Serwlety i JSP

Autor: Marek Zawadka
deekay@gazeta.pl

Plan prezentacji

- Wstęp
- CGI i inne wcześniejsze rozwiązania
- Serwlety
  - ulepszenia
  - klasa HTTPServlet
  - obsługa sesji
  - wielowątkowość
- JSP
  - czym się różni od serwletów
  - elementy skryptowe
  - dyrektywy
  - elementy akcji
- Metody poprawienia wydajności w serwletach i JSP
- Podsumowanie
- Źródła

Internet

- Miliony komputerów
- Wymiana plików/dokumentów/informacji
- Poczta elektroniczna, IRC, listy dyskusyjne
- FTP
- WWW (World Wide Web)
  - Przeglądarka www
  - Serwer
  - Przeglądarka wysyła żądanie w formie adresu URL, serwer pobiera plik dokumentu z dysku i wysyła go do użytkownika
- W celu ujednolicenia formatu dokumentów udostępnianych przez usługę WWW wprowadzono język opisu dokumentów HTML
- Istnieje szereg rozwiązań technologicznych, które ułatwiają realizację oprogramowania pracującego w sieci Internet

Generowanie i udostępniania dokumentów

- **Użytkownik może pobierać z serwera dokumenty wcześniej przygotowane**
- **Czasami treść dokumentu może się zmieniać co chwilę (dane z giełdy)**
- **Rozwiązania:**
  - **Zatrudnienie “klepacza(-y)” wprowadzającego dane**
  - **Dynamiczne tworzenie dokumentów**
- **CGI (Common Gateway Interface)**
- **Serwlety**
- **JSP**

CGI

- **The Dark Ages**
- **Standard komunikacji pomiędzy serwerem i klientem umożliwiający zdalne wykonanie pewnego programu umieszczonego na serwerze i przesłanie wyniku do klienta**
- **Program ten jest zapisywany na dysku serwera, a następnie uruchamiany na żądanie**
- **Wynikowy dokument HTML trafia do przeglądarki**
- **Jedno żądanie = jedno uruchomienie programu CGI**
- **Dodatkowy koszt czasowy na zatrzymanie się programu**
- **Brak bezpieczeństwa**
- **Różne API: Microsoft's ISAPI oraz Netscape's NSAPI (przystosowane do serwerów określonych producentów – kod głównie w C/C++)**
- **Brak solidności (źle ustawiony wskaźnik czy dzielenie przez zero wywala cały serwer)**

Serwlety (1/4)

- **Alternatywa dla CGI o podobnych zasadach działania**
- **Według Sun Microsystems:**

"a standard approach to extending server functionality without the limitations of cgi-based or server-specific approaches"

- **Serwlety realizują model programowania żądanie-odpowiedź**
- **Uruchamiane są wewnątrz serwerów przetwarzających zapytania i generujących odpowiedzi**
- **Rozszerzają one funkcjonalność serwerów**
- **Stanowią doskonałe rozwiązanie problemu programowania po stronie serwera. Są jedną z przyczyn przechodzenia na programowanie w Javie**

Serwlety (2/4)

Różnice (wada):

- **Java**

```
Integer ObjectTally = (Integer)
  hPoll.get ( "choice" );
int tally = ObjectTally.intValue();
tally++;
hPoll.put ( "choice", new Integer (tally) );
```

- **Perl**

```
$hPoll{'choice'}++;
```

Serwlety (3/4)

Różnice (zalety):

- **Serwlet pracuje nieprzerwanie w środowisku serwera WWW**
- **Piszemy w dobrze znanym języku Java**
- **Są wieloplatformowe**
- **Są szybkie**
- **Serwlety są bardziej "eleganckie" (programowanie obiektowe zapewnia lepsze zorganizowanie kodu, łatwiejszą pielęgnację kodu oraz lepsze zrozumienie go)**
- **Niezależność dla programisty od konkretnego typu serwera**
- **Serwlety są wielowątkowe a skrypty CGI nie**
- **Wzbogacenie serwera o szereg dodatkowych funkcji przydatnych programistom**
  - **mechanizmy bezpieczeństwa**
  - **komunikacja z bazami danych**

Serwlety (4/4)

- **Serwlet to klasa języka Java, która implementuje interfejs Servlet**
- **javax.servlet.***
- **Są one łatwiejsze do pisania dzięki wykorzystaniu środowiska Javy**
- **Są szybciej wykonywane, gdyż wywołanie serwletu odbywa się nie poprzez uruchomienie nowego procesu, co jest kosztowne ze względu na czas procesora i zasoby pamięciowe, lecz jako wątek**
- **Kod wykonywalny dla serwletu jest ładowany do serwera tylko raz, gdy po raz pierwszy żądana jest usługa oferowana przez dany serwlet lub automatycznie, gdy zostanie zmieniony kod serwletu**
- **Potem serwlet pozostaje w pamięci serwera i może równolegle obsługiwać wiele zapytań z możliwością komunikacji między nimi**
- **GenericServlet jest podstawową klasą serwletów i nie jest używana bo nic (ciekawego) nie robi. Implementuje interfejs Servlet**

Interfejs Servlet

```
public interface Servlet {
  public void init(ServletConfig config) throws ServletException;
  public ServletConfig getServletConfig();
  public void service(ServletRequest req, ServletResponse res)  throws ServletException, IOException;
  public String getServletInfo();
  public void destroy();
}
```

- **GetServletInfo()** - zwraca w postaci tekstowej informacje o o serwlecie: dane autora, wersja itp.
- **GetServletConfig()** - zwraca obiekt ServletConfig zawierający parametry inicjalizacyjne i i startowe serwletu

HTTPServlet (1/2)

- **Rozszerza klasę GenericServlet o obsługę protokołu HTTP**

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

public class MyServlet extends HttpServlet {
  int i = 0;
  public void service(HttpServletRequest req, HttpServletResponse res) throws
       IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.print("<HTML><HEAD><TITLE>");
    out.print("Przykladowy serwlet");
    out.print("</TITLE></HEAD><BODY>");
    out.print("<h1>Wywołanie numer: " + i++);
    out.print("</h1></BODY></HTML>");
    out.close();
  }
}
```

HTTPServlet (2/2) – obsługa formularzy

```
public class EchoForm extends HttpServlet {
  public void service(HttpServletRequest req, HttpServletResponse res) throws IOExcep-
      tion {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    Enumeration flds = req.getParameterNames();
    if(!flds.hasMoreElements()) {
      out.print("<html>");
      out.print("<form method=\"POST\"" + " action=\"EchoForm\">");
      for(int i = 0; i < 10; i++)
        out.print("<b>Field" + i + "</b> " +
          "<input type=\"text\""+" size=\"20\" name=\"Field" + i + "\" value=\"Value" + i +
        "\"><br>");
      out.print("<INPUT TYPE=submit name=submit"+"
        Value=\"Submit\"></form></html>");
    } else {
      out.print("<h1>Your form contained:</h1>");
      while(flds.hasMoreElements()) {
        String field= (String)flds.nextElement();
        String value= req.getParameter(field);
        out.print(field + " = " + value+ "<br>");
      }}out.close();
  }}
```

HTTPServlet – POST i GET

- *public void **doGet/doPost**(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException*

- **Uzyskanie parametrów żądania**

```
Enumeration hdrs = req.getHeaderNames();
 while(hdrs.hasMoreElements()) {
    String header = (String) hdrs.nextElement();
    String value = req.getHeader(header);
    out.print(header + " = " + value + "<br>");
 }
```

Obsługa sesji

- **HTTP jest bezstanowy => wprowadzenie mechanizmu pozwalającego programistom WWW na śledzenie sesji użytkownika nie było proste**

- **Metody śledzenia sesji:**
  - **Ciasteczka (Cookies)**
    - mała porcja informacji wysyłana przez serwer WWW do przeglądarki
    - przeglądarka przechowuje cookies na lokalnym dysku, a następnie przy jakimkolwiek odwołaniu do adresu URL, z którym cookie jest związane, wysyła je niejawnie wraz z żądaniem HTTP, dostarczając w ten sposób serwerowi pożądanej informacji o tym kto aktualnie się odzywa.
    - klient ma możliwość wyłączenia używania cookies w swojej przeglądarce
  - **przepisywanie adresu URL oraz dołączanie do niego session ID**
  - **pola ukryte formularzy**

Klasa Cookie

- **zdefiniowana w API serwletów (od wersji 2.0)**
- **Obejmuje ona w sobie wszystkie szczegóły dotyczące nagłówków HTTP oraz pozwala na ustawianie różnych atrybutów ciasteczek**
- **Użycie Cookie polega na prostym dodawaniu ich do obiektu odpowiedzi**
- **Konstruktor Cookie pobiera nazwę cookie jako pierwszy argument oraz jego wartość jako drugi**

  Cookie ciacho = new Cookie("nazwa", "wartość"):

- **Cookies są dodawane do obiektu odpowiedzi, zanim zaczniemy wysyłać zawartość strony**

  res.addCookie(ciacho);

- **Pobierane są przez wywołanie metody getCookie() obiektu HttpServletRequest**

  Cookie[] cookies = req.getCookies():

- **Możemy teraz np. wywołać metodę getValue()**

Klasa Session

- **Sesja to jedno lub więcej odwołań klienta do strony WWW na serwerze w określonym przedziale czasowym**
- **Serwletowy obiekt Session żyje po stronie serwera (przechowuje użyteczne informacje na temat klienta, gdy ten porusza się po naszej witrynie WWW)**
- **Wykorzystuje klasę Cookie**
- **Wszystko czego potrzebują obiekty Session to unikatowy identyfikator przechowywany po stronie klienta i przekazywany do serwera**
- **Metody**:
  - getSession() i getSession(true) //HttpServletRequest
  - setAttribute(String name, Object value) i getAttribute(String attr)
  - getAttributeNames()
  - getMaxInactiveInterval() i setMaxInactiveInterval(int interval)
  - getLastAccessedTime()

Serwlety a wielowątkowość

- **Kontener serwletów dysponuje pulą wątków które przydziela do obsługi żądań klientów**
- **Może się zdarzyć, że metoda service() danego serwletu wywołana zostanie równocześnie przez dwóch klientów**

```
public class ThreadServlet extends HttpServlet {
  int i;
//....
  public void service(HttpServletRequest req,
    HttpServletResponse res) throws IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    synchronized(this) {
      //sekcja krytyczna
    }
    out.print("<h1>Finished " + i++ + "</h1>");
    out.close();
  }
}
```

- **Interfejs SingleThreadModel (tylko jedna instancja metody service). Nie zawiera żadnych metod, jest tylko znacznikiem**

Synchronizacja

- **Przykład (not thread-safe)**

```
public class SimpleServlet extends HttpServlet
{
 private int counter = 0;

 public void doPost(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException
 {
      resp.getWriter().println("<HTML><BODY>");
      for (int c = 0; c < 10; c++)
      {
           resp.getWriter().println("Counter = " + counter + "<BR>");
           try
           {
                Thread.currentThread().sleep((long) Math.random() * 1000);
                counter++;
           }
           catch (InterruptedException exc) { }
      }
      resp.getWriter().println("</BODY></HTML>");
 } }
```

- public void doPost(HttpServletRequest req, HttpServletResponse resp)
  { private int counter = 0; .....}

Przekierowanie

- //HttpServletResponse res
  res.sendRedirect("http://www.mimuw.edu.pl");

- **RequestDispatcher**

  RequestDispatcher rd =
  getServletContext().getRequestDispatcher("/servlet/Welcome");
  //zwracany jest null jesli nie znaleziono

- **Forward**
  rd.forward(request, response);

- **Include**
  rd.include(request, response);

Tips and Tricks

- **Od momentu kiedy serwlet zostaje załadowany do pamięci podczas pierwszego uruchomienia, przy każdej zmianie kodu musi on być przeładowywany. W wielu przypadku wielu serwerów jak np. JRun (Macromedia) jest to robione automatycznie**

- **Kiedy uruchamiamy serwlety na serwerach wspierających aliasy są możliwe dwie ścieżki pozwalające dostać sie do danego serwletu**
  - **Ścieżka logiczna zdefiniowana przez serwer. Jest to standardowa ścieżka tak jak w skryptach CGI**

    http://webserver/servlets/SnoopServlet
  - **Ścieżka będąca aliasem. Np. w JWS może to wyglądać tak**

    http://webserver/SnoopServlet

- **Pisząc serwlety można o nich myśleć jak o transakcjach. Taka transakcja składać będzie się z żądania, wykonania działań związanych z żądaniem oraz odpowiedzi. Można napisać zestaw dynamicznych serwletów, które "złączone" razem będą powodować określone działanie (pełna transakcja)**

JSP (Java Server Pages)

- **Utrudnieniem dla programistów tworzących zarówno skrypty CGI jak i serwlety Java jest to, że programy muszą generować komplet-ne dokumenty HTML**
- **Często jednak pod pojęciem dynamicznej generacji rozumiemy modyfikację fragmentu, a nie całości dokumentu**
- **JSP: ręczne tworzenie dokumentów HTML, w których umieszczane są specjalne “wstawki”**
- **“Wstawki” te podmieniane są na dynamiczną zawartość podczas dostarczania dokumentu użytkownikowi**
- **Wszystko tłumaczone jest na serwlety**

ZALETA:

- **większa wydajność pracy programistów, którzy nie muszą już być odpowiedzialni za szatę graficzną dokumentów**

Bardzo prosty przykład

```
<HTML>
<HEAD>
<TITLE>Prosty przykład</TITLE>
</HEAD>
<BODY COLOR=#FFFFFF>
<FONT FACE=”Arial”>
Data: <%= new java.util.Date() %>
</FONT>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Prosty przykład</TITLE>
</HEAD>
<BODY COLOR=#FFFFFF>
<FONT FACE=”Arial”>
Data: Mon March 26 23:35:18
      GMT+01:00 2005
</FONT>
</BODY>
</HTML>
```

- **Czy kod Javy jest konieczny w pliku JSP ?**

Elementy skryptowe (1/2)

- **Skryptlety**
  - *<% kod %>* albo *<jsp:scriplet> kod </jsp:scriplet>*

```
<% if ((request.getParameter("imie") == "Marek") {%>
      <B> Mamy tak samo na imie !</B>
<%  } else { %>
      <B> Witaj <%= request.getParameter("imie") %> </B>
<%} %>
```

- **Deklaracje**
  - *<%! kod %>* albo *<jsp:declaration> kod </jsp:declaration>*

```
<%!
   public java.util.Date getDate() {
      return (new java.util.Date());
   }
%>
```

- ***Jak się zachowuje ta sama zmienna zdefiniowana w skryptlecie a jak w deklaracji?***

Elementy skryptowe (2/2)

- **Wyrażenia**
  - *<%= kod %>* albo *<jsp:expression> kod </jsp:expression>*

- **Komentarze**
  - **Składnia JSP:** <%-- komentarz --%>
  - **Składnia XML:** <!-- komentarz -->

- **Drugi komentarz trafia do wynikowego pliku HTML, zaś pierwszy przed zbudowaniem strony jest usuwany**
- **Do komentarzy można wstawiać elementy JSP (np. wyrażenie wstawione w komentarz będzie widoczne w źródle strony HTML - debugging)**
- **Niektóre implementacje motorów JSP nie obsługują poprawnie komentarzy JSP (usuwają znaczniki <% oraz %> resztę pozostawiając nienaruszoną)**

  <% //to jest komentarz w skryptlecie %>

Dyrektywy

- **Składnia**
  - <% @ nazwa_dyrektywy %>
  - < jsp:directive.nazwa_dyrektywy >
- **page**
  - <%@ page buffer="none|xxxkb" %>
  - <%@ page autoFlush="true|false" %>
  - <%@ page errorPage="errorpage.jsp"%>
  - <%@ page isErrorPage="true|false" %>
  - <%@ page import="java.util.*, cart.*" %>
  - <%@ page isThreadSafe="true|false" %>
  - <%@ page language="scripting language" %>
  - <%@ page session="true|false" %>
- **include**
  - <%@ include file="plik.jsp" %>
- **taglib**

Elementy akcji

- <jsp:nazwa_akcji atrybut=”wartość” />
- **Atrybuty: id i scope** (page, request, session, application)

- <jsp:useBean id=”newBean” class=”com.javadesktop.MyBean”>
  <jsp:setProperty name=”newBean” property=”val” value=”3”>
  </jsp:useBean>

- <jsp:include page=”/data/login.jsp” flush=”true” >
  <jsp:param name=”user” value=”Joe Rich” />
  </jsp:include>

- <jsp:forward page=”date.jsp”>

Tworzenie własnych znaczników (1/3)

Możliwości:

- **Wyłączenie kodu Javy z plików *.jsp**
- **Wielokrotne użycie tego samego kodu**
- **Udostępnianie kodu innym**

```
<body>
    <%@ taglib prefix="mt" uri="http://www.mimuw.edu.pl/mojetagi" %>
    <mt:if condition="true">
        <%= "Pierwszy "%> if.<br>
    </mt:if>

    <mt:if condition="false">
        <%= "Drugi "%> if.<br>
    </mt:if>

    <mt:if condition="<%= true %>">
        <%= "Trzeci "%> if.<br>
    </mt:if>
</body>
```

Tworzenie własnych znaczników (2/3)

- **Dekskryptor biblioteki znaczników** (TLD ang. tag library descriptor)

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <uri>http://www.mimuw.edu.pl/mojetagi</uri>

  <tag>
    <name>if</name>
    <tag-class>moje_tagi.IfTag</tag-class>
    <body-content>JSP</body-content>
    <description>Wypisuje zawartość w zalezności od warunku.</description>
    <attribute>
      <name>condition</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
</taglib>
```

Tworzenie własnych znaczników (3/3)

- **Deskryptor biblioteki znaczników**

```
import javax.servlet.jsp.tagext.*;

public class IfTag implements BodyTagSupport {
      private boolean condition = true;

      public void setCondition(boolean condition) {
            this.condition = condition;
      }

       public int doStartTag() throws JspException {
            if (condition)
              return EVAL_BODY_INCLUDE;
            else
              return SKIP_BODY;
      }

      public int doEndTag() throws JspException {
            return EVAL_PAGE;
      }

      public void release() {}
     //int doAfterBody() {}
}
```

Poprawianie wydajności (1/2)

- **Jak najwięcej pracy w metodzie init()**
- **Wyłączenie opcji auto-reloading (przeładowywanie serwletu co jakiś czas, przydatna jedynie przy tworzeniu programu)**
- **Kontrola HttpSession**
  - **Nie twórz nowej sesji w JSP jako default, jeśli nie jest konieczna**

    <%@ page session="false"%>
  - **Nie przechowuj dużych obiektów w HttpSession (koszt serializacji, za każdym razem serwer musi na nowo przetworzyć daną sesję)**
  - **Unieważniaj sesję gdy nie jest już potrzebna**

    public void **invalidate()**
  - **Ustaw time-out sesji (niska wartość)**

Poprawianie wydajności (2/2)

- **Nie używaj SingleThreadModel (nowy serwlet na każde żądanie, Servlet 2.4. - deprecated)**
- **Każde nowe żądanie tworzy nowy wątek (co jest kosztowne) lepiej jest więc utrzymywać pulę wątków**
- **Wybieraj prawidłową wartość zasięgu dla akcji** *useBean*

```
<jsp:useBean id="name" scope="page|request|session|application" class=
        "package.className">
</jsp:useBean>
```

- **Unikaj konkatenacji napisów** (StringBuffer zamiast tego)
- **ServletOutputStream vs. PrintWriter** (binaria, dane znakowe)

Kompresja

- **Standardowo treść stron WWW klientowi przez serwer przesyłanych jest w postaci pełnego tekstu opisującego daną stronę**
- **Nie każda przeglądarka ją obsługuje**

```
Enumeration e = ((HttpServletRequest)request).getHeaders("Accept-Encoding");
while (e.hasMoreElements())
{
    String header=(String)e.nextElement();
    if (header!=null && (header.toUpperCase().indexOf("GZIP")>-1))
    {
        setGZIPContent(response);
        //response.setHeader("Content-Encoding" , "gzip");
        out = new GZIPoutputStream(response.getOutputStream());
    }
    else
    {
        out= response.getOutputStream();
    }}
```

- **Znacznie szybsze rozwiązanie pomimo czasu koniecznego na kompresję danych i rozpakowywanie ich u klienta (dane tekstowe się bardzo dobrze kompresują)**

Podsumowanie

- **Serwlety są potężnym zastosowaniem Javy dla serwerów WWW**
- **Posiadają kilka unikalnych cech: (niezależność od platformy, łatwa implementacja, wydajność, wielowątkowość, kontrola sesji oraz specjalne klasy i metody dla protokołu HTTP). Stają się tym samym znacznie lepszym rozwiązaniem niż CGI**
- **Java Server Pages są mechanizmem o podobnych cechach jak serwlety jednak kod jest pisany w zupełnie inny sposób (jako strona HTML ze wstawionym kodem Javy)**
- **JSP definiuje wiele własnych znaczników ułatwiających pisanie kodu, można też tworzyć własne znaczniki**

Źródła

- Bruce Eckel "Thinking in Java", 2nd Edition
- D. Hougland, A. Tavistock "JSP – tworzenie stron WWW"

- http://www.servlet.com/srvpres/srprspgs/srprstoc.html
- http://java.sun.com/docs/books/tutorial/servlets/index.html
- http://www.javaworld.com/javaworld
- http://www.novocode.com/doc/servlet-essentials
- http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

KONIEC.