записало и може да изчезва ако не се ползва дълго време, взависимост от параметрите зададени при създаването й. Посредством cookies Web-приложенията могат да записват на машината на потребителя някакъв идентификатор на сесия и след това като я прочитат при всяка заявка, да разпознават потребителя. Другият начин за следене на потребителските сесии е чрез добавяне на допълнителен параметър към URL-то. При започване на работа на потребителя се генерира уникален ключ и той се добавя като параметър при всяка GET или POST заявка. Този подход изисква допълнителни усилия за добавяне на скрити полета във всяка HTML форма и добавяне на параметри към всеки hyperlink и затова се използва рядко, обикновено когато клиентският браузър не поддържа cookies или потребителят ги е забранил. С cookies усилията за проследяване на потребителите са значително по-малки.

В Java сървлетите и JSP скриптовете поддръжката на потребителски сесии е напълно автоматична. Програмистът не е нужно да изпраща и чете cookies или да добавя и разпознава след това допълнителни параметри към URL-то. Достатъчно е да се използва API-то за работа със сесии, което се дава от framework-а за Web-приложения. Основен е класът HttpSession, който представя потребителската сесия. Получаването на обект, асоцииран с текущата сесия можем да вземем от HttpRequest обекта по следния начин:

```
HttpSession session = request.getSession();
```

Ако сесия не съществува такава ще бъде създадена. Взимането на сесията за един и същ потребител връща един и същ обект, а за различни потребители връща различни обекти. За всеки нов потребител се създава нов обект от класа HttpSession и се връща този обект. В обекта session могат да се съхраняват произволни данни за потребителя посредством методите сетAttribute(key, value) и getAttribute(key). Веднъж съхранени в сесията, тези обекти са достъпни по време на всяка заявка от потребителя в рамките на тази сесия. Ето един типичен пример за използване на сесия е когато на всеки потребител се предоставят различни ресурси в зависимост от това като какъв се е автентикирал, а неавтентикираните потребители не се допускат. Представяме си сорсовете на сървлета LoginServlet:

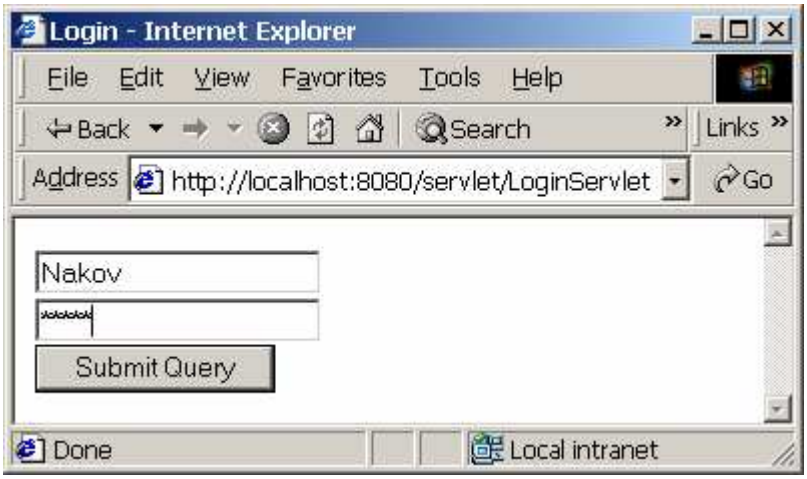
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    throws IOException, ServletException {
```

```
        String user =
            request.getParameter("user");
        String password =
            request.getParameter("password");
        PrintWriter out = response.getWriter();
        if ((user==null) || (password==null)) {
            showLoginForm("", out);
        } else if (user.equals(password)) {
            HttpSession session =
                request.getSession();
            session.setAttribute("USER", user);
            response.sendRedirect(
                "/servlet/MainServlet");
        } else {
            showLoginForm(
                "Invalid login.<br>", out);
        }
    }

    private void showLoginForm(
        String captionText, PrintWriter out)
    {
        out.println(
            "<html><title>Login</title>\n" +
            "<form method=\"GET\" action=" +
            "\"/servlet/LoginServlet\">\n" +
            captionText +
            "<input type=\"text\"" +
            " name=\"user\"><br>\n" +
            "<input type=\"text\"" +
            " name=\"password\"><br>\n" +
            "<input type=\"submit\">\n" +
            "</form></html>"
        );
    }
}
```

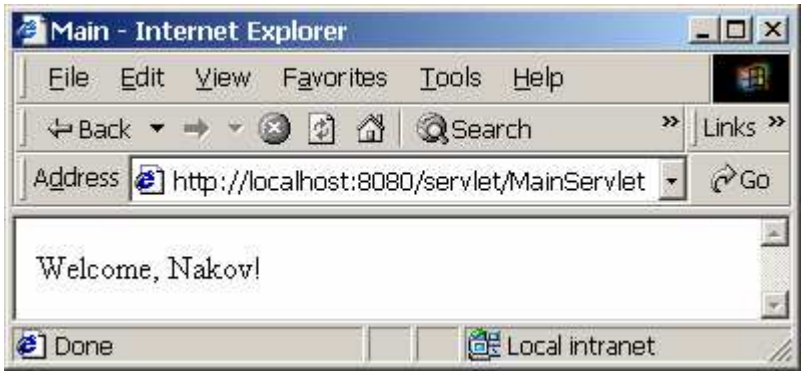
и сървлета MainServlet:

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

public class MainServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    throws IOException, ServletException {
        HttpSession session =
            request.getSession();
        String user = (String)
            session.getAttribute("USER");
        PrintWriter out = response.getWriter();
        if (user==null) {
            showMainForm("Not authenticated." +
                " Please <a href=\""+
                "/servlet/LoginServlet\">" +
                "login</a> first.", out);
        } else {
            showMainForm(
                "Welcome, " + user + "!", out);
        }

    }

    private void showMainForm(
        String captionText, PrintWriter out)
    {
        out.println(
            "<html><title>Main</title>\n" +
            captionText + "</html>"
        );
    }
}
```

Първият сървлет (LoginServlet) служи за автентикация на потребителите. Когато се извика без параметри, той показва HTML форма за попълване на име на потребител и парола:



Когато се попълни и изпрати формата, се извиква същият сървлет, но въведените име на потребител и парола се изпращат към него като параметри. Когато сървлетът разпознае валидна комбинация от потребителско име и парола, записва в автоматично създадената за текущия потребител сесия под ключ "USER" въведеното потребителско име. За простота в нашия пример валидни комбинации от потребителско име и парола са всички, в които потребителското име съвпада с паролата. След успешна автентикация потребителят се препраща към основния сървлет (MainServlet) чрез response.sendRedirect(URL). Основният сървлет разпознава неговата сесия по съществуването стойност под ключа "USER", прочита от нея името на автентикирания потребител и го поздравява с кратко съобщение:



Ако в същия момент друг потребител се опита да зареди MainServlet-а, като напише адреса му в браузъра си, той ще бъде разпознат от системата като различен от първия. В неговия HttpSession обект няма да има никаква стойност под ключ "USER" и така основният сървлет ще разбере, че този потребител не е автентикиран:

Както видяхме проследяването на потребителската сесия става автоматично и в нея могат да се записват различни обекти под различни ключове, като се гарантира, че HttpSession обекта за всеки различен потребител е различен. Можем да задаваме времето на неактивност в милисекунди, за

което една сесия изтича и се изтрива от сървъра чрез метода setMaxInactiveInterval() на класа HttpSession. Изтичането на сесиите (session expiration) е полезно от съображения за сигурност. След като един потребител е автентикиран веднъж, неговата автентикация важи само определено време, за да не може след като стане от компютъра някой друг да седне на негово място и да използва сесията му. Потребителският интерфейс на Web-приложенията обикновено освен автентикация и оторизация (login) предлага и изход от системата (logout), което прекратява потребителската сесия. Прекратяването на сесията на потребител в нашия пример може да стане като се изтрие стойността с ключ “USER” от сесията чрез session.removeAttribute(“USER”).