PA165 - webové aplikace

Jan Pavlovič

PA165 - webové aplikace

Jan Pavlovič

Vydáno 18. října 2004

Obsah

Seznam obrázků

Seznam příkladů

Kapitola 1. Úvod - Servlety a JSP

Technologie JSP a Java Servlety jsou jedny ze základních technologií Java™ 2 Platform, Enterprise Edition (J2EE).

Servlet je program v Javě, který rozšiřuje funkcionalitu webového serveru, generuje dynamický obsah a komunikuje s webovými klienty pomocí request-response paradigmatu.

JavaServer Pages™ (JSP) je abstrakce servletového modelu. JSP stránky jsou rozšiřující webovou technologií, využívající značkovací šablony, specifikovatelné elementy, skriptovací jazyky a server-side Java objekty pro zpřístupnění dynamického obsahu klientovi. Nejčastěji se jako šablony používají HTML či XML elementy a ve většině případů je klientem webový browser.

Specifikace servletu a JSP je dostupná na: http://java.sun.com/products/servlet a http://java.sun.com/products/jsp.

Servlety

Servlety jsou reakcí Javy na CGI scripty. Jedná se o programy v Javě, které běží na webovém serveru a odpovídají na požadavky ze stany klientů. Servlety nejsou spjaty z žádným konkrétním client-server protokolem, nicméně nejširší využití servletů je s HTTP protokolem. Slovem „Servlet“ je tedy často míněno „HTTP Servlet“.

Servlety jsou implementací tříd v balíku `javax.servlet` (základní Servlet framework) a `javax.servlet.http` (rozšíření Servlet frameworku pro servlety odpovídají na HTTP požadavky). Jelikož jsou servlety napsány ve vysoce portabilním jazyku a s splňují požadavky na framework, umožňují vytváření sofistikovaných serverových aplikací nezávisle na operačním systému.

Servlety se používají zejména pro:

- Zpracování a ukládaní dat z HTML formulářů.
- Generovaní dynamického obsahu např. vracení databázových dotazů klientovi.
- Manipulace se stavovými informacemi nad bezstavovým HTTP protokolem např. realizace on-line nákupního sytému, který souběžně obsluhuje několik zákazníků a přiřazuje každý požadavek odpovídajícímu zákazníkovi.

Servlety vs CGI

Tradiční způsob přidávání další funkcionality webovému serveru je pomocí Common Gateway Interface (CGI), jedná se o jazykově nezávislý interface, který umožňuje serveru spouštět externí procesy. Každý požadavek je zodpovězen separátní instancí CGI scriptu.

Servlety mají nad CGI několik výhod:

- Servlet neběží v separátním procesu. Což odstraňuje zátěž s vytvářením nového procesu pro každý požadavek.
- Servlet zůstává v paměti mezi požadavky. CGI script je nutné nahrávat a spouštět pro každý požadavek.
- Existuje pouze jediná instance servletu, která souběžně obsluhuje všechny požadavky. Tímto se ušetří paměť a zároveň může servlet jednoduše obsluhovat všechna data.
- Servlet může běžet v sandboxu a mít tak pevně definované bezpečnostní omezení.

Použití servletu

Na obrázku 1 je zobrazen jeden z nejčastějších využití servletů. Uživatel (1) vyplní formulář, který se odkazuje na servlet a kliknutím na submit tlačítko vyšle požadavek na informaci (2). Server (3) lokalizuje požadovaný servlet (4). A teď vyhodnotí a vrátí požadované informace ve formě web stránky (5). Ta je poté zobrazena v uživatelově prohlížeči (6).

Základní struktura servletů

Servlet ve své nejobecnější formě je instance třídy, která implementuje `javax.servlet.Servlet` interface. Většina servletů je potomkem jednoho ze standardních implementací tohoto interfacu, jmenovitě `javax.servlet.GenericServlet` a `javax.servlet.http.HttpServlet`.

Při inicializaci servletu načte server třídu servletu (a ostatní odkazované třídy) a vytvoří instanci voláním bezparametrového konstruktoru. Následně zavolá metodu servletu `init(ServletConfig config)`. Servlet při vykonávání této metody uloží `ServletConfig` objekt, který bude dostupný metodou `getServletConfig()`. Toto vše je obstaráno GenericServletem. Servlety, které jsou potomky `GenericServletu` (nebo jeho podtřídy `HttpServlet`) by měli volat `super.init(config)` na začátku metody `init`. Vzniklý objekt `ServletConfig` obsahuje parametry servletu a odkaz na `ServletContext` obsahující runtime environment informace. Metoda `init` je v životním cyklu servletu volána pouze jednou.

Poté co je servlet inicializován, je jeho metoda `service(ServletRequest req, ServletResponse res)` volána pro každý požadavek na servlet. Tato metoda je volána souběžně (např. multiple threads mohou volat tuto metodu ve stejný okamžik) a tudíž je nutné, aby byla implementována jako thread-safe.

V případě potřeby odstranění servletu z paměti (např. z důvodu nahrání nové verze nebo vypínaní serveru) je volána metoda `destroy()`.

HTTP

Ještě před tím, než se pustíme do psaní našeho prvního servletu, musíme znát několik základních rysů HTTP protokolu.

HTTP je request-response orientovaný protokol. HTTP požadavek se skládá z metody request , URI, hlaviček a těla dotazu. HTTP odpověď obsahuje kód odpovědi, hlavičky a tělo.

Metoda `service` HttpServletu rozděluje požadavek vstupním metodám servletu podle typu HTTP požadavku. Jsou rozeznávaný standardní HTTP/1.1 metody: GET, HEAD, PUT, POST, DELETE, OPTIONS a TRACE. Ostatní metody jsou vráceny jako: Bad Request HTTP error. HTTP metoda `XXX` je přiřazena metoda servletu `doXxx`, (GET -> `doGet()`). Všechny tyto metody očekávají parametry „(`HttpServletRequest` req, `HttpServletResponse` res)“. Metody `doOptions()` a `doTrace()` mají dostatečnou defaultní implementaci a obvykle je nepředefinovávají. Metoda HEAD (která vrací pouze hlavičky) je řešena voláním `doGet()` a ignorací výstupu této metody. Tím nám zůstávají metody doGet, doPut, doPost a doDelete, jejich výchozí implementace v HttpServletu vrací: Bad Request HTTP error. Podtřída HttpServletu předefinovává jednu či více těchto metod smysluplnou implementací.

Data požadavku vstupují do servletu přes první argument typu `HttpServletRequest` (který je podtřídou obecnější třídy `ServletRequest`). Odpověď může být vytvořena skrz druhou proměnou, která je typu `HttpServletResponse` (podtřída `ServletResponse`).

V okamžiku, kdy v našem prohlížeči zadáme požadavek na URL, je použita metoda GET. Odpověď se bude skládat z těla odpovědi a hlaviček popisující tělo (obzvláště Content-Type a Content-Encoding). Pokud posíláme HTML formulář, můžeme použít metodu GET nebo POST. S GET požadavkem jsou parametry zakódovaný v URL a s POST požadavkem jsou přeneseny v těle požadavku.

Náš první servlet

Začneme obvyklým „Hello World“ příkladem

Příklad 1.1. HelloClientServlet.java

```
 1:  import java.io.*;
 2:  import javax.servlet.*;
 3:  import javax.servlet.http.*;
 4:
 5:  public class HelloClientServlet extends HttpServlet
 6:  {
 7:    protected void doGet(HttpServletRequest req,
 8:                         HttpServletResponse res)
 9:              throws ServletException, IOException
10:    {
11:      res.setContentType("text/html");
12:      PrintWriter out = res.getWriter();
13:      out.println("<HTML><HEAD><TITLE>Hello Client!</TITLE>"+
14:                  "</HEAD><BODY>Hello Client!</BODY></HTML>");
15:      out.close();
16:    }
17:  }
```

Podívejme se jak daný servlet funguje.

Řádky 1 až 3 importují balíky potřebné pro běh servletu.

```
 1:  import java.io.*;
 2:  import javax.servlet.*;
 3:  import javax.servlet.http.*;
```

Třída servletu je deklarovaná na řádku 5. Servlet dědí třídu javax.servlet.http.HttpServlet, standardní třída pro HTTP servlety.

```
 5:  public class HelloClientServlet extends HttpServlet
```

Na řádcích 7 až 16 je předefinovaná metoda `doGet()` HttpServletu.

```
 7:    protected void doGet(HttpServletRequest req,
 8:                         HttpServletResponse res)
 9:              throws ServletException, IOException
10:    {
         ...
16:    }
```

Na řádku 11 je pro nastavení content type odpovědi použita metoda třídy `HttpServletResponse`. Všechny hlavičky odpovědi musí být nastaveny před tím než je zavolán `PrintWriter` nebo `ServletOutputStream` pro výpis dat.

```
11:      res.setContentType("text/html");
```

Řádek 12: pomocí třídy `PrintWriter` je zapsán text do odpovědi.

```
12:      PrintWriter out = res.getWriter();
```

13 až 14 no comment ...

```
13:      out.println("<HTML><HEAD><TITLE>Hello Client!</TITLE>"+
14:                  "</HEAD><BODY>Hello Client!</BODY></HTML>");
```

Po ukončení zápisu zavíráme na řádku 15 `PrintWriter`.

```
15:       out.close();
```

Tento řádek je uveden pro úplnost. Není striktně vyžadován. Web server zavírá `PrintWriter` nebo `ServletOutputStream` automaticky po návratu metody `service`.

JSP

TODO

Kapitola 2. Servlet/JSP containers

Servlet/JSP container umožňuje používání JSP souborů. Container dostane (od webového serveru) JSP soubor, ke zpracování, přeloží ho do java-byte kódu, provede a výstup předá zpátky (webovému serveru).

Tomcat

Jedním z nejpoužívanějších Servlet/JSP containerů je Apache Jakarta Tomcat. http://jakarta.apache.org/tomcat

Tomcat verze 5.5 implementuje Servlet 2.4 a JavaServer Pages 2.0 specifikaci a obsahuje mnoho dalších vlastností, které z něj činní vhodnou platformu pro vývoj a provoz webových aplikací a webových služeb.

Adresářová struktura Tomcatu

- `bin` - startovací scripty
- `common` - adresář pro třídy a balíky sdílené všemi aplikacemi i serverem
- `conf` - soubory s konfigurací serveru
- `logs` - výstup a logy serveru
- `server` - knihovny nezbytné pro běh serveru
- `shared` - adresář pro třídy a balíky sdílené všemi aplikacemi
- `webapps` - úložiště pro aplikace
- `work` - pracovní adresář pro běžící aplikace
- `temp` - adresář používaný JVM pro dočasné soubory (java.io.tmpdir)

Obsluha serveru

Tomcat se spouští pomocí příkazu **startup.sh**, poté je server dostupný na portu 8080 na daném serveru http://localhost:8080

Ukončení běhu serveru se provádí příkazem **shutdown.sh**

Struktura aplikace

Top-level adresář ve struktuře aplikací je i kořenovým adresářem naší aplikace. Po nahrání aplikace na server bude dostupná právě pod jménem aplikace: např soubor index.html v aplikaci catalog, bude odkazovaná jako http://server/catalog/index.html

Vnitřní struktura aplikace odpovídá formátu WAR souboru

- `*.html, *.jsp, atd.` - HTML a JSP stránky musí být spolu s ostatními soubory viditelné pro prohlížeče klientů (to platí i pro JavaScript, CSS a obrázky). V rozsáhlejších aplikacích se přistupuje do rozdělení těchto souborů do jednotlivých podadresářů
- `/WEB-INF/web.xml` - (Web Application Deployment Descriptor) je XML soubor pro naši aplikaci popisující servlety a ostatní komponenty v aplikaci. Dále obsahuje případné definice inicializačních parametrů a bezpečnostních omezen.
- `/META-INF/context.xml` (Tomcat Context Descriptor) je soubor, který může být použit pro specifické definice Tomcatu jako: loggers, data sources, session manager configuration a další.
- `/WEB-INF/classes/` - Adresář obsahující zkompilované soubory servletů a ostatních tříd, které nejsou zabalené v JAR archívu. Pokud máme třídy organizované v balíčcích, musíme respektovat tuto adresářovou strukturu i v /WEB-INF/classes/ např. třída com.mycompany.mypackage.MyServlet musí být uložena v souboru /WEB-INF/classes/com/mycompany/mypackage/MyServlet.class.
- `/WEB-INF/lib/` - Adresář obsahující JAR soubory.

Struktura zdrojových kódů aplikace

Základní myšlenka strukturu naší aplikace je oddělení zdrojových kódu od binárních. Takovéto rozdělení přináší následující výhody:

- Obsah zdrojových adresářů lze jednoduše spravovat, přesouvat a zálohovat.
- Správa zdrojových kódů je jednodušší pokud adresáře obsahují pouze zdrojové soubory.
- Distribuční soubory jsou hierarchicky oddělené od zdrojových.

Později uvidíme, že vytváření hierarchické struktury adresářů za pomocí antu je velmi snadné.

Doporučená struktura adresářů:

- `docs/` - Dokumentace.
- `src/` - Java zdrojové kódy pro servlety, beany a další třídy. Pokud jsou zdrojové soubory organizovány do balíčků (což je vřele doporučováno), musí struktura balíčku odpovídat struktuře adresářů.
- `web/` - statické stránky (HTML, JSP, JavaScript, CSS a obrázky). Tento adresář bude kořenovým adresářem webové aplikace. Veškerá podadresářová struktura bude zachována.
- `web/WEB-INF/` - konfigurační soubory pro aplikaci (web.xml), taglibs a jiné. Soubory v tomto adresáři nebudou přístupné klientům. Z tohoto důvodu je právě toto místo vhodné pro ukládaní konfiguračních souboru s citlivými informacemi (hesla k přístupu do databáze).

V průběhu kompilace dojde k vytvoření dvou adresářů:

- `build/` - po spuštění antu obsahuje tento adresář kompletní obraz přeložené aplikace.
- `dist/` - do toho adresáře umístí ant war soubor s aplikací

Není vhodné do aplikace začleňovat JAR soubory běžných aplikací. Ty je vhodnější umístit na server do sdílených složek `common` nebo `shared`.

Taktéž není příliš vhodné umisťovat přiložené třídy do SVN repozitory.

web.xml

V souboru `web.xml` je uveden popis aplikace, použití různých filtrů, taglibs, servletů atd. V následují ukázce je uvedeno na jaký odkaz má Tomcat přemapovat servlet `HelloWorldExampleServlet`, který je spouštěn třídou `mypackage.HelloWorldExample`.

Příklad 2.1. web.xml

```
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>Hello, World Application</display-name>
  <description>
    This is a simple web application with a source code organization
    based on the recommendations of the Application Developer's Guide.
  </description>

  <servlet>
    <servlet-name>HelloWorldExampleServlet</servlet-name>
    <servlet-class>mypackage.HelloWorldExample</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorldExampleServlet</servlet-name>
    <url-pattern>/HelloWorldExample</url-pattern>
  </servlet-mapping>
</web-app>
```

Instalace aplikace

Existují dva hlavní způsoby jak svou aplikaci do Tomcatu nainstalovat:

- Použít webovou nástavbu tomcat manageru, což je nástroj který umožňuje instalovat do Tomcatu aplikace. Buď můžeme použít jeho grafickou nástavbu: a aplikaci zabalenou do war jím nainstalovat http://kore.fi.muni.cz:8080 a nebo, což je úplně nejjednodušší, použít Ant, který za nás aplikaci do waru sestaví a managerem nainstaluje. K tomu abychom mohli manažer používat, je nutné buď znát heslo managera nebo jako uživatel mít roli managera. Jelikož uživatel v roli managera může odstraňovat veškeré aplikace, dávejte si pozor, aby jste neodstranili aplikaci i někomu jinému!
- Použít antový target, který používá třídy z balíku `catalina-ant.jar`. Balík obsahuje i další targety, které umožňují plně využít veškeré schopnosti tomcat managera.

Ant

Použití Antu je snadná věc, kterou zvládne opravdu každý a ušetří si tak spoustu práce. Pro instalaci do tomcat manageru je potřeba nakopírovat soubor `catalina-ant.jar` z tomcatu do `$ANT_HOME/lib`. Ant v modulech již uvedený soubor obsahuje a tak není potřeba nic kopírovat. Stačí přidat Ant:

```
module add ant
```

Ant používá jako konfigurační Makefile soubor `buil.xml` v kterém jsou uvedeny jednotlivé *targets* a parametry. Pro základní použití není potřeba konfiguračnímu souboru příliš rozumět, navíc je možné parametry includovat z externího souboru většinou nazvaném `buil.properties`, takže když potřebujeme něco změnit v konfiguraci stačí změnit několik údajů v tomto souboru a do `buil.xml` vůbec nezasahovat.

Příklad 2.2. build.properties

```
app.name=pokus12
catalina.home=/packages/share/tomcat
manager.url=http://kore.fi.muni.cz:8080/manager
manager.username=manager
manager.password=manager
```

Nyní již stačí aplikaci zkompilovat a nainstalovat.

`ant compile`
aplikaci pouze zkompiluje a výsledný `war` soubor uloží do adresáře `/dist`

`ant install`
aplikaci zkompiluje a nainstaluje do adresáře `/webapps/${app.name}`, která bude dostupná jako http://kore.fi.muni.cz:8080/${app.name}.

`ant deploy`
přenese war soubor aplikace do tomcatu a nainstaluje do adresáře `/webapps/${app.name}`, aplikace bude dostupná jako http://kore.fi.muni.cz:8080/${app.name}. T[Aento cíl umožňuje také použití vlastní konfigurací kontextu aplikace, které uvedeme v souboru `context.xml`. Aplikace bude jednak v tomcatu nainstalována a navíc bude do tomcatu přenesen i distribuční war soubor.

`ant undeploy`
odstraní deploynutou aplikaci i war soubor.

`ant stop`
aplikaci pozastaví (nebude dostupná z webu).

`ant start`
aplikaci spustí pozastavenou aplikaci.

`ant reload`
zastaví aplikaci a provede její znovu načtení.

Poznámka

konfigurace aplikace uložená v souboru `/WEB-INF/web.xml` není při reloadu znovu načtena a je použita konfigurace předchozí. Pokud jsme udělali změny konfiguračním souboru je nutné aplikaci nejprve zastavit a opětovně spustit.

Ukázková aplikace

Nejlepší je se inspirovat již nějakou hotovou aplikací: http://www.fi.muni.cz/~xpavlov/tomcat/app1.tar.gz. K dispozici je i originální ukázková aplikace z distribuce tomcatu http://jakarta.apache.org/tomcat-5.0/tomcat-docs/appdev.

mod_jk2

V případě, že se do apache zkompiluje mod_jk2, není nutné pro použití JSP souboru specifikovat port, na kterém tomcat běží. Apache pak sám pozná, který soubor má předat tomcatu ke zpracování a který soubor má zpracovat sám.