WEBOVÉ APLIKAČNÍ RÁMCE

Tomáš Pitner[a]
a) Masarykova univerzita v Brně, Fakulta informatiky, Botanická 68a, 602 00 Brno, e-mail: tomp@fi.muni.cz

Abstrakt
Prakticky zaměřený příspěvek seznamuje s rámci pro vývoj webových aplikací jako s ucelenými prostředími usnadňujícími návrh, implementaci, provoz a údržbu rozsáhlých webových aplikací. Oblast bude zmapována na rámcích určených pro jazyk Java.

1. WEBOVÝ RÁMEC A JEHO ARCHITEKTURA

1.1 Úvod

Webové rámce poskytují více či méně ucelené prostředí, do něhož tvůrce aplikace vkládá své šablony, programový kód, konfigurace a popisovače a tak vytváří webovou aplikaci. Vlastní fungování webového rámce nejlépe vystihuje *Hollywood Principle: „Don't call us, we will call you!"* – rámec se nepoužívá jako knihovna, jejíž funkce, resp. objekty ve svých programech voláme/využíváme, nýbrž jako prostředí, které "funguje samo o sobě" a jehož chování pouze modifikujeme podle potřeb. Rovněž samo anglické označení *framework* charakterizuje ucelenost a svébytnost. Abychom dokázali role webových rámců pochopit, je třeba se nejdříve podívat celkově na architekturu webových aplikací.

1.2 Třívrstvá architektura, koncepce MVC

Architektury informačních i jiných programových systémů jsou často konstruovány jako vícevrstvé (multi-tier, multi-layer), standardně se hovoří o minimálně třech vrstvách: *prezentační* (nejblíže uživateli), *aplikační* (ta, kde se skutečně „počítá") a *datová* (kde se obsluhuje perzistentní úložiště zpracovávaných informací).

MVC, neboli Model-View-Controller je obecná koncepce *objektové* architektury řešící základní oddělení výše uvedených vrstev. Netýká se pouze webových aplikací; je jako metodika používána i v jiných, převážně však interaktivních aplikacích, kde je její smysl nejvíce vidět.

Model je tvořen objekty – daty aplikace – a aplikační logikou. *View* jsou pomocné objekty (komponenty) zajišťující pohled na (tj. prezentaci) modelu. Model je v koncepci MVC *zcela nezávislý* na View. To přináší snazší udržovatelnost – pohled (view) na model lze vcelku snadno vyměnit za jiný, upravit, atd. *Controller* řídí komunikaci s uživatelem a mezi komponentami modelu a pohledů.

1.3 Webové aplikace na platformě Java

Než se budeme zabývat přímo webovými rámci pro javovou platformu, představíme stručně prostředí, v němž se webové aplikace programují a provozují. Základním rozhraním, kolem něhož se 99 % javových aplikací pro web buduje, je *Java Servlet API* [1] a jeho nadstavby. Toto rozhraní předepisuje a dává prostředky, jak programovat serverové komponenty webových aplikací, tj. *servlety* a *Java Server Pages*. Servlety lze velmi vzdáleně přirovnat k CGI skriptům, mají však vůči nim řadu předností. Stránky JSP jsou javovým protějškem např. k PHP, ASP a dalším „Server Pages".

Tyto komponenty lze provozovat v prostředí tzv. *javových webových kontejnerů*, což jsou buďto samostatné serverové aplikace, anebo tvoří volnou součást jiných, např. HTTP, serverů (Apache). Nyní už k funkcionalitě samotných rámců.

2. FUNKCIONALITA WEBOVÝCH RÁMCŮ

2.1 Oddělení prezentační vrstvy

Více či méně dokonalé oddělení aplikační a prezentační vrstvy budované aplikace patří k základním cílům téměř všech webových rámců. Většina rámců (např. Velocity, [2]) nabízí pro popis prezentační vrstvy webových aplikací *jazyk šablon* (template language), v němž lze – často s použitím klasických editorů HTML, jako je např. Macromedia Dreamweaver, Microsoft FrontPage, HomeSite, HTML Kit a dalších) – vytvářet graficky kvalitní webové stránky, do nichž jsou až na základě klientského dotazu vkládány výstupy dynamických komponent. To také umožňuje ponechat design stránek na odborníkovi a programátor se věnuje jen tvorbě dynamických komponent.

```
<html><head>
  <title>#showWebsiteTitle()</title>
  <style type="text/css">#includePage("_css")</style>
  #showRSSAutodiscoveryLink()
</head>
<body>
<div id="Content">
    <center>
      <h1>#showWebsiteTitle()</h1>
      <p class="descrip">#showWebsiteDescription()</p>
      #showWeblogCategoryChooser()<br>
    </center>
    #showWeblogEntries("_day" 15)
    <hr />
    #showReferers( 40 25 )
</div>
</body></html>
```

Obr. 1 Šablona pro weblog (Velocity)

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-faces"
  prefix="s" %>
<f:use_faces>
    <s:form action="/listFlights">
     <h:input_text id="fromCity" valueRef="FlightSearchForm.fromCity"/>
     <h:input_text id="toCity" valueRef="FlightSearchForm.toCity"/>
     <h:input_text id="departureDate"
       valueRef="FlightSearchForm.departureDate">
     <h:input_text id="arrivalDate"
       valueRef="FlightSearchForm.arrivalDate">
     <h:command_button id="submit" action="success" label="Submit"
       commandName="submit" />
     <h:command_button id="reset" action="reset" label="Reset"
       commandName="reset" />
     <s:errors/>
    </s:form>
</f:use_faces>
```

Obr. 2 Java Server Faces (JSF) při návrhu formuláře

Významný pokrok od jednoduchých šablon představuje nové rozhraní *Java Server Faces*, které umožňuje navrhovat uživatelské rozhraní webových aplikací jednoduše, bez znalosti programovacího jazyka – jak je vidět na ukázce stránky JSP psané za použití Java Server Faces, [3].

2.2 Zabezpečení

Řada webových rámců zjednodušuje zabezpečení webových aplikací. Standardní Java Servlet API a běžné javové webové kontejnery nabízí základní umisťování webových aplikací do tzv. *realms*, k nimž lze, chceme-li, přistupovat jen po autentizační výzvě a ověření uživatelského jména/hesla. Tvůrcům aplikací ale mnohdy nevyhovuje politika, kdy všichni uživatelé, kterým se přístup do aplikace povolí, musejí mít účet buďto na celém systému nebo alespoň v rámci javového webového kontejneru. To se pro servery s tisíci, často jen příležitostnými, uživateli nehodí. Proto rámce často nabízí snadnou rekonfigurovatelnost subsystému starajícího se o přihlašování uživatelů a jejich zabezpečený přístup. Na výpisu je vidět příklad

```
<!DOCTYPE application SYSTEM
"http://www.actionframework.org/dtd/ActionServlet_0_94.dtd">
<application repository="/devel/ActionServlet/examples/LoginServlet/classes">
   <templates>
      <template name="Login.wm" is-new-session="true"/>
   </templates>
   <components>
      <component name="Authenticator" class="Authenticator"
persistence="session">
       <action name="/login"
               method="login(String userName, String password)">
            <output-variable name="loginOK"
                             component="Authenticator" value="isLoggedIn()"/>
            <on-return value="void" show-template="SuccessfulLogin.wm"/>
            <on-exception class="LoginException" show-template="Login.wm"/>
         </action>
      </component>
   </components>
</application>
```

Obr. 3 Konfigurace řízení přístupu (ActionServlet)

konfigurace rámce *ActionServlet*, [4].

Pokud jde o zabezpečení třeba i proti chybě programátora, moderní javové kontejnery (např. Tomcat řady 5.x) dokáží využít možností *Java Security Manageru*, což je mocná součást javového běhového prostředí určená k zabezpečení javových aplikací. Pak přestává být problémem i hosting cizích (nedůvěryhodných) aplikací, neboť je zajištěno, že jim lze jemně a spolehlivě nastavit přístupová práva k hostujícímu systému.

```
The permission granted to your JDBC driver
grant codeBase
      "jar:file:${catalina.home}/webapps/examples/WEB-INF/lib/driver.jar!/-" {
    permission java.net.SocketPermission "dbhost.mycompany.com:5432", connect";
};
…
// These permissions apply to the container's core code, plus any additional
// libraries installed in the "server" directory
grant codeBase "file:${catalina.home}/server/-" {
        permission java.security.AllPermission;
};
…
```

Obr. 4 Definice bezpečnostních pravidel (Java Security Manager/Tomcat)

Na obr. 4. jsou vidět zápisy pravidel bezpečnostní politiky webového kontejneru Tomcat 5.

2.3 Kontrola uživatelských vstupů

Webové aplikace doposud komunikují s uživatelem především prostřednictvím klasického webového rozhraní – výstupy programu se zobrazují jako (X)HTML stránky obvykle opatřené obslužnými skripty. Uživatelské vstupy mohou pocházet z URL *odkazů* obsažených na stránce prezentované uživateli – aplikace je obdrží, jakmile uživatel klikne na příslušný odkaz a prohlížeč požádá o načtení stránky. Dalším zdrojem vstupů jsou *formuláře* (HTML Forms) a jejich *vstupní prvky*. Zpracování uživatelských vstupů je pak opakovaně řešenou úlohou s velmi podobným zadáním:

- Ověřit, zda vstup přišel od *autentizovaného* uživatele a identifikovat, od *kterého* a v rámci které *relace* (session) – jeden uživatel může mít současně se stejnou aplikací otevřeno více relací.
- *Dekódovat vstupy* – ve HTTP protokol přenáší data zakódovaná nejrůznějšími způsoby – jako části URL (např. HTTP metoda GET), v těle dotazu (metoda POST). Zejména v případě zaslání dat v rámci URL je poměrně náročné zajistit korektní kódování/dekódování – zejména, zde-li o data se znaky mimo sadu US-ASCII. Webové rámce by aplikačního programátora měly od neproduktivního řešení těchto starostí maximálně odstínit.
- Zjistit, zda zaslaná data vyhovují jak po stránce základního *datového typu*, tak i *povoleného rozsahu* – prvek z daného výčtu, klíč do číselníku, celé a reálné číslo (v jistém rozsahu), řetězec (např. ještě vyhovující určitému regulárnímu výrazu), logická hodnota (kódovaná různě), evt. i hodnota složená (seznam hodnot).

O *dekódování* vstupů se na javové webové platformě do značné míry nemusíme starat – přímo Java Servlet API nabízí metody k získání hodnoty parametru – uživatelského vstupu – ať už přišel jakoukoli cestou. Stejně tak je možné prostřednictvím metod objektů Servlet API identifikovat uživatele, s nímž se komunikuje, a příslušnou relaci (session).

```
<form-validation>
   <formset>
      <form name="addSubjectForm">
         <field property="subjID"
                depends="required" page="1">
             <arg0 key="admin.subject.missing.ID"/>
         </field>
         <field property="subjName"
                depends="required" page="1">
             <arg0 key="admin.subject.missing.name"/>
         </field>
         <field property="groupID"
                depends="required" page="2">
             <arg0 key="admin.subject.missing.groupID"/>
         </field>
      </form>
      <form name="addTaskForm">…
```

Obr. 5 Průvodce vyplňováním formuláře (Struts)

Webové rámce se pak postarají a následné zpracování vstupů: validaci vstupu provádí většina rámců, a to v zásadě dvěma způsoby:

- Deklarativní specifikací požadavků na vstup (datový typ, rozsah, povolené hodnoty, formát…) jednoduchý, čitelný a rozšiřitelný způsob. Rámec podle specifikace sám hodnoty vstupů ověří.
- Procedurální specifikací požadavků na vstup: programátor musí obvykle napsat třídu implementující určité rozhraní předepisující metodu, jež se volá pro ověření vstupní údaj. Metoda např. vrací *true*, je-li vstup v pořádku.

2.4 Udržování informací o relaci

Silným trendem až módou se v uživatelských rozhraních desktopových aplikací posledních let staly tzv. *průvodci* (wizards). Umožňují postupovat krok po kroku určitým procesem komunikace s aplikací, vyplňovat formuláře s možností návratu atd. V posledních letech uživatelé totéž vyžadují i po aplikacích webových. Tam je však tradičně významnou technologickou překážkou bezestavovost HTTP komunikace. Základní Servlet API k překlenutí tohoto omezení nabízí objekty třídy HttpSession, v nichž lze ukládat informace patřící dané relaci s uživatelem. Takto uložené informace „přežijí" čas mezi odesláním jednotlivých stránek (formulářů). Webové rámce umožňují deklarativním způsobem specifikovat, které objekty (obvykle psané jako *JavaBeans*) mají takto fungovat.

2.5 Řízení toku aplikace

Rozsáhlejší webové aplikace mívají často velmi složitou navigační strukturu. Jednotlivé případy užití (Use Cases) představují různé posloupnosti stránek prezentovaných uživateli, závislých samozřejmě také na uživatelských vstupech. Řízení aplikace přísluší v modelu MVC komponentě *Controller*. Webové rámce nabízí obvykle možnost deklarativně specifikovat tok řízení a tak Controller instruovat, kam relaci s uživatelem směřovat. Uvedený výsek konfiguračního kódu naznačuje, jak vypadá specifikace řízení toku v rámci *Struts*, [5].

```
<!-- Action Mapping Definition -->
<action-mappings>
  <!-- List Flights action -->
  <action path="/listFlights"
     type="foo.bar.FlightSearchAction"
     name="FlightSearchForm"
     scope="request"
     input="/faces/FlightSearch.jsp">
     <forward name="success" path="/faces/FlightList.jsp"/>
  </action>
</action-mappings>
```

Obr. 6 Popis řízení toku (Struts)

2.6 Zotavení z chyb a záznamy o událostech

Speciálním případech řízení toku je ošetřování chyb způsobených buďto nevhodnými uživatelskými akcemi (špatné zadání vstupu, požadavek na nepodporované URL, pokus o porušení zabezpečení) nebo systémovými chybami bez přičinění uživatele. Na rozdíl od řízení „hlavního" toku jsou chyby do značné míry asynchronní, nelze často předvídat místo jejich vzniku.

Javové Servlet API nabízí u stránek JSP deklarativní specifikaci toho, co se má uživateli zobrazit, nastane-li chyba. Server potom při vzniku chyby v aplikaci najde vyhovující pravidlo a vygeneruje stránku s použitím údajů o vzniku a příčině chyby.

Webové rámce tuto koncepci rozšiřují o nejrůznější možnosti automatického zaznamenávání těchto chyb podle jejich vážnosti (logging), jejich automatické posílání správci systému/aplikace mailem, ukládání do databáze, do souboru…

2.7 Podpora internacionalizace a lokalizace

Nutným atributem moderních aplikací, zejména těch provozovaných ve webovém prostředí, je podpora více *národních prostředí*. Adaptaci pro dané národní prostředí nazýváme *lokalizací* (localization, „l10n"), návrh aplikace vhodné k lokalizaci pak *internacionalizací* (internationalization, „i18n").

Jazyk Java je v tomto směru dobře vybaven, již na úrovni Java Core API je k dispozici třída java.util.Locale – objekt nese informace o určitém národním nastavení – *jazyku* (language), užívané *znakové sadě* (charset), *pravidlech lexikografického řazení* (collate), *symbolu měny*. Další třídy poskytují sofistikované možnosti formátovat podle národních zvyklostí *datum*, *čas*, *čísla* (vč. *měny*). Java rovněž nemá zásadní problémy s používáním všech běžných kódování znaků pro vstup a výstup textových informací; vnitřní reprezentace řetězců je založena na Unicode. Pro vstup/výstup lze použít běžná *osmibitová kódování* (pro češtinu ISO-8859-2, Windows-1250, dokonce i PC Latin 2 nebo KOI-8) a stejně dobře i kódování celé sady Unicode, jako jsou UTF-8, UTF-16.

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:if test="${lang==null}">
 <fmt:setBundle basename="com.heaton.bundles.Forum"
    var="lang" scope="session"/>
</c:if>
<c:if test="${param.lang!=null}">
 <fmt:setLocale value="${param.lang}"/>
 <fmt:setBundle basename="com.heaton.informit.I18NBundle"
    var="lang" scope="session"/>
 <c:redirect url="index.jsp"/>
</c:if>
<html><head><title>I18N Example</title></head>
<body>
    <h1><fmt:message key="login.pleaselogin" bundle="${lang}"/></h1>
    <form method=post action=main.jsp>
        <fmt:message key="login.uid" bundle="${lang}"/><input name=uid><br/>
        <fmt:message key="login.pwd" bundle="${lang}"/><input name=pwd><br/>
        <input type="submit" name="action" value="<fmt:message
        key="login.title" bundle="${login}"/>">
    </form>
    <h1><fmt:message key="login.language" bundle="${lang}"/></h1>
    <ul>
        <li><a href="index.jsp?lang=en">
        <fmt:message key="login.english" bundle="${lang}"/>(English)</li>
        <li><a href="index.jsp?lang=es">
        <fmt:message key="login.spanish" bundle="${lang}"/>(Spanish)</li>
    </ul>
</body></html>
```

Obr. 7 Internacionalizovaná aplikace (Java Standard Tag Library)

To je však jen nezbytný technický základ; pro konstrukci internacionalizované webové aplikace dále potřebuje nástroje umožňující snadnou lokalizaci všech textových informací používaných a generovaných programem (např. řetězcových literálů zde uvedených). K tomu Java nabízí třídu *ResourceBundle*, která umí podle explicitně nebo implicitně zadaných Locale vybrat příslušnou sadu řetězců pro daný národní jazyk. Ani to však neřeší vše – přinejmenším ne v celosvětovém prostoru. Tam musí programátor respektovat i další odlišnosti – např. uživatelské rozhraní bude vypadat jinak u jazyků, které se čtou zprava doleva nebo shora dolů.

Dosud uvedené vlastnosti nabízí přímo Java. Úkolem webových rámců je spíše *usnadnit a zrychlit využití* nabízených vlastností. Hlavním přínosem rámců je transparentnost – programátor se nemusí starat, jak získat příslušný balík (bundle) s řetězci v národním jazyce pro daného uživatele: rámec např. podle informací automaticky zaslaných prohlížečem ví, jaký jazyk uživatel očekává, a podle toho řetězce vybere. Výše uvedená stránka JSP je psaná s využitím přístupu k lokalizovaným verzím řetězců pomocí knihovny značek [6].

2.8 (Automatické) zajištění perzistence objektů

Snažíme-li se budovat aplikaci jako skutečně objektovou, tzn. nevyužíváme-li z objektového repertoáru jazyka jen komponenty uživatelského rozhraní – narazíme dříve či později na potřebu perzistentního ukládání entitních objektů, tj. objektů, které reprezentují entity z (datového) modelu aplikace. Jde o to, že aplikace musí být schopna v případě korektního, ale i neočekávaného ukončení (výpadek napájení, havárie operačního systému…), obnovit objektové struktury v paměti. Stejně tak je perzistentní uložení nezbytné i proto, že není možné mít veškeré entity spravované aplikací v jednu chvíli současně v paměti – je potřeba je odkládat.

Java v této oblasti sice nabízí možnost *serializace* a následné uložení objektů do souboru, rovněž tak bychom mohli ručně naprogramovat „rozklad“ a uložení objektů do relační databáze přes standardní databázové rozhraní (JDBC). To s sebou nese převážně rutinní, nezáživné programátorské úkony, v nichž lze napáchat mnoho chyb. Ve webových rámcích se proto stále častěji používají prostředky, které umožní do značné míry automatizovat a zprůhlednit proces převodu a uložení objektů do perzistentního úložiště (soubory, databáze) a jejich obnovu. Ukázka naznačuje, jak vypadá popisovač mapování mezi javovými objekty a jejich reprezentací v relační databázi pomocí prostředků balíku *Hibernate*, [7].

```
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
    <class name="test.hibernate.Order" table="orders">
        <id name="id" type="string" unsaved-value="null" >
            <column name="id" sql-type="char(32)" not-null="true"/>
            <generator class="uuid.hex"/>
        </id>
        <property name="date">
         <column name="order_date"
                 sql-type="datetime" not-null="true"/>
        </property>
        <property name="priceTotal">
        <column name="price_total"
                sql-type="double" not-null="true"/>
        </property>

        <set name="orderItems" table="order_items" inverse="true"  cascade="all">
            <key column="order_id" />
            <one-to-many class="test.hibernate.OrderItem" />
        </set>
</class>
</hibernate-mapping>
```

Obr. 8 Zajištění perzistence - mapování objektů na databázi (Hibernate)

2.9 Škálovatelnost a distribuovanost

Velmi zatížené aplikace nelze provozovat na jednom serveru, byť výkonném. Webové aplikace často pro rozložení zátěže *clustering*, rozběhnutí aplikace současně na svazku počítačů, jímž jsou klientské dotazy podle potřeby rovnoměrně distribuovány. S tím vzniká řada problémů, např. chceme-li uchovávat informace mezi následnými požadavky zaslanými během jedné relace s uživatelem. Pokročilé webové rámce řeší i tyto záležitosti, takto škálovatelné aplikace se většinou provozují na tzv. *aplikačních serverech*, což jsou rozsáhlé programové systémy (na úrovni složitosti operačního systému) poskytující komplexní middlewarovou infrastrukturu – s daleko širším záběrem než uvedené webové rámce.

Základní i pokročilé informace o aplikačních serverech a budování rozsáhlých podnikových aplikací nad nimi lze najít např. na webu *TheServerSide.com*.

3. VÝSTAVBA APLIKACÍ NAD WEBOVÝMI RÁMCI

3.1 Přehled dostupných rámců a řešení

V současnosti existuje více než *dvě desítky* zdarma dostupných webových rámců pro platformu Java, na mnohých jsou postavena i velká webová sídla. Odkazy lze najít v seznamu literatury, většina rámců je zachycena v souhrnném zdroji [8] anebo je lze vyhledat přímo na místě, kde je řada z nich hostována – na serveru Sourceforge.net, [9].

3.2 Podpora vývojovými nástroji

Výhodnost rámců se ještě zvýrazní, mají-li podporu u vývojových nástrojů. Ze známých uveďme např. *Eclipse* (+Struts Studio) a *Borland JBuilder* (+InternetBeans Express).

LITERATURA

[1] Java Servlet Technology, http://java.sun.com/products/servlet/

[2] Apache Jakarta Velocity Template Engine, http://jakarta.apache.org/velocity/

[3] Java Server Faces, http://java.sun.com/j2ee/javaserverfaces/index.jsp

[4] ActionServlet - Action-driven component-based opensource web application framework, http://www.actionframework.org/

[5] The Apache Struts Web Application Framework, http://jakarta.apache.org/struts/

[6] JavaServer Pages Standard Tag Library, http://java.sun.com/products/jsp/jstl/

[7] Hibernate - Relational Persistence For Idiomatic Java, http://hibernate.org

[8] Wafer - Web Application Framework Research project, http://www.waferproject.org

[9] Sourceforge.net, http://sf.net