Príručka Java

Prehľad Java EE a Servlety

IT Academy s.r.o., Tomášikova 50/A, 831 04 Bratislava, tel.: 0917/095 406, 0907/375 543
IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

Obsah

Túto príručku môžete využiť ako pomôcku pri práci s programom Java. **Príručka podlieha autorským právam a jej vlastníkom je spoločnosť IT Academy s.r.o.**

I. Servlety

Servletmi sa myslia krátke programy **spustené** na serverovej strane webového pripojenia. Podobne ako **aplety** dynamicky rozširujú funkcionalitu webového prehliadača.

Pokiaľ chcete plne porozumieť všetkým výhodám servletov, musíte mať aspoň základné znalosti o spôsobe, akým **spolupracuje** webový server s **prehliadačom** pri sprístupňovaní obsahu používateľovi.

Životný cyklus servletu

Z hľadiska životné cyklu sú zásadné **tri metódy**, ktorými sú **init(), service(), destroy().** Tie sú implementované každým servletom a sú **serverom** volané v určitých časových okamihoch. Povedzme si nejaký typický príklad, na ktorom by sme si mohli ukázať, kedy sú zmienené metódy **volané**.

Predpokladajme, že používateľ zadal URL do **adresného** riadku **prehliadača**. Ten vzápätí pre túto adresu URL vygeneroval HTTP **požiadavku**. Táto požiadavka bola následne odoslaná na príslušný server. V druhom kroku bola HTTP požiadavka prijatá **webovým** serverom.

Server namapoval túto požiadavku na príslušný **servlet**, ktorý bol potom dynamicky načítaný do adresného **priestoru** webového serveru. Po tretie server zavolal metódu **init()** načítaného servletu. Pritom platí, že táto metóda je volaná iba v prípade prvého načítania servletu do **pamäti**.

Súčasne je možné predávať **inicializačné** parametre, čo znamená, že servlet je schopný **konfigurovať** sám seba. V štvrtom kroku server zavolal metódu **service()** načítaného servletu. Táto metóda je volaná s cieľom spracovania HTTP požiadavky.

Servlet je schopný čítať dáta, ktoré boli zaslané v HTTP požiadavke. Súčasne je schopný vytvoriť aj **http odpoveď**. Súčasne je servlet **schopný** vytvoriť aj HTTP odpoveď pre klienta. Servlet zostáva v adresnom priestore webového serveru a je schopný **spracovať** akúkoľvek ďalšiu požiadavku prijatý od klientov.

Ako sme si už povedali, pre spracovanie každej HTTP požiadavky je volaná metóda **service()**.

A nakoniec sa server môže rozhodnúť odstrániť **servlet** zo svojej pamäti. Pritom algoritmy, na ktorých základe sa server takto rozhodne, sú špecifické pre každý webový server. V tomto prípade server zavolá metódu **destroy()** servletu, v dôsledku čoho dôjde k **uvoľneniu** všetkých **prostriedkov** alokovaných servletom.

Dôležité dáta môžu byť uložené na trvalom **úložisku**. Následne môže proces **garbage collection** uvoľniť aj celú pamäť využívanú **servletom** a jeho objekty.

II. Možnosti pre vývoj servletov

Pri vytváraní servletov budete potrebovať prístup ku kontajneru servletov či serverov. Medzi často používané patria napríklad servery **Glassfish** a **Tomcat**. Server **Glassfish** je poskytovaný spoločnosťou Oracle a je súčasťou SDK Java EE. Tento server je tiež podporovaný vývojovým prostredím **NetBeans**.

Server **Tomcat** je **open-source** serverom podporovaným **Apache Software Foundation**. Aj tento server je možné využiť spolu s vývojovým prostredím NetBeans. Súčasne ale platí, že ako server **Glassfish**, tak aj server **Tomcat** môžu byť využívané aj s inými vývojovými prostrediami, napríklad s **Eclipse**.

Aj napriek tomu, že vývojové prostredia typu **NetBeans** či **Eclipse** sú veľmi užitočné a môžu vývoj servletov zjednodušiť a urýchliť, nebudeme s nimi pracovať. Platí totiž, že spôsob **nasadenia** a vývoja servletov sa v jednotlivých vývojových prostrediach **líšia** a z tohto dôvodu nie je možné popísať všetky **prípadné** prostredia.

III. Používanie serveru Tomcat

Súčasťou serveru **Tomcat** sú všetky knižnice tried, dokumentácie, ktoré budete potrebovať pre vývoj a testovanie servletov. Príklad, ktorý si popíšeme predpokladá **prácu** v prostredí **32-bitového** systému **Windows**.

Ak vychádzame z predpokladu, že ste si stiahli 32-bitovú verziu, uložili ju do **koreňa** pevného disku a tu ste ju aj rozbalili, potom **Tomcat** bude štandardne umiestnený v ceste: C:\apache-tomcat-7.0.47-windows-x86\apache-tomcat-7.0.47

Toto je tiež umiestnenie, z ktorého vychádzajú všetky príklady, ktoré budeme uvádzať. Pokiaľ ste si Tomcat uložili do inej cesty, budete musieť všetky príklady zodpovedajúcim spôsobom upraviť. Je tiež možné, že budete musieť nastaviť premennú prostredia JAVA_HOME tak, aby obsahovala tú cestu, do ktorej bola inštalovaná sada **Java Development Kit.**

Pri nastavovaní tejto **premennej** majte na mysli, že **nesmie** obsahovať medzeru. Po dokončení inštalácie môžete server **Tomcat** spustiť pomocou súboru **startup.bat**, nachádzajúceho sa v podzložke bin zložky apache-tomcat-7.0.47. K ukončeniu chodu serveru Tomcat spustite **shutdown.bat**, nachádzajúci sa v tej istej zložke.

Triedy a rozhrania potrebné pre vývoj servletov sa nachádzajú v balíčku **servlet-api.jar**, umiestnenom v tomto **adresári**:

C:\apache-tomcat-7.0.47-windows-x86\apache-tomcat-7.0.47\lib

Pokiaľ chcete tento balíček **sprístupniť** aj pre vývoj, musíte aktualizovať premennú prostredia **CLASSPATH** tak, aby obsahovala nasledujúci **údaj**:

C:\apache-tomcat-7.0.47-windows-x86\apache-tomcat-7.2.47\lib\servlet-api.jar

Prípadne môžete tento súbor zadať pri kompilovaní servletov pomocou tohto príkazu:

javac HelloServlet.java-classpath "C:\apache-tomcat-7.0.47-windows-x86\
apache-tomcat-7.0.47\lib\servlet-api.jar"

Hneď ako **servlet** skompilujete, musíte serveru **Tomcat** umožniť jeho nájdenie. V našom prípade to znamená uložiť servlety do adresára nachádzajúceho sa v adresári **webappsv** tej ceste, do ktorej bol inštalovaný Tomcat a zadanie jeho názvu do súboru **web.xml**. V rámci snahy o zjednodušenie celého **procesu** budú všetky príklady využívať adresár **web.xml.**

Ten je súčasťou inštalácie **serveru** Tomcat a obsahuje jeho vlastné ukážkové servlety. Vďaka tomu nebude musieť vytvárať žiadne súbory ani **adresáre** iba preto, aby sme mohli experimentovať s **ukážkovými** servletmi.

Najprv skompilujeme súbor triedy **servletov** do nasledujúceho adresára:

C:\apache-tomcat-7.0.47-windows-x86\apache-tomcat-7.0.47\webapps\
examples\WEB-INF\classes

Následne zadajte názov servletu a **príslušné** mapovanie do súboru web.xml nachádzajúceho sa v tejto ceste:

C:\apache-tomcat-7.0.47-windows-x86\apache-tomcat-7.0.47\webapps\
examples\WEB-INF

Predpokladajme, že náš prvý príklad sa bude volať **HelloServlet**, potom musíme do sekcie definujúcej servlety pridať nasledujúce **riadky**:

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>HelloServlet</servlet-class>
</servlet>
```

Potom sa musíte **presunúť** ešte do sekcie popisujúcej **mapovanie** servletov a pridať do nej tieto riadky:

```
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url/pattern>/servlets/servlet/servlet/HelloServl
  et</url-pattern>
</servlet-mapping>
```

Tento všeobecný postup môžete **použiť** pri práci so všetkými našimi príkladmi.

IV. Jednoduchý servlet

Aby ste sa zoznámili so základnými **konceptmi** servletov, pripravili sme jednoduchý servlet, ktorý si môžete vyskúšať. Základné kroky sú **nasledujúce**:

1. **Vytvorte** a skompilujte zdrojový kód servletu. Potom **nakopírujte** súbor triedy servletu do príslušnej zložky a názov **servletov** spolu s potrebnými mapovaniami zadajte do **príslušného** súboru web.xml.
2. **Spusťte Tomcat.**
3. **Spusťte** webový **prehliadač** a **skúste** načítať servlet.

Vytvorenie a kompilácia zdrojového kódu servletu

Pre začiatok si vytvoríme pomerne **jednoduchý** súbor s názvom **HelloServlet.java** obsahujúci tento zdrojový **kód**:

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

public class HelloServlet extends GenericServlet; {
  public void service(ServletRequest request,
    ServletResponse response)

    throws ServletException, IOException {
     response.setContentType(“text/html”);
     PrintWriter pw=response.getWriter();
     pw.println(“<b>Ahoj!</b>”);
     pw.close();
  }
}
```

Program **importuje** balíček javax.servlet. Tento balíček obsahuje triedy a **rozhranie** potrebné pre tvorbu servletov. Následne program definuje triedu HelloServlet ako **podtriedu** triedy GenericServlet. Súčasťou triedy GenericServlet sú funkcie zjednodušujúce **vytváranie** servletov.

Nájdete v nej napríklad aj také verzie metód **init()** a **destroy(),** ktoré môžete použiť také aké sú – bez potreby akýchkoľvek zmien. Sami teda budete musieť napísať iba kód metódy **service()**. Vo vnútri triedy **HelloServlet** nájdete prekrytú metódu **service()**. Všimnite si, že prvým argumentom tejto metódy je objekt typu **ServletRequest**.

Tento objekt umožňuje servletu načítanie dát prijatých v klientskej požiadavke. Druhým argumentom metódy **service()** je potom objekt typu **ServletResponse**. Vďaka nemu je servlet schopný vytvoriť odpoveď pre klienta.

Ďalej je volaná metóda **getWriter()** načítavajúca objekt typu **PrintWriter**. Všetko, čo je zapísané do tohto prúdu, je klientovi odoslané ako súčasť HTTP odpovede. K zapísaniu nejakého jednoduchého kódu HTML tvoriaceho HTTP odpoveď sme potom použili metódu **println().**

Skompilujte tento zdrojový kód a umiestnite súbor **HelloServlet.class** do tej zložky serveru **Tomcat**, o ktorom sme sa zmienili v predchádzajúcej časti. Okrem toho súbor **HelloServlet** pridajte do súboru **web.xml** a to spôsobom, ktorý je popísaný v predchádzajúcej časti.

V. Spustenie serveru Tomcat

Server Tomcat spustíte spôsobom popísaným vyššie. Platí, že server musí byť **spustený** skôr ako sa pokúsite **spustiť** servlet.

Spustenie webového prehliadača a odoslanie požiadavky na načítanie servletu

Spustite webový **prehliadač** a do jeho adresného riadku zadajte nasledujúcu **adresu**:

http://localhost:8080/examples/servlets/servlet/HelloServlet

Alternatívne môžete použiť aj túto **adresu**:

http://127.0.0.1:8080/examples/servlets/servlet/HelloServlet

Pokiaľ ste spravili všetko **správne**, vo webovom prehliadači uvidíte výstup **servletu**.

API servletu

Pre vytváranie **servletov** sú potrebné dva **balíčky** obsahujúce všetky potrebné triedy a rozhrania. Týmito **balíkmi** sú **javax.servlet** a **javax.servlet.http**. Môžeme povedať, že tieto dva balíky vytvárajú aplikačné programovacie rozhranie servletu. Tieto balíčky nie sú súčasťou sady **základných** balíkov Java.

To znamená, že ich **nenájdeme** priamo v inštalácií Javy SE. Namiesto toho sú tieto balíky dodávané spolu so serverom **Tomcat**, okrem toho ich nájdeme aj v inštalácií Java EE. API servletov prechádza stále obdobím **vývoja**.

VI. Balík javax.servlet

Tento balík obsahuje množstvo rozhraní a tried vytvárajúcich základný **framework**, v ktorom je servlet vykonávaný. Prehľad základných **rozhraní**, ktoré sú súčasťou popisovaného balíku nájdeme v nasledujúcej tabuľke. Je možné povedať, že **pravdepodobne** najdôležitejším z nich je rozhranie **Servlet**.

Platí, že všetky **servlety** musia buď implementovať toto rozhranie priamo alebo musia rozlišovať nejakú triedu, ktorá toto rozhranie **implementuje**. Ďalším dôležitým rozhraním sú rozhrania **ServletRequest** a **ServletResponse**.

Rozhranie	Popis
Servlet	Deklaruje metódy životného cyklu servletu.
ServletConfig	Umožňuje servletom načítanie inicializačných parametrov.
ServletContext	Umožňuje servletom protokolovanie udalostí a pristupovanie k informáciám ohľadom ich prostredia.
ServletRequest	Používa sa k načítaniu dát z klientskych požiadaviek.
ServletResponse	Používa sa k zapisovaniu dát do odpovedí pre klientov.

Tab. 1 Prehľad rozhraní obsiahnutých v balíčku javax.servlet

Ďalšia tabuľka potom **obsahuje** prehľad všetkých základných tried, ktoré je súčasťou balíka **javax.servlet**.

Trieda	Popis
GenericServlet	Implementuje metódy životného cyklu servletu.
ServletInputStream	Umožňuje servletom načítanie inicializačných parametrov.
ServletOutputStream	Umožňuje servletom protokolovanie udalostí a pristupovanie k informáciám týkajúcich sa ich prostredia.
ServletException	Naznačuje, že pri behu servletu došlo k nejakej chybe.
UnvaliableException	Naznačuje, že daný servlet nie je dostupný.

Tab. 2 Prehľad tried obsiahnutých v balíku javax.servlet

VII. Rozhranie servlet

Ako sme si povedali, všetky servlety musia **implementovať** rozhranie Servlet. Toto rozhranie deklaruje metódy init(), service(), destroy() volané **servletom** v priebehu životného cyklu servletu. Okrem nich toto rozhranie obsahuje aj metódu umožňujúcu servletu načítanie všetkých **inicializačných** parametrov.

Vývojár servletov obvykle prekrýva metódu **getServletConfig()** preto, aby pomocou nej mohol zadať reťazec obsahujúci užitočné informácie **týkajúce** sa servletu. Aj táto metóda je volaná **servletom**.

Metóda	Popis
void destroy()	Táto metóda je volaná vo chvíli uvoľnenia servletu z pamäti.
ServletConfig getServletConfig	Vracia objekt typu ServletConfig obsahujúci všetky inicializačné parametre.
String getServletInfo()	Vracia reťazec popisujúci servlet.
void init(ServletConfig sc) throws ServletException	Táto metóda sa volá behom inicializácie servletu.
void service(ServletRequest poziadavka, ServletResponse odpoved) throws SetvletException, IOException	Táto metóda sa volá s cieľom spracovania požiadaviek od klienta. Pritom táto požiadavka je predávaná pomocou argumentu **poziadavka**. Argument **odpoved** potom určuje objekt odpovede, ktorá bude odoslaná späť klientovi.

Tab. 3 Metódy definované rozhraním Servlet

Rozhranie ServletConfig

Rozhranie **ServletConfig** umožňuje servletu načítanie konfiguračných dát v dobe **načítania** servletu. Metódy deklarované týmto **rozhraním** sú stručne popísané nižšie.

Metóda	Popis
ServletContext getServletContext()	Vracia kontext volajúceho servletu.
String getInitParameter(String parameter)	Vracia hodnotu inicializačného parametru majúceho názov parameter.
Enumeration<String> getInitParameterNames()	Vracia vymenovávané názvy všetkých inicializačných parametrov.
String getServletName	Vracia názov volajúceho servletu.

Tab. 4 Metódy definované rozhraním ServletConfig

Rozhranie ServletContext

Rozhranie **ServletContext** umožňuje **servletom** načítanie informácií o ich prostredí.

Rozhranie ServletRequest

Rozhranie **ServletRequest** umožňuje **servletom** načítanie informácií týkajúcich sa klientskych **požiadaviek**.

Rozhranie ServletResponse

Rozhranie **ServletResponse** umožňuje servletom vytváranie odpovedí pre klientov.

Metóda	Popis
Object GetAttribute(String atribut)	Vracia hodnotu atribútu serveru majúceho názov **atribut**.
String getMimeType(String subor)	Vracia MIME typ súboru daného argumentu **subor**.
String getRealPath(String vCesta)	Vracia skutočnú cestu zodpovedajúcu virtuálnej cene danej argumentom **vCesta**.
String getServerInfo()	Vracia informácie o serveri.
void log(String retazec)	Zapisuje reťazec do protokolu servletu.
void log(String retazec, Throwable výnimka)	Zapisuje reťazec a stack trace výnimky danej argumentom **vynimka** do protokolu servletu.
void setAttribute(String atribut, Object hodnota)	Atribút danému argumentu **atribut** priraďuje hodnotu danú argumentom **hodnota**.

Tab. 5 Niektoré z metód definovaných rozhraním ServletContext

Metóda	Popis
Object getAttribute(String atribut)	Vracia hodnotu atribútu, ktorého názov zodpovedá argument **atribut**.
String getCharacterEncoding()	Vracia kódovanie znakov klientskej požiadavky.
int getContentLength()	Vracia veľkosť požiadaviek. Hodnota -1 je vrátená v prípade, že veľkosť nie je možné zistiť.
String getContentType()	Vracia typ požiadaviek. Hodnota null je vracaná v prípade, že veľkosť nie je možné stanoviť.
ServletInputStream getInputStream() throws IOException	Vracia objekt typu ServletInputStream, ktorý je možné využiť k čítaniu binárnych dát z požiadaviek. Výnimka typu IllegalStateException je vyvolaná vtedy, pokiaľ metóda getReader() volajúcej požiadavky bola už volaná.
String getParameterString(String nazovParametra)	Vracia hodnotu parametrov, ktorých názov odpovedá argumentu **nazovParametru**.
Enumeration<String> getParameterNames()	Vracia vymenovávanie obsahujúce názvy parametrov volajúcej požiadavky.
String[] getParentValues(String nazov)	Vracia pole obsahujúceho hodnoty, ktoré súvisia s parametrom, ktorého názov zodpovedá argumentu **nazov**.
String getProtocol()	Vracia popis protokolu.
BufferReader getReader() throws IOException	Vracia objekt typu BufferReader ktorý je možné využiť k čítaniu textu z požiadaviek.

String getRemoteAddr()	Vracia reťazec zodpovedajúci IP adrese klienta.
String getRemoteHost()	Vracia reťazec zodpovedajúci názvu počítača klienta v systéme DNS.
String getScheme	Vracia to prenosové schéma URL adresy, ktoré bolo použité v požiadavke.
String getServerName	Vracia názov servletu.
int getServerPort	Vracia číslo portu.

Tab. 6 Niektoré z metód definovaných rozhraním ServletRequest

Metóda	Popis
String getCharacterEncoding()	Vracia kódovanie znaku odpovede pre klienta.
ServletOutputStream getOutputStream() throws IOException	Vracia objekt typu ServletOutputStream, ktorý je možné využiť k zápisu binárnych dát do odpovede.
PrintWriter getWriter() throws IOException	Vracia objekt typu PrintWriter, ktorý je možné využiť k zápisu znakov do odpovede.
void setContentLength(int velkost)	Nastavuje dĺžku obsahu odpovede na veľkosť.
void setContentType(String typ)	Nastavuje typ obsahu odpovede na typ.

Tab. 7 Niektoré z metód definované rozhraním ServletResponse

Trieda GenericServlet

Táto trieda zaisťuje implementáciu **základných** metód životného cyklu servletu. **Implementuje** ako rozhranie Servlet, tak aj rozhranie **ServletConfig**. Okrem toho je súčasťou tejto triedy aj metóda **umožňujúca** prepojenie reťazca k súboru protokolu servletu. **Signatúry** tejto metódy vyzerajú nasledovne:

```
void log(String retazec)
void log(String retazec, Throwable vynimka)
```

Tu argument reťazec predstavuje **retazec**, ktorý má byť pripojený k protokolu a argument **vynimka** popisuje výnimku, ktorá sa **vyskytla**.

Trieda ServletInputStream

Trieda **ServletInputStream** rozširuje triedu **InputStream**. Táto trieda je implementovaná kontajnerom servletov a vytvára **vstupný** prúd, ktorý môže vývojár servletu využiť k **čítaniu** dát z klientskej požiadavky.

Súčasťou popisovanej triedy je **konštruktor**. Okrem neho obsahuje táto trieda aj metódu, ktorá slúži k čítaniu bajtov z prúdu. Zmienená metóda je **definovaná nasledovne**:

```
int readLine(byte[] buffer, int pozicia, int
velkost)
throws IOException
```

Argument **buffer** predstavuje pole, do ktorého budú načítané bajty o súhrnnej veľkosti **veľkosť.** Tieto **bajty** budú do poľa ukladané od pozície danej argumentom **pozicia**. Metóda vracia skutočný počet načítaných bajtov, prípadne hodnotu -1, pokiaľ behom čítania **narazí** na **koniec** prúdu.

Trieda ServletOutputStream

Táto trieda rozširuje triedu **OutputStream**. Je implementovaná kontajnerom servletov a vytvára vstupný prúd, ktorý môže vývojár servletu využiť k zápisu dát do **odpovede** triedy **ServletException**. Pokiaľ je vyvolaná táto výnimka, znamená to, že servlet nie je **dostupný**.

Trieda výnimiek servletu

Súčasťou balíka **javax.servlet** sú aj triedy dvoch výnimiek. Prvá z nich je **SErvletException**, ktorá hovorí, že počas chodu servletu sa vyskytol nejaký **problém**. Druhá je potom trieda **UnvaliableException**, rozširujúca triedu **SErvletException**. Pokiaľ je vyvolaná táto **výnimka**, znamená to, že servlet nie je dostupný.

VIII. Čítanie parametrov servletu

Súčasťou rozhrania **ServletRequest** sú metódy umožňujúce **čítanie** názvov a hodnôt parametrov obsiahnutých v klientskej požiadavke. Ukážeme si príklad servletu, v ktorom budú tieto metódy **využité**.

Pritom celý príklad je tvorený **dvoma** súbormi. Samotná webová stránka je definovaná **zdrojovými** kódmi uloženými v súbore PostParameters.html, zatiaľ čo kód servletu **nájdete** v súbore **PostParameterServlet.java**.

Nasledujúci výpis vám **ukazuje** zdrojový kód súboru **PostParameters.html**. Ako vidíte, v kóde je definovaná **tabuľka** obsahujúca dva popisy a dve textové polia. Jeden z týchto popisov je **Zamestnanec** a druhý je **Linka**. Okrem toho je súčasťou tejto stránky aj tlačidlo pre **odoslanie**.

```
<html>
  <body>
   <center>
       <form name="Form1" method="post"
       action="http://localhost:8080/D1-02-
       CitanieParametrov/PostParametreServlet">
        <table>
           <tr>
               <td><b>Zamestnanec</b></td>
           <td><input type="text" name="e"
           size="25" value="" /></td>
```

```
            </tr>
            <tr>
                <td><b>Oddelenie</td>
                <td><input type="text" name="p"
            size="25" value="" /></td>
            </tr>
          </table>
          <input type="submit" value="Odoslať" />
        </form>
    </center>
  </body>
</html>
```

Dostávame sa k zdrojovému kódu servletu **PostParametersServlet.java** uvedenom nižšie. Metóda **service()** je prekrytá tak, aby umožnila spracovávanie klientskych požiadaviek. Metóda **getParameterNames** vracia vymenovávanie názvov **parametrov**. Tie sú následne spracované v slučke. Pritom **hodnota** parametrov je načítaná metódou **getParameter()**.

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

public class PostParameterServlet extends
GenericServlet {

public void service(ServletRequest request,
  ServletResponse response)

throws ServletException, IOException {
  req.setCharacterEncoding("utf-8");
```

```
  // Vytvorenie projektu typu PrintWriter.
  PrintWriter pw=response.getWriter();

  // Vytvorenie vymenovávania obsahujúceho názvy
  parametru.
  Enumeration e=request.getParameterNames();

  // Zobrazenie názvu a zodpovedajúcich hodnôt.
  while(e.hasMoreElements()) {
    String pname=(String)e.nextElement();
    pw.print(pname + "=");
    String pvalue=request.getParameter(pname);
    pw.println(pvalue);
  }
  pw.close();
 }
}
```

Skompilujte servlet. Vzniknutý súbor **skopírujte** do príslušného adresára a súbor web.xml zaktualizujte spôsobom popísaným vyššie. Potom si príklad **vyskúšajte**, pričom postupujte nasledovne:

1. **Spustite** server **Tomcat**.
2. Zobrazte si **webovú** stránku **PostParameters.html** v prehliadači.
3. Do jednotlivých polí stránky **zadajte** meno **zamestnanca** a jeho linku.
4. **Odošlite webovú** stránku.

Pokiaľ vykonáme **všetky** kroky správne, prehliadač vám zobrazí odpoveď, dynamicky vygenerovanú **servletom**.

Balík javax.servlet.http

V predchádzajúcich príkladoch sme využívali triedy a rozhrania definované v balíku javax.servlet, napríklad ServletRequest, ServletResponse a GenericServlet. Pomocou nich sme si ukázali základné **funkcionality** servletov. Pokiaľ pracujete s protokolom HTTP, budete obvykle využívať tie triedy a rozhrania, ktoré sa nachádzajú v balíku **javax.servlet.html**.

Rozhranie	Popis
HttpServletRequest	Umožňuje načítanie dát z HTTP požiadaviek.
HttpServletResponse	Umožňuje servletom zapisovanie dát do HTTP odpovedí.
HttpSession	Umožňuje čítanie a zapisovanie dát relácie.
HttpSessionBindingListener	Informuje objekt, že je viazaný k relácií alebo že je jeho väzba k relácií uvoľnená.

Tab. 8 Prehľad rozhraní obsiahnutých v balíku javax.servlet.http

Tried	Popis
Cookie	Umožňuje ukladanie stavových informácií na klientskom počítači.
HttpServlet	Obsahuje metódy pre spracovanie HTTP požiadaviek a odpovedí.
HttpSessionEvent	Zapúzdruje udalosť, ktorá hovorí, že došlo k zmene relácie.
HttpsessionBindingEvent	Naznačuje, či je poslucháč viazaný s hodnotou relácie alebo či bola väzba poslucháča na hodnotu relácie uvoľnená. Okrem toho hovorí, že došlo k zmene atribútu relácie.

Tab. 9 Prehľad tried obsiahnutých v balíku javax.servlet.http

Rozhranie HttpServletRequest

Rozhranie **HttpServletRequest** umožňuje **servletom** načítanie informácií týkajúcich sa klientskej **požiadavky**.

Metóda	Popis
String getAuthType()	Vracia schémy autentizácie.
Cookie[] getCookies()	Vracia pole cookie obsiahnuté vo volajúcej požiadavke.
long getDateHeader(String pole)	Vracia hodnotu hlavičkového poľa, ktoré obsahuje dátum a je nazvané pole.
Enumeration<String> getHeaderNames	Vracia vymenovávanie hlavičkových názvov.
String getHeader(String pole)	Vracia hodnotu hlavičkového poľa nazvaného pole.
int getIntHeader(String pole)	Vracia hodnotu typu int odpovedajúcu hlavičkovému polu s názvom pole.
String getMethod()	Vracia HTTP metódu volajúcej požiadavky.
String getPathInfo()	Vracia všetky informácie o ceste nachádzajúcej sa za cestou servletu a pred opytovacím reťazcom URL adresy.
String getPathTranslated()	Vracia všetky informácie o ceste nachádzajúcej sa za cestou servletu a pred opytovacím reťazcom URL adresy prevedenej do podoby skutočnej cesty.
String getQueryString()	Vracia opytovací reťazec obsiahnutý v URL adrese.
String getRemoteUser()	Vracia meno požívateľa, ktorý odoslal požiadavku.

String getRequestedSessionId()	Vracia ID relácie.
String getRequestURL()	Vracia URL adresu.
String getServletPath()	Vracia tú časť URL adresy, ktorá identifikuje servlet.
HttpSession getSession()	Vracia reláciu volajúcej požiadavky. Pokiaľ žiadna relácie neexistuje, je vytvorená nová a tá je vrátená.
HttpSession getSession(boolean nova)	Ak je argument **nova** rovný hodnote true a neexistuje zatiaľ žiadna relácia, je vytvorená nová, ktoré je vrátená. V opačnom prípade je vrátená existujúca relácia volajúcej požiadavky
boolean isRequested SessionIdFromURL()	Vracia hodnotu true, pokiaľ URL adresa obsahuje ID relácie. V opačnom prípade vracia hodnotu false.
boolean isRequested SessionIdFromCookie()	Vracia hodnotu true, pokiaľ cookie adresa obsahuje ID relácie. V opačnom prípade vracia hodnotu false.
boolean isRequested SessionIdValid()	Vracia hodnotu true, ak je požadované ID relácie platné v kontexte aktuálnej relácie.

Tab. 10 Metódy definované rozhraním HttpServletRequest

Rozhranie HttpServletResponse

Toto rozhranie umožňuje **servletom** vytvárať **HTTP** odpovede pre klientov. Súčasťou tohto rozhrania je niekoľko **konštánt** odpovedajúcich rôznym stavovým **kódom**, ktorým je možné priradiť HTTP odpovede. Tak napríklad konštanta SC_OK naznačuje, že HTTP **požiadavka** bola úspešná, zatiaľ čo konštanta SC_NOT_FOUND hovorí, že zdroj **požiadavky** nie je dostupný.

Metóda	Popis
void addCookie(Cookie cookie)	Pridá cookie danú argumentom cookie HTTP odpovede.
Boolean containsHeader(String pole)	Vracia hodnotu true pokiaľ hlavička HTTP požiadavky obsahuje pole s názvom pole.
String encodeURL(String url)	Určuje, či v prípade komunikácie s URL adresou danou argumentom url musí byť ID relácie kódované.
String encodeRedictURL(String url)	Určuje, či v prípade komunikácie s URL adresou danou argumentom url musí byť ID relácie kódované.
void sendError(int kod) throws IOException	Klientovi odosiela chybný kód kod.
void sendRedirect(int kod, String retazec) throws IOException	Klientovi odosiela chybný kód kod spolu so správou danou argumentom **retazec**.
viod sendRedirect(String url) throws IOExcption	Presmerovanie klienta na URL adresu danú argumentom url.
void setDateHeader(String pole, long milisekundy)	Do hlavičky pridá pole s názvom pole a vkladá do neho dátum odpovedajúci argumentu milisekundy.
void setHeader(String pole, String hodnota)	Do hlavičky pridáva pole s názvom pole a vkladá do neho hodnotu odpovedajúcu argumentu **hodnota**.
void setIntHeader(String pole, int hodnota)	Do hlavičky pridáva pole s názvom pole a vkladá do neho hodnotu odpovedajúcu argumentu **hodnota**.
void setStatus(int kod)	Nastavuje stavový kód volajúci odpovede na **kod**.

Tab. 11 Metódy definované rozhraním HttpServletResponse

Rozhranie HttpSession

Toto rozhranie umožňuje **servletom** načítanie a zapisovanie informácií **súvisiacich** s HTTP reláciou. Platí, že pokiaľ bola relácia už zneplatnená, všetky tieto metódy vyvolávajú výnimku **IllegalStateException**.

Rozhranie HttpSessionBindongListener

Toto rozhranie by malo byť implementované tými **objektmi**, ktoré potrebujú byť upozornené na to, že sú viazané k nejakej **HTTP** relácií alebo či ich väzba k takej relácií už bola uvoľnená. V prípade vzniku či zániku väzby sú volané nasledujúce **metódy**:

```
void valueBound(HttpSessionBindingEvent udalost)
void valueUnbound(HttpSessionBindingEvent udalost)
```

Tu argument udalosť **predstavuje** objekt udalostí popisujúcich **väzbu**.

Metóda	Popis
Object getAttribute(String atribut)	Vracia hodnotu zodpovedajúcu atribútu, ktorého názov je daný argumentom **atribut**. Ak nie je daný atribút nájdený, vracia hodnotu null.
Enumeration<String> getAttributeNames()	Vracia vymenovávanie všetkých názvov atribútov súvisiacich s reláciou.
long getCreationTime()	Vracia čas vytvorenia relácie v milisekundách, ktoré uplynuli od polnoci.
String getId()	Vracia ID relácie.
long getLastAccessedTime()	Vracia ten čas, kedy klient naposledy požiadal volajúcu reláciu. Pritom čas je vrátený v milisekundách.
void invalidate()	Zneplatňuje volajúcu reláciu a odstraňuje ju z kontextu.
boolean isNew()	Vracia hodnotu true, pokiaľ je relácia na serveri už vytvorená, ale ešte k nej nepristupoval žiadny klient.
void removeAttribute(String atribut)	Z volajúcej relácie odstráni atribút daný argumentom **atribut**.
void setAttribute(String atribut, Object hodnota)	Pripojuje hodnotu predanú argumentom hodnota s atribútom, ktorého názov zodpovedá argumentu **atribut**.

Tab. 12 Niektoré z metód definovaných rozhraním HttpSession

IX. Trieda Cookie

Trieda Cookie zapuzdruje **cookie**. Cookie sú výhodné predovšetkým z hľadiska sledovania aktivít používateľa. Predpokladajme napríklad, že nejaký **používateľ** navštívi nejaký online obchod. Do cookie sa potom môže **uložiť** meno používateľa, jeho adresa a iné **informácie**. Používateľ potom nemusí tieto dáta znova zadávať pri každej návšteve toho istého **obchodu**.

Servlet je schopný zapísať **cookie** na klientskych počítačoch pomocou metódy **addCookie**(), ktorá je súčasťou rozhrania **HttpServletResponse**. Dáta tohto cookie sú potom vložené do hlavičky HTTP odpovede odosielanej do **prehliadaču**.

Názvy a hodnoty cookie sú **uložené** na klientskom počítači. K informáciám ukladaným do každého cookie patria napríklad:

- **Názov cookie.**
- **Hodnota cookie.**
- **Dátum vypršania platnosti cookie.**
- **Doména a cesta k cookie.**

Dátum vypršania platnosti hovorí, kedy bude **dané** cookie z klientskeho počítača vymazané. Pokiaľ nejakému cookie nie je explicitne **priznaný** žiadny dátum vypršania platnosti, potom také cookie bude vymazané vo chvíli ukončenia aktuálnej relácie **prehliadača**.

Doména a cesta ku cookie určujú, kedy bude dané cookie pridané do **hlavičky** HTTP požiadavky. Pokiaľ používateľ zadá URL **adresu**, ktorej doména a cesta zodpovedajú hodnotám uloženým v cookie, bude cookie odoslané na webový server **spolu** s požiadavkou. V ostatných prípadoch **odoslané** nebude.

Trieda cookie má jediný konštruktor, ktorého **signatúru** vidíte na nasledujúcom riadku:

```
Cookie(String nazov, String hodnota)
```

Ako vidíte, názov a hodnota cookie sú **konštruktoru** predávané ako argumenty.

X. Trieda HttpServlet

Trieda **HttpServlet** rozširuje triedu **GenericServlet**. Táto trieda sa používa predovšetkým pri vývoji servletov, ktoré majú prijímať a spracovávať HTTP **požiadavku**.

Metóda	Popis
Object clone()	Vracia kópiu volajúceho objektu.
String getComment()	Vracia komentár.
String getDomain()	Vracia doménu.
Int getMaxAge()	Vracia maximálnu životaschopnosť cookie v sekundách.
String getName()	Vracia názov.
String path()	Vracia cestu.
boolean getSecure()	Vracia hodnotu true, ak je cookie zabezpečené. V opačnom prípade vracia hodnotu false.
String getValue()	Vracia hodnotu-
int getVersion	Vracia verziu.
boolean isHttpOnly	Vracia hodnotu true, ak má cookie atribút HTTPOnly.
void setCommentString(String retazec)	Do komentáru vkladá reťazec daný argumentom **retazec**.
void setDomainString(String domena)	Doménu nastavuje na tú, ktorá je uvedená v argumente **domena**.
void setHttpOnly(boolean ibaHttp)	Ak je argument iba HTTP rovný true, je do cookie pridaný atribút HttpOnly. Ak je tento argument rovný false, atribút HttpOnly je z cookie odstránený.
void setMaxAge(int sekundy)	Nastavuje maximálnu životnosť cookie na dobu danú

	argumentom sekundy. Pritom životnosť cookie znamená doba, po ktorej uplynutí bude cookie vymazané.
void setPath(String cesta)	Cestu nastavuje na tu, ktorá je uvedená v argumente cesta.
void setSecure(boolean zabezpecenie)	Ak je argument **zabezpecenie** rovný true, cookie bude môcť byť odoslané na server iba pomocou zabezpečeného protokolu, napríklad HTTP či SSL.
void setValue(String hodnota)	Hodnotu nastavuje na tu, ktorá je daná argumentom **hodnota**.
void setVersion(int verzia)	Verziu nastavuje na tu, ktorá je daná argumentom **verzia**.

Tab. 13 Metódy definované triedou Cookie

XI. Trieda HttpSessionEvent

Táto trieda rozširuje triedu **EventObject** a jej inštancie sú generované vždy, keď dôjde ku nejakej zmene relácie. Súčasťou popisovanej triedy je **nasledujúci konštruktor**:

```
HttpSessionEvent(HttpSession relacia)
```

Tu argument **relácia** predstavuje zdroj udalostí. Okrem toho je súčasťou tejto triedy aj metóda getSession(), ktorá má nasledujúcu **definíciu**:

```
HttpSession getSession().
```

Táto metóda vracia tú **reláciu**, v ktorej sa daná udalosť **vyskytla**.

XII. Trieda httpSessionBindingEvent

Trieda **HttpSessionBindingEvent** rozširuje triedu **HttpSessionEvent**. Jej inštancie sú generované vždy, keď je nejaký poslucháč viazaný k objektu typu **HttpSession** alebo keď je väzba nejakého poslucháča k tomuto typu objektu uvoľňovaná. Ďalej sú inštancie tejto triedy generované vždy, keď je nejaký atribút viazaný alebo keď je väzba atribútu uvoľňovaná.

Konštruktory triedy **HttpBindingEvent** vyzerajú nasledovne:

```
HttpSessinBindingEvent(HttpSession relacia, String
nazov)
HttpSessionBindingEvent(HttpSession relacia, String
nazov, Object hodnota).
```

Argument **relacia** predstavuje zdroj udalostí a **nazov** predstavuje názov súvisiaci s tým objektom, ktorý je viazaný alebo ktorého väzba je uvoľňovaná. Ak je viazaný nejaký atribút alebo ak je **uvoľňovaná** väzba nejakého atribútu, je jeho hodnota predávaná argumentom **hodnota**.

Metóda getName() potom **umožňuje** načítanie toho názvu, ktorý je viazaný alebo ktorého väzba je uvoľňovaná. Definíciou tejto metódy je:

```
String getName()
```

Nižšie nájdete deklaráciu metódy **getSession(),** ktorá slúži k načítaniu tej relácie, ku ktorej je nejaký poslucháč viazaný alebo od ktorej je nejaký **poslucháč** uvoľňovaný:

```
HttpSession getSession()
```

Ako poslednú si **ukážeme** metódu **getValue**(), pomocou nej môžete zistiť hodnotu, ktorá je viazaná alebo ktorej väzba je **uvoľňovaná**.

```
Object getValue()
```

XIII. Spracovanie HTTP požiadaviek a odpovedí

Súčasťou triedy HttpServlet sú špecializované metódy **spracovávajúce** rôzne typy HTTP požiadaviek. Pripomeňme si, že sa jedná o **metódy** deDelete(), doGet(), doHead(), doOptions(), doPost(), doPu(), doTrace(). Je možné povedať, že vývojári servletov obvykle prekrývajú iba jednu z týchto **metód**.

Spracovanie HTTP požiadaviek Get

Nami pripravený servlet je **spustený** vo chvíli odoslania formuláru umiestneného na nejakej webovej stránke. Pritom celý príklad je **tvorený** dvoma súbormi. Zdrojový kód **webovej** stránky nájdete v súbore ColorGet.html, zatiaľ čo príslušný servlet je daný kódom **uloženým** v súbore **ColorGetServlet.java**.

Nasledujúci výpis ukazuje zdrojový kód HTML stránky **ColorGet.html**. Ako vidíte, tento kód definuje formulár obsahujúci prvok pre výber a tlačidlo pre odosielanie. Všimnite si, že parameter **action** špecifikuje URL adresu. Tá ukazuje na servlet, ktorý má spracovať **príslušnú** HTTP požiadavku typu Get.

```
<html>
  <head>
  <meta charset="UTF-8">
  </head>
    <body>
        <center>
            <form name="Form1"
                action=http://localhost:8080/exampl
                es/servlets/servlet/servlet/ColorGe
                tServlet>
                <b>Farba:</b>
                <select name="color" size="1">
                    <option
                    value="Red">Červená</option>
                    <option
                    value="Green">Zelená</option>
                    <option value="Blue">Modrá
                    </option>
                </select>
                <br><br>
                <input type="submit"
                value="Odoslať" />
            </form>
        </center>
    </body>
</html>
```

Zdrojový kód servletu **ColorGetServlet.java** je uvedený nižšie. V kóde je prekrytá metóda **doGet()**, s cieľom spracovávania všetkých HTTP požiadaviek typu **Get**, ktoré sú danému servletu predané. Prekrytá metóda **doGet()** využíva metódu **getParameter**() triedy **HttpServletRequest** k načítaniu výberu vykonaného používateľom.

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

public class ColorGetServlet extends HttpServlet {

  public void doGet}HttpServletRequest request,
    HttpServletResponse response)

    throws ServletException, IOExcepion {
      String color=request.getParameter(“color”);

      switch(color) {
        case “Red”: color=“Červená”;
          break;
        case “Green”: color=“Zelená”;
          break;
        case “Blue”: color=“Modrá”;
          break;
      }
```

```
        response.setContentType("text/html;charset=UTF
        -8");
        PrintWriter pw=response.getWriter();
        pw.println("<html>");
        pw.println("<head>");
        pw.println("<meta charset=\"UTF-8\">");
        pw.println("<head>");
        pw.println("<b>Vybraná bola farba:");
        pw.println(color);
        pw.println("<html>");
        pw.close();

    }

}
```

Servlet skompilujte, potom triedy skopírujte do príslušného adresára a postupom popísaným vyššie zaktualizujte súbor **web.xml**. Následne si servlet vyskúšajte, pričom postupujte nasledovne:

1. Spustite servlet **Tomcat**, pokiaľ už nie je **spustený**.
2. Do **adresárneho** riadku webového prehliadača zadajte adresu stránky ColorGet.html a stlačte tlačidlo **Enter**.
3. V zobrazenej stránke si **vyberte farbu**.
4. Webovú **stránku odošlite**.

Po dokončení všetkých týchto krokov by ste vo webovom prehliadači mali vidieť **odpoveď** dynamicky vygenerovanú **servletom**.

XIV. Spracovanie HTTP požiadaviek POST

Servlet je **spustený** vo chvíli, keď je odoslaný formulár tvoriaceho súčasť webovej stránky. Aj tento príklad je tvorený dvomi **súbormi**. Zdrojový kód webovej stránky nájdete v súbore ColorPost.html, zatiaľ čo zdrojový kód servletu nájdete v súbore **ColorPostServlet.java**.

Nasledujúci výpis ukazuje HTML **zdrojový** kód stránky **ColorPost.html**. Ako vidíte, zdrojový kód je prakticky identický s kódom stránky **ColorGet.html** a to až na dva rozdiely: po prvé parameter metódy formulára explicitne hovorí, že ma byť **použitá** metóda Post a za druhé parameter akcie formuláru špecifikuje iný **servlet**.

```
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <center>
        <form name="Form1" method="post"
        action="http://localhost:8080/examples/serv
        lets/servlet/servlet/ColorGetServlet">
            <b>Farba:</b>
            <select name="color" size="1">
                <option
                value="Red">Červená</option>
                <option
                value="Green">Zelená</option>
                <option value="Blue">Modrá
                </option>
            </select>
            <br><br>
            <input type="submit" value="Odoslať" />
        </form>
    </center>
  </body>
</html>
```

Dostávame sa k **výpisu** druhého potrebného zdrojového kódu a to servletu **ColorPostServlet.java**. Metóda **doPost()** je prekrytá s cieľom umožniť spracovanie všetkých HTTP požiadaviek typu POST odoslaných tomuto **servletu**.

Ako vidíte, prekrytá metóda využíva metódu **goParameter()** triedy **HttpServletRequest** k načítaniu výberu vykonaného používateľom. Následne je vytváraná HTTP **odpoveď**.

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

public class ColorGetServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws
    ServletException, IOExcepion {

      String color=request.getParameter("color");
      switch(color) {
        case "Red" color="Červená";
          break;
        case "Green" color="Zelená";
          break;
        case "Blue" color="Modrá";
          break;
      }
```

```
        response.setContentType("text/html;charset=UTF
        -8");
        PrintWriter pw=response.getWriter();
        pw.println("<html>");
        pw.println("<head>");
        pw.println("<meta charset=\"UTF-8\">");
        pw.println("<head>");
        pw.println("<b>Vybraná bola farba:");
        pw.println(color);
        pw.println("<html>");
        pw.close();

    }

}
```

Servlet **skompiluje** a následne vykonajte tie kroky, ktorých popis nájdete v predchádzajúcich častiach. Potom si servlet **vyskúšajte**.

XV. Práca s cookies

Skúsme si teraz **pripraviť** servlet, na ktorom by sme si ukázali prácu s **cookies**. Servlet by mal byť **volaný** vždy, keď dôjde k odoslaniu formuláru umiestneného na webovej stránke. Celý náš príklad je tvorený troma **súbormi**.

Súbor	Popis
AddCookie.html	Umožňuje používateľovi zadanie hodnoty cookie nazvaného MyCookie.
AddCookieServlet.java	Reaguje na odosielanie formuláru zo stránky AddCookie.html.
GetCookiesServlet.java	Zobrazuje hodnoty cookie.

Tab. 14 prehľad súborov Cookies

Zdrojový kód webovej **stránky AddCookie.html** nájdete v nasledujúcom výpise. Súčasťou pripravenej stránky je textové pole, do ktorého môže používateľ zadať nejakú hodnotu. Ďalej sa na tejto stránke nachádza aj tlačidlo pre **odoslanie**.

Po stlačení tlačidla je hodnota z textového poľa odoslaná servletu **AddCookieServlet.java** a to prostredníctvom http požiadavky typu **POST**.

```
<html>
  <head>
    <meta charset="UTF-8">
  </head>
    <body>
        <center>
            <form name="Form1" method="post"
            action="http://localhost:8080/examples/
            servlets/servlet/AddCookie/Servlet">
            <b>Zadajte hodnotu pre MyCookie</b>

            <input type="text" name="data" size=25
            value="">
            <br><br>
            <input type="submit" value="Odoslať">
            </form>
        </center>
    </body>
</html>
```

Ďalší výpis potom **obsahuje** zdrojový kód servletu **AddCookieServlet.java**. Servlet načíta hodnotu parametrov nazvaného „**data**". Následne vytvára objekt typu Cookie, ktorý má názov „**MyCookie**" a obsahuje hodnotu parametra „**data**". Metódou **addCookie()** je takto vytvorený objekt pridaný k hlavičke HTTP odpovede.

Nakoniec je vo webovom prehliadači zobrazovaná informácia o vykonanej úlohe:

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

public class addCookieServlet extends HttpServlet {

  public void doPost(HTTPSErvletRequest request,
    HttpServletResponse response) throws
    ServletException, IOExeption {

    // Načítanie parametrov z HTTP požiadavky.
    request.setCharacterEncoding("UTF-8");
    String data=request.getParameter("data");

    // Vytvorenie cookie.
    Cookie cookie=new Cookie
    ("My Cookie", URLEncoder.encode(data, "UTF-8"));

    // Pridanie cookie k HTTP odpovede.
    response.addCookie(cookie);

    // Zobrazenie výstupu v prehliadači.
    response.setContentType("text/html; charset=UTF-
    8");
    PrintWriter pw=response.getWriter();
    pw.println("<html>");
    pw.println("<head>");
    pw.println("<meta charset=\"UTF-8\">");
```

```
    pw.println("<head>");
    pw.println("<b>Hodnota cookie MyCookie bola
    nastavená na:");
    pw.println(data);
    pw.println("</html>");
    pw.close();
  }
}
```

Posledný výpis potom ukazuje kód servletu **GetCookiesServlet.java**. Ako vidíte, v kóde je volaná metóda **GetCookies(),** umožňujúca načítanie všetkých cookie, ktoré sú súčasťou HTTP požiadaviek typu GET. Názvy a hodnoty týchto cookie sú potom zapísané do HTTP odpovede. Pre načítanie príslušných informácií sú volané metódy **getName**() a **getValue().**

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

public class GetCookiesServlet extends HttpServlet
{

  public void doGet(HTTPServletRequest request,
    HttpServletResponse response) throws
    ServletException, IOExeption {

    // Načítanie cookie z hlavičky HTTP požiadaviek.
    Cookie[] cookies=request.getCookies();

    // Zobrazenie cookie vo výstupe.
    response.setContentType("text/html; charset=UTF-
    8");
```

```
    PrintWriter pw=response.getWriter();
    pw.println("<html>");
    pw.println("<head>");
    pw.println("<meta charset=\"UTF-8\">");
    pw.println("<head>");
    pw.println("<b>");

    for(int i=0; i<cookies.length; i++) {
      String name=cookies[i].getName();
      String value=URLDecoder.decode
        (cookies[i].getValue(), "UTF-8");
      pw.println("name="+name +";value="+value);
    }

    pw.println("</html>")
    pw.close();
  }
}
```

Oba pripravené **servlety** skompilujte. Potom vzniknuté súbory tried skopírujte do príslušného adresára a postupom popísaným vyššie zaktualizujte súbor **web.xml**. Následne si celý príklad vyskúšajte, pričom postupujte **takto**:

1. **Spustite** server **Tomcat**, pokiaľ už nejde.
2. Do adresného riadku webového prehliadača **zadajte adresu** stránky AssCookie.html a stlačte **Enter**.
3. **Zadajte** nejakú hodnotu **cookie MyCookie**.
4. **Webovú** stránku **odošlite**.

Po dokončení všetkých týchto krokov sa vám vo webovom prehliadači zobrazí správa informujúca vás o **nastavení** hodnoty cookie:

Potom skúste do adresného riadku webového prehliadača zadať nasledujúcu adresu:

http://localhost:8080/examples/servlets/servlet/GetCookiesServlet

Vo webovom prehliadači sa vám následne zobrazí **názov** aj **hodnota** cookie.

V tomto príklade sme nevyužili metódu **setMaxAge()** triedy Cookie, ktorou by sme mohli vytváranému cookie explicitne priradiť nejakú dobu **platnosti**. Z tohto dôvodu platnosť cookie skončí vo chvíli ukončenia relácie **prehliadača**.

XVI. Sledovanie relácií

Protokol HTTP je **bez stavový**. To znamená, že každá **požiadavka** je celkom **nezávislá** na požiadavke predchádzajúcej. V niektorých prípadoch je však potrebné **stavové** informácie ukladať a to preto, aby bolo možné zhromaždiť informácie z niekoľkých interakcií, ktoré prebehnú medzi **prehliadačom** a serverom. A presne taký mechanizmus ponúka **relácia**.

Reláciu je možné **vytvoriť** pomocou metódy getSession() rozhrania HttpServletRequest vracajúci objekt typu **HttpSession**. V tomto objekte môže byť uložená sada väzieb prepájajúca názvy s **objektmi**.

K správe týchto **väzieb** sa dá potom použiť metódy setAttribute(), getAttrbute(), **getAttributesNames()** a **removeAttribute()** rozhrania Httpession. Pritom stav relácie je zdielaný všetkými servletmi prepojenými s **klientom**.

Ukážku využitia stavu relácie nájdete v nasledujúcom **servlete**. Metóda **getSession()** načíta aktuálnu reláciu. Pritom pokiaľ žiadna **relácia** ešte neexistuje, je vytvorená nová. Následne je volaná metóda **getAttribute()**, pomocou ktorej je **načítaný** objekt viazaný s názvom „**date**“. Jedná sa o objekt typu Date zapuzdrujúci dátum a čas posledného **prístupu** k danej stránke.

Potom je vytváraný ďalší objekt typu **Date**, tento krát však zapuzdrujúci aktuálny dátum a čas. Volaním metódy **setAttribute()** je názov „date“ prepojený s týmto objektom.

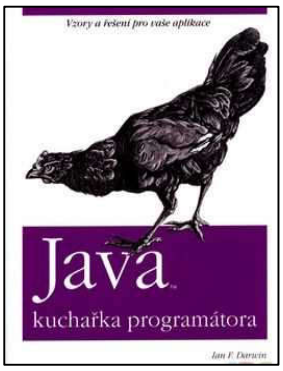
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.nio.charset.*;
import java.net.*;
import java.util.*;

public class DateServlet extends HttpServlet {

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

    // Načítanie objektu HttpSession.
    HttpSession hs=request.getSession(true);

    // Vytvorenie objektu typu PrintWriter.
    response.setContentType("text/html; charset=UTF-
    8");
    PrintWriter pw=response.getWriter();
    pw.println("<html>");
    pw.println("<head>");
    pw.println("<meta charset=\"UTF-8\">");
    pw.println("<head>");
    pw.println("<b>");

    // Zobrazenie dát a času posledného prístupu.
    Date date = (Date)hs.getAttribute("date");
    if(date != null) {
      pw.print("Posledný prístup: " + date +
      "<br>");
    }
```

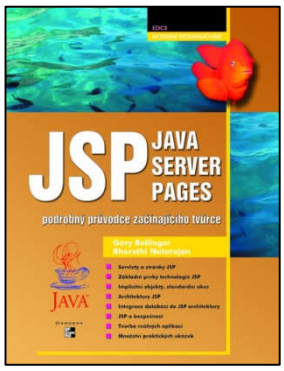
```
    // Zobrazenie aktuálneho dáta a času.
    date = new Date();
    hs.setAttribute("date", date);
    pw.print("Aktuálny dátum: " + date);

    pw.println("</html>");
    pw.close();

    }

}
```

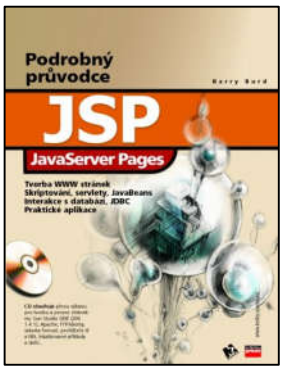
Po spustení tohto **servletu** vám prehliadač zobrazí iba jeden riadok obsahujúci aktuálny dátum a čas. Pri každom nasledujúcom **načítaní** vám však prehliadač zobrazí dva riadky. Prvý z nich bude obsahovať dátum a čas **posledného** prístupu k servletu, zatiaľ čo druhý bude obsahovať **aktuálny** dátum a čas.

XVII. Odporúčaná literatúra a zdroje

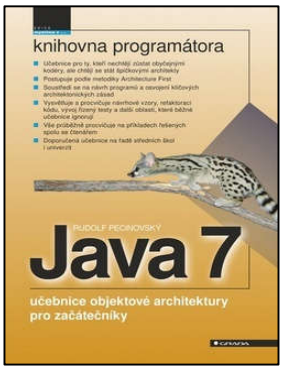
1. **Java kuchařka programátora -** Ian F.Darwin
2. **Java 7 -** Herbert Schildt
3. **JSP Java Server Pages -** Gary Bollinger
4. **JSP Java Server Pages Podrobný průvodce -** Barry Burd

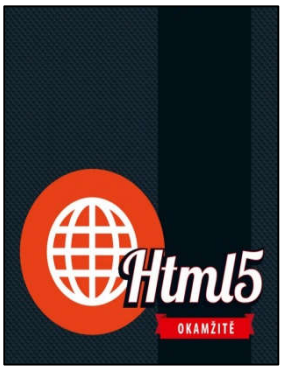






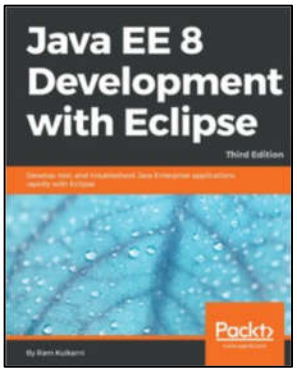
5. **Mistrovství Java -** Herbert Schildt
6. **Java 7 -** Rudolf Pecinovský
7. **1001 tipů a triků pro jazyk Java -** Bogdan Kiszka
8. **HTML5 Okamžitě -** Tiffany B. Brown , Sandeep Panda

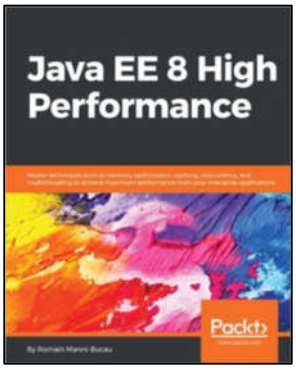




Zahraničná literatúra

1. **Learning Java EE 8 -** Sebastian Daschner
2. **Java EE 8 Application Development -** Tomasz Lelek
3. **Java EE 8 Development with Eclipse -** Ram Kulkarni
4. **Java EE 8 High Performance -** Romain Manni-Bucau





5. **Optimizing Java -** Benjamin J Evans
6. **Effective Java -** Joshua Bloch
7. **Java Performance -** Scott Oaks
8. **Head First Java -** Kathy Sierra, Bert Bates