Aplety, servlety a jiná drůbež (4)

Minule jsme dokončili povídání o servletech. V tomto – již závěrečném – dílu našeho malého průvodce programováním pro web se ještě podíváme na stránky JSP a alespoň krátce si povíme o zvláštnostech ladění tohoto druhu programů.

Stránky JSP

Jak jste se snad přesvědčili minule, servlety jsou skvělá věc – ale také mají své problémy. Často je obsah HTML stránky z větší části „statický“, pevně daný, a pouze malé úseky se vytvářejí dynamicky, na základě požadavků. Pak tvoří převážnou část servletu příkazy println(), které vypisují stále stejný zdrojový kód HTML. Jistě si dovedete představit, že napsat takový servlet je úmorná práce.

A právě tady oceníme tzv. stránky JSP. Zkratka pochází z anglického JavaServer Pages, a znamená tedy něco jako „javské serverové stránky“.

Seznamte se...

Jako nejjednodušší příklad stránky JSP si vezmeme obyčejný HTML soubor, např. tento:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- Soubor JSP1.jsp -->
<HTML> <HEAD>
 <TITLE>První JSP stránka</TITLE>
</HEAD> <BODY>
<CENTER>
 <H1>Začínáme s JSP</H1>
 a tohle je naše první stránka.
</BODY> </HTML>
```

Soubor uložíme pod názvem JSP1.jsp do adresáře, kam se na serveru ukládají obyčejné stránky HTML (na serveru Tomcat 3.2 to je podadresář Webapps\Root v domovském adresáři serveru), a vyžádáme si jej stejným způsobem jako jakoukoli běžnou stránku HTML.

Typická stránka JSP obsahuje směs HTML textu a tzv. skriptovacích prvků. Veškerý text jazyka HTML přejde beze změn do výsledné HTML stránky. (Proto se o HTML textu ve stránkách JSP také hovoří jako o „šabloně textu“.) Pokud tedy stránka JSP obsahuje pouze HTML text, bude výsledkem HTML stránka naprosto identická se stránkou původní, jen odezva serveru bude pomalejší.

Skriptovací prvky, zvané též skriptlety, představují úseky javského programu „propašované“ do HTML textu. Přijde-li požadavek na stránku JSP, server tuto stránku nejprve přeloží do Javy – vytvoří servlet, který generuje odpovídající stránku HTML – a tento servlet spustí. Vytvořený servlet bude mimo jiné obsahovat metodu _jspService(), kterou bude volat nám už známá metoda service().

Skriptlety

Pod tímto podivným označením (v originále scriptlets; že by „skriptíky“?) se skrývají úseky javského textu uzavřené mezi znakové posloupnosti <% a %>. Dříve než přejdeme k jejich druhům, zastavíme se u komentářů v JSP.

Vložíme-li do stránky JSP komentář podle pravidel HTML, např.

```
<!-- Soubor JSP1.jsp -->
```

přejde tento komentář do vytvořené HTML stránky. Uživatel ho sice na obrazovce neuvidí, ale znaky, jimiž je tvořen, se budou po síti přenášet. Pokud chceme, aby se komentář do textu vytvořené HTML stránky nepřevzal (tzv. skrytý komentář, hidden comment), použijeme zápis

```
<%-- Soubor JSP1.jsp --%>
```

Jak plyne z definice, právě jsme použili skriptlet. Podobně jako zde znaky -- znamenaly, že jde o skrytý komentář, jsou jednotlivé druhy skriptletů (o nichž si povíme v následujících odstavcích) rozlišeny konkrétním znakem za úvodní dvojicí <%.

Výrazy

Výraz v JSP stránce je vlastně libovolný výraz vyhovující pravidlům Javy, zapsaný mezi znakové posloupnosti <%= a %>. Výraz se vyhodnotí a jeho hodnota se zapíše na odpovídající

místo do výsledné HTML stránky.

V takových výrazech můžeme používat předdefinované proměnné s těmito významy:

* request je proměnná typu HttpServletRequest představující požadavek;
* response je proměnná typu HttpServletResponse představující odezvu na požadavek;
* out je výstupní proud typu PrintWriter;
* session je instance typu HttpSession odpovídající aktuálnímu sezení;
* application obsahuje instanci typu ServletContext; k dispozici je však ještě několik dalších proměnných.

První dvě předdefinované proměnné, request a response, mají týž význam jako stejnojmenné parametry metod pro obsluhu požadavku v servletu.

Jako příklad použití výrazu napíšeme stránku JSP, která uživateli řekne, kdy jeho požadavek dorazil na server (podobným sloganem začínal v šedesátých letech jeden populární rozhlasový pořad). K tomu postačí vytvořit nový objekt třídy java.util.Date a použít ho jako výraz. Celá stránka může vypadat takto:

```
<%-- Soubor Cas.jsp --%>
<HTML> <HEAD>
<TITLE> Nepřesný čas </TITLE>
</HEAD> <BODY>
  <H1>Je právě tolik hodin, kolik právě je.</H1>
  <P>Váš požadavek dorazil na server
  <%= new java.util.Date() %> <P>
 Máte-li na svých hodinkách více nebo méně, není to naše vina.
</BODY> </HTML>
```

Výsledek ukazuje obrázek 1.

Stiskneme-li v prohlížeči tlačítko Obnovit, dostaneme nový (aktuálnější) časový údaj. (Další úpravy – např. formátu data – si můžete zkusit sami.)

Javský kód

Vložený úsek javského kódu signalizuje „samotná“ úvodní dvojice <%, tj. bezprostředně za ní následuje některý z „bílých znaků“ (mezera, nový řádek atd.); úsek, jak už víme, končí řetězcem %>. Tento kód bude vložen do metody _jspService(). I v javském kódu samozřejmě můžeme používat předdefinované proměnné, můžeme volat jejich metody apod.

Jako příklad napíšeme stránku JSP, která nám ukáže typ požadavku a všechna jeho záhlaví.

```
<%-- Soubor Pozadavek.jsp --%>
<HTML> <HEAD>
<TITLE> Podrobnosti o požadavku </TITLE>
</HEAD> <BODY>
 <H1>Server obdržel Váš požadavek.</H1>
 Typ požadavku: <%= request.getMethod() %>
 <P> <%
  java.util.Enumeration hn = request.getHeaderNames();
  while(hn.hasMoreElements())
  {
   String pom = hn.nextElement().toString();
   out.println(pom + ": " + request.getHeader(pom)+ "<BR>");
  } %>
</BODY> </HTML>
```

Zavoláme-li tuto stránku z prohlížeče pomocí URL http://localhost:8080/Pozadavek.jsp, objeví se obrázek 2.

Direktivy

Direktivy mají tvar

```
<%@ jméno_direktivy atribut="hodnota" %>
```

Atributů může být i více; pak se zápisy atribut="hodnota" zapisují za sebe (nebo spíše pod sebe) a oddělují se jen bílými znaky; hodnota musí být vždy uzavřena v uvozovkách. JSP používá direktivy page, include a taglib. První z nich umožňuje pracovat se strukturou stránky JSP, druhá slouží ke vkládání souborů do JSP stránek a třetí umožňuje používat uživatelem definované značky ve stránkách JSP. Podrobný rozbor by přesáhl možnosti našeho seriálu, a proto mi nezbývá než odkázat na dokumentaci [6] nebo na knihu M. Halla [1]. (V době, kdy budete tento článek číst, už by měl být na trhu český překlad.)

Zde si alespoň řekneme, že atribut import direktivy page umožňuje zpřístupnit ve stránce JSP

(tj. v jejích výrazech, javských kódech nebo deklaracích) třídu nebo celý balík, podobně jako příkaz import ve zdrojovém souboru v Javě.

V předchozím příkladu jsme použili rozhraní java.util.Enumeration. Vložíme-li jako první řádek direktivu

<%@ page import="java.util.*" %>

můžeme v deklaraci proměnné hn vynechat kvalifikaci jménem balíku:

```
<%
  Enumeration hn = request.getHeaderNames();
 ...
```

Deklarace

Deklarace v JSP začíná <%! a může obsahovat libovolnou deklaraci vyhovující syntaktickým pravidlům Javy. Tato deklarace se uloží do těla třídy servletu, který vznikne překladem JSP stránky. (Jde tedy o deklaraci datové složky nebo metody servletu. Jak jsme viděli v předchozím příkladu, může i javský kód obsahovat deklaraci proměnné, půjde ovšem o proměnnou lokální v metodě _jspService().)

Poznamenejme, že v deklaracích nelze používat předdefinované proměnné.

Jako příklad napíšeme stránku JSP, která bude počítat faktoriály nezáporných celých čísel. (Jistě si pamatujete, že faktoriál čísla n se značí n! a je to součin 1.2. ... . n. Není definován pro záporná čísla, navíc platí 0! = 1.) Nejprve vytvoříme formulář HTML, který bude obsahovat jedno vstupní pole pro zadání čísla a tlačítko pro jeho odeslání. I když jde o téměř doslovnou analogii formuláře z minulého dílu, pro pohodlí si ho zde zopakujeme:

```
<!-- Soubor Form2.html -->
<HTML> <HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1250">
<TITLE>Pokusný formulář</TITLE>
</HEAD> <BODY>
<CENTER>
 <H1>Spočti si svůj faktoriál</H1>
 <H2>S faktoriálem do nového tisíciletí</H2>
 </CENTER> <P>
 <FORM
 ACTION="http://localhost:8080/faktorial.jsp"
 METHOD=GET ACCEPT-CHARSET="windows-1250">
  <CENTER>
   Napište nezáporné celé číslo<BR>
   <INPUT TYPE="TEXT" NAME="cislo" VALUE= ""><P>
   <INPUT TYPE="RESET" VALUE="Smazat">
   <INPUT TYPE="SUBMIT" VALUE="Odeslat">
   </CENTER>
 </FORM> <HR>
</CENTER>
</BODY> </HTML>
```

Stránka JSP, která se o výpočet faktoriálu postará, by mohla vypadat např. takto:

```
<%-- Soubor Faktorial.jsp --%>
<HTML> <HEAD>
<TITLE> Výpočet faktoriálu </TITLE>
</HEAD> <BODY>
 Zadané číslo: <%= request.getParameter("cislo") %> <BR>
 Jeho faktoriál je: <%= f(request.getParameter("cislo"))%>
<%!
 // Funkce pro výpočet faktoriálu
 private String f (String co)
 {
  try {
   // Převeď znakový řetězec na číslo
   int n = Integer.parseInt(co);
   if(n < 0) return "Číslo musí být nezáporné";
   long s = 1; // s obsahuje výsledek
   while(n > 1) s *= n--; // Vlastní výpočet
   return String.valueOf(s);
```

```
 }
 catch(Exception e)
 {
  return "Špatně zadané číslo.";
 }
}
%>
</BODY> </HTML>
```

Výstup této stránky JSP ukazuje obrázek 3.

JSP a čeština

V minulém dílu jsme si řekli, že při přenosu českého textu z formuláře do servletu narazíme na problémy, a ukazovali jsme si, jak je lze řešit. Je to s podivem, ale stránky JSP podobnými neduhy netrpí. O tom se snadno přesvědčíme, jestliže v předchozím příkladu zadáme místo čísla opět onu známou větu o žluťoučkém koni. Výsledek ukazuje obrázek 4.

Podmíněný text

Někdy potřebujeme, aby se do výsledné stránky vložil různý text v závislosti na tom, zda je splněna nějaká podmínka. Trik, který to umožňuje, je založen na tom, že:

* jednotlivé řádky HTML textu („šablony textu“) se převádějí na příkazy výstupu a ukládají se do těla metody _jspService() v pořadí, v jakém jsou zapsány ve zdrojovém souboru;
* skriptlety se beze změny vkládají do zdrojového kódu metody _jspService() opět v pořadí, v jakém jsou zapsány v textu stránky JSP;
* pořadí skriptletů a šablon textu se zachovává.

Z toho plyne, že můžeme napsat např. takovouto konstrukci:

```
<% if(podm) { %>
<H1> Sláva! Podmínka je splněna.</H1>
<% } else { %>
<SMALL>Bohužel, všechno je jinak.</SMALL>
<% } %>
```

Podle toho, zda je splněna podmínka podm, se vloží do výstupu jeden nebo druhý nápis.

Další prvky

Na stránkách JSP můžeme vedle dosud uvedených prvků použít ještě další, které umožňují vložit aplet, použít komponentu JavaBeans atd. Většina těchto prvků používá syntaxi založenou na jazyce XML. To znamená, že začíná úvodní značkou <jsp:xxx>, pak následuje tělo prvku a koncová značka tvaru </jsp:xxx>. Značky, které nemají tělo, mají tvar <jsp:xxx/> (a neexistuje k nim koncová značka). Ve značce mohou být uvedeny parametry, například

</jsp:xxx width="25" height="100">; hodnota parametru musí být v uvozovkách.

Chceme-li třeba, aby vytvořená stránka JSP obsahovala aplet založený na JDK 1.2, který používají odpovídající plug-in, použijeme prvek <jsp:plugin>. V prvním dílu našeho seriálu jsme napsali aplet Aplet1, který vypisoval jakýsi mravoučný nápis. Budeme-li chtít vložit tento aplet do stránky JSP a použít přitom JDK 1.2, napíšeme

```
<jsp:plugin type="applet"
code="Aplet1.class"
width="300" height="150">
</jsp:plugin>
```

Do prvku <jsp:plugin> může být vnořen prvek <jsp:fallback> uvádějící alternativní text, který zobrazí prohlížeče, jež nepodporují Javu (nebo když se nepodaří nainstalovat plug-in).

Dále může být do prvku <jsp:plugin> vnořen prvek <jsp:params>, který uvádí specifikaci parametrů apletu. Předchozí text bychom tedy mohli upravit např. takto:

```
<jsp:plugin type=...>
 <jsp:params>
   <jsp:param name="text"
value="Chaos vládne i bez ministrů."/>
   <jsp:param name="autor"
value="Bobbyho přesvědčení"/>
 </jsp:params>
 <jsp:fallback>
```

```
        Je mi líto, ale Váš prohlížeč nezvládá Javu.
       </jsp:fallback>
      </jsp:plugin>
```

Vytvořený zdrojový text HTML bude vypadat takto:

```
      <OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="300"
height="150"  codebase="http://java.sun.com/products/plugin/1.2.2/
                                        jinstall-1_2_2-win.cab#Version=1,2,2,0">
      <PARAM name="java_code" value="Aplet1.class">
      <PARAM name="type" value="application/x-java-applet;">
      <PARAM name="autor" value="Bobbyho přesvědčení">
      <PARAM name="text" value="Chaos vládne i bez ministrů.">
      <COMMENT>
      <EMBED type="application/x-java-applet;"  width="300"  height="150"
pluginspage="http://java.sun.com/products/plugin/"
                           java_code="Aplet1.class" autor=Bobbyho přesvědčení
      text=Chaos vládne i bez ministrů.>
      <NOEMBED>
      </COMMENT>
      Je mi líto, ale Váš prohlížeč nezvládá Javu.
      </NOEMBED></EMBED>
      </OBJECT>
```

Ladění servletů a JSP

Při ladění aplikací založených na servletech jde o několik různých problémů. Prvním z nich je odladění vlastního algoritmu pro zpracování požadavku, druhým pak je odladění vytvářené HTML stránky. Další problémy se mohou týkat síťového spojení, jeho rychlosti atd.

Poznamenejme, že profesionální vývojové nástroje obvykle nabízejí možnost spustit servlet přímo ve vývojovém prostředí, zobrazit si jeho výstup bez použití prohlížeče atd. V dalším textu ale nepředpokládám žádný konkrétní nástroj.

Webový server

Při vývoji servletů je prakticky nezbytné mít na počítači instalovaný některý z webových serverů nebo jejich náhražek umožňujících spouštět servlety a JSP. Spouštíme-li servlety na takovémto lokálním serveru, máme k dispozici javskou konzolu, na kterou si můžeme vypisovat ladicí informace.

O serveru Apache Tomcat (apačský kocour) od Apache Software Foundation jsme již hovořili ve druhém dílu tohoto seriálu; doplňme ještě, že ho lze používat jako samostatný server pro ladění nebo jako doplněk do webového serveru Apache, který mu umožňuje spouštět servlety a JSP.

Vedle toho lze pro ladění použít JavaServer Web Development Kit (JSWDK) od firmy Sun. O něm lze získat podrobnější informace na adrese [8]. Jde o stroj pro spouštění servletů podle specifikace 2.1 a JSP podle specifikace 1.0. JSWDK však již není dále vyvíjen a firma Sun nabízí na svých stránkách ke stažení Tomcat.

Další možnost představuje LiteWebServer od Gefion Software, který si lze stáhnout z [9]. V současné době aktuální verze 2.2.2 implementuje servlety verze 2.2 a JSP verze 1.1.

Adresář pro laděné servlety

Některé servery mají zvláštní adresář pro servlety, jejichž třídy se často mění (jinými slovy pro laděné servlety). Změní-li se soubor .class hlavní třídy servletu v tomto adresáři, server ji automaticky znovu zavede do paměti.

Server Apache Tomcat tuto vlastnost bohužel nemá. To znamená, že při každé změně třídy servletu musíme server zastavit a znovu spustit. Pokud na to zapomeneme, opravená třída servletu se do paměti nezavede a poběží stále neopravená verze. (Totéž platí i o změně stránky JSP.)

Prohlížeč

Další problém může představovat webový prohlížeč, přesněji Internet Explorer nejnovějších verzí. V případě, že se nepodaří vyhovět požadavku – např. proto, že požadovaná stránka nebyla nalezena, že byla zrušena, přesunuta atd. –, obsahuje odezva serveru HTML stránku s informacemi o vzniklém problému. MS IE ovšem tuto stránku nahradí svou vlastní (viz obr. 5), která je sice příjemnější pro nezasvěceného uživatele, ale pro programátora, který ladí webovou aplikaci, je naprosto nevhodná, neboť neříká, co se vlastně stalo.

Naštěstí lze u IE nastavit standardní chování. V jeho nabídce Nástroje vybereme možnost

Možnosti sítě Internet..., přejdeme na kartu Upřesnit a vyhledáme skupinu Procházení (je poměrně rozsáhlá). Zde zrušíme zaškrtnutí položky Zobrazovat jednoduché chybové zprávy protokolu HTTP (obr. 6). (Přesný název této položky se v některých verzích IE liší.) Poté již bude prohlížeč zobrazovat chybová hlášení serveru, a to včetně hlášení o neošetřených výjimkách v servletu nebo JSP.

Připomeňme si, že po úpravě HTML stránky nebo servletu je třeba zadat příkaz Obnovit (příp. Aktualizovat – může se jmenovat i jinak), který způsobí znovuzavedení HTML stránky do paměti nebo nové volání servletu.

HTML

Jiným problémem je správnost vytvořeného HTML dokumentu. Abychom si mohli prohlédnout zdrojový kód vytvořeného HTML dokumentu, stačí v IE v nabídce Zobrazit použít příkaz Zdrojový kód. (Podobnou možnost nabízí většina prohlížečů.) Vedle toho lze použít i speciální programy pro ověření správnosti HTML dokumentů, tzv. validátory.

O čem jsme nehovořili

Do čtyřdílného seriálu se pochopitelně nevejde vše, co bychom mohli o programování pro internet napsat. S výjimkou apletů jsme zcela pominuli programování na straně klienta (např. skriptování v HTML stránkách pomocí jazyků, jako je JavaScript nebo VBScript); totéž platí o používání databází v servletech a JSP. (Základní informace o JDBC najdete např. v dodatku k [1].)

Vůbec jsme se nezmínili o vytváření distribuovaných aplikací založených na standardu CORBA nebo na RMI. (Základní poučení lze najít v knize [2].) Nedostalo se ani na přesměrování požadavků na jiné servlety nebo JSP stránky nebo na přímou komunikaci servletu a apletu („tunelování“ HTTP).

Nehovořili jsme o vytváření knihoven uživatelských značek JSP, o používání komponent JavaBeans v JSP stránkách a o mnoha dalších tématech.

V případě apletů jsem si dovolil předpokládat, že je znáte a umíte s nimi zacházet. Základní informace o nich najdete prakticky v každé učebnici Javy, dokonce i v mé učebnici [3] určené naprostým začátečníkům. Podrobnější informace na toto téma získáte např. v [2] nebo ve [4].

O CGI skriptech založených na jazyku C lze najít informace např. v knize [5]. Další informace o servletech a JSP najdete v on-line dokumentaci [6]. Také výklad o protokolu HTTP byl velice povrchní. Podrobnější informace lze najít v [7].

Miroslav Virius

Odkazy

[1] M. Hall: Core Servlets and JavaServer Pages. Prentice Hall 2000
[2] B. Eckel: Myslíme v jazyku Java (1. a 2. díl). Grada Publishing, Praha 2001
[3] M. Virius: Java pro zelenáče. Neokortex, Praha 2001
[4] P. Herout: Java – grafické uživatelské prostředí a čeština. Kopp, České Budějovice 2001
[5] R. Stones, N. Matthew: Linux – začínáme programovat. Computer Press 2000
[6] http://java.sun.com/j2ee/j2sdkee/techdocs/api/
[7] http://www.ietf.org/rfc/rfc2068.txt
[8] http://java.sun.com/products/servlet/archive.html
[9] http://www.gefionsoftware.com/LiteWebServer/