Tvorba pokročilých webových aplikací v Javě

Část 1. - Úvod, srovnání, Servlety, JSP, JDBC

Lukáš Fryč, Martin Večeřa, Ondřej Žižka

Říjen 2010

redhat

Úvod: Kdo, co, proč, komu, jak

- Co?
  - Úvod do webové Javy
  - Úvod do “enterprise” Javy (Java Enterprise Edition, Java EE)
  - Rozsah, obsah, podoba
  - Zápočet, zkouška, kredity
- Proč?
  - Vyspělost a zpětná kompatibilita
  - Stabilita a škálovatelnost
  - Standardizace
- Komu?
  - Znalost jazyka Java, HTML, jakž takž HTTP
- Jak?
  - Méně přednášení, více kódování ;-)
  - Dva projekty: demonstrační + vlastní

redhat

Porovnání PHP a Java EE

- PHP: scriptovací jazyk
- Java EE: development framework
- Tyto dvě technologie se tedy nedají porovnat
  - Spíše PHP vs. JSP

redhat

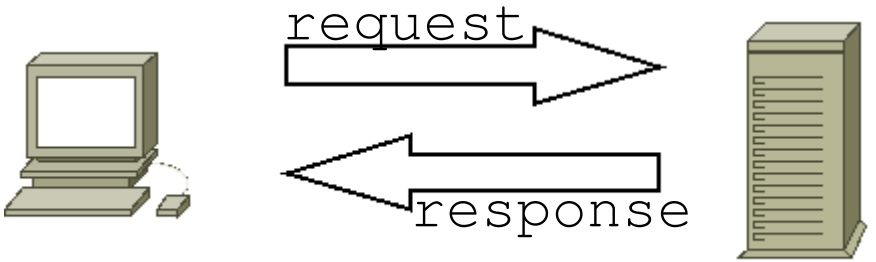
- **PHP**
  + dobrý pro rychlé vytvoření malé a funkční aplikace
  + stačí i domácí hardware
  + free webhosting
  - vede k míchání konceptů (model, pohled)
  - žádný koncept či vize
  - nekompatibilita verzí

- **Java EE**
  + výkonnost (JIT compiler) a škálovatelnost (+ failover)
  + podpora transakčního zpracování
  + podpora pokročilých technologií (messaging, clustering, bezpečnost)
  + standardizované API (JDBC, JMS)
  + napojení na legacy systémy (JCA)
  - pro malé aplikace je to kanón na vrabce
  - pomalá učicí křivka

redhat

HTTP

- Bezestavový protokol
  - Požadavek je ihned zodpovězen a “zapomenut”

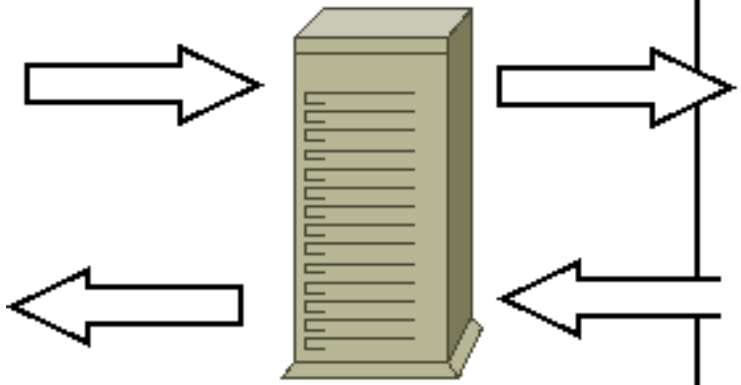


  - Původně určen pro statický obsah
  - Později použit i pro dynamicky generovaný obsah

redhat

Servlety – obsah generovaný v Javě (1997)

- Servlet: Třída implementující rozhraní `HttpServlet`
- Server přijme HTTP požadavek, zavolá callback metodu,
- ta vrátí obsah, případně ještě pozmění HTTP metadata



KalkulackaServlet.java

```
public class ObraceckaServlet extends HttpServlet {

    @Override
    protected void doGet( HttpServletRequest request,
                          HttpServletResponse response )
    throws ServletException, IOException
    {
        response.setContentType("text/plain;charset=UTF-8");
        PrintWriter out = response.getWriter();

        try {
            out.println("Obracečka 1.0");
            String res = reverse( request.getParameter("text") );
            out.println("Výsledek: " + res );
        } finally {
            out.close();
        }
    }
    // ...
```

redhat

Servlety – obsah generovaný v Javě

- Servlet: Třída implementující rozhraní `HttpServlet`
- Server přijme HTTP požadavek, zavolá callback metodu,
- ta vrátí obsah, případně ještě pozmění HTTP metadata

```
public class KalkulackaServlet extends HttpServlet {

    @Override
    protected void doGet( HttpServletRequest request, HttpServletResponse response )
    throws ServletException, IOException {
        response.setContentType("text/plain;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("Obracečka 1.0");
            String obracene = reverse( request.getParameter("text") );
            out.println("Výsledek: " + obracene );
        } finally {
            out.close();
        }
    }
```

- jako “vedlejší efekt” může provést cokoliv
  - SQL dotazy
  - Komunikace s dalšími servery
  - ...

redhat

Struktura Java EE web aplikace

- index.jsp
- WEB-INF
  - lib
    - myWebAppClasses.jar
  - classes
    - *.class
  - **web.xml ← konfigurace aplikace**

redhat

web.xml – konfigurace a navázání servletů na URL cestu

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <servlet>
        <servlet-name>ObraceckaServlet</servlet-name>
        <servlet-class>org.jboss.course.jsp.ObraceckaServlet</servlet-cla
    </servlet>
    <servlet-mapping>
        <servlet-name>ObraceckaServlet</servlet-name>
        <url-pattern>/ObraceckaServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

redhat

Servlety – cvičení 1

- Vytvoření jednoduchého servletu
  - Vytvořte servlet Kalkulačka, který bez zadání parametrů zobrazí formulář se dvěma textovými poli a výběrem operací plus, mínus, krát, děleno.
  - Při odeslání formuláře zobrazí servlet výsledek dané operace se zadanými operandy.
  - Můžete realizovat jako dva různé servlety.
- Rozšíření servletu
  - Do formuláře zabudujte paměť dříve naposledy zadaných hodnot. Nápověda: Použijte session scope.
  - Ve výše uvedeném servletu přidejte ukládání provedených operací a vytvořte servlet, který tuto historii vypíše. Nápověda: Použijte application scope.
  - Pro jednoduchost může být výstup textový.
- Kdo má hotovo, může vytvořit servlet zobrazující počet aktivních session. Nápověda: `HttpSessionListener`, a `<listener>` ve `web.xml`

redhat

Servlety – pojmy

- Servlet Context
  - Označuje webovou aplikaci jako celek, tj.
    - souhrn částí (servletů, JSP stránek a dalších),
    - konfiguraci – web.xml, logickou cestu (za hostname),
    - stav aplikace – všechny atributy ve všech scopech (viz dále)

- Servlet pooling (thread-safety)
- Listeners:
  - `HttpSessionListener: sessionCreated(), ~Destroyed()`
  - `ServletContextListener: contextInitialized(), ~Destroyed()`
  - Attribute listeners – `atributeAdded(), ~Removed(), ~Replaced()`
  - Binding listeners – `valueBound(), valueUnbound()`

redhat

Servlet filter, filter chain

- Před a po servletu může požadavek zpracovat filtr.
- Filtry je možno do sebe vnořovat (chain of responsibility)
- Konfiguruje se ve `WEB-INF/web.xml`

```
<filter>
    <filter-name>BmpToPngFilter</filter-name>
    <filter-class>org.jboss.test.BmpToPngFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>BmpToPngFilter</filter-name>
    <url-pattern>*.bmp</url-pattern>
</filter-mapping>

<servlet>
    <servlet-name>KalkulackaServlet</servlet-name>
    <servlet-class>org.jboss.course.jsp.ObraceckaServlet</servle
    <init-param>
        <param-name>presnost</param-name>
        <param-value>3</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>KalkulackaServlet</servlet-name>
    <url-pattern>/KalkulackaServlet</url-pattern>
</servlet-mapping>
```

redhat

Servlety – přesměrování, RequestDispatcher

- Často je nutné předávat request mezi částmi aplikace
  - např. podle MVC: servlet controller -> JSP view

```
doGet( HttpServletRequest req, HttpServletResponse res){
  // Logická cesta relativní k aktuální cestě.
  RequestDispatcher rd =
    request.getRequestDispatcher("vypis.jsp");
  // Nebo přes servlet context - absolutní log. Cesta.
  rd = getServletContext().
         getRequestDispatcher("/admin/vypis.jsp");
  rd.forward( req, res );
}
```

redhat

Servlety – cvičení 2

- Kapku složitější servlety – malá webová aplikace
  - Vytvořte jednoduchý webový aukční server se čtyřmi stránkami:
    - Úvodní stránku s nabídkou akcí “vložit nabídku” a “zobrazit aukce”,
    - Formulář pro vložení nové nabídky,
    - Stránku vypisující probíhající i ukončené aukce,
    - Stránku zobrazující vybranou aukci.
  - Stránky vhodně prolinkujte.
  - Jako úložiště dat použijte List<AukceBean> v *application scope*.
  - Zatím není nutné vytvářet přihlašování ani zabezpečení.
  - Pokud se na to cítíte, můžete zkusit identifikovat, zda je ta která aukce zadaná aktuálním uživatelem. Nápověda: z objektu `session` lze získat session ID.

redhat

Servlety – nevýhody

- pro textový výstup nepraktické,
- nutná kompilace při každé změně,
- může vést k nehezkému kódu...

```
out.println("<html>" +
   "<head><title>" +
   book.getName()
   + "</title></head>");
```

```
out.println("<div id=\"user\">");
out.println("<div id=\"login\">");
out.println("<form action=\"login\" ...>");
out.println("<input type=\"text\" ... />");
out.println("</form>");
out.println("</div>");
out.println("</div>");

out.println("<h1>"+book.getName()+"</h1>");
out.println("<h2>k dostání zde v e-knihkupectví</h2>");
out.println("<div>");
out.println("<p>Autor:" + book.getAuthor() + "</p>" );
out.println("<p>Rok:" + book.getYear() + "</p>" );
out.println("<p>Cena:" + book.getPrice() + "</p>" );
out.println("<p>Popis: ");
out.println(book.getDesc() );
out.println("</p>");
out.println("</div>");

out.println("</body></html>");
out.close();
```

- ...a proto...

redhat

JSP – Java Server Pages (1999)

- Obrácený zápis – kód v textu
- Obdobně jako v PHP či ASP
  - `<% ... %>, <%= ... %>, <%@ ... %>, <%! ... %>`

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP stránka</title>
    </head>
    <body>
        <h1>Ahoj lidi!</h1>
        <div>Aktuální čas: <%= new java.util.Date() %></div>
    </body>
</html>
```

redhat

JSP – Java Server Pages

- Obrácený zápis – kód v textu
- Obdobně jako v PHP či ASP
  - `<% ... %>`, `<%= ... %>`, `<%@ ... %>`, `<%! ... %>`
  - ukázka na wiki
- Navíc:
  - Stručné vkládání hodnot - `${...}`
    - `<div><em>Autor:</em> ${book.author}</div>`
  - JSP tagy
    - podobně jako v ColdFusion CFML
    - komponenty – např. `<jsp:plugin>`
    - logika, např. `<c:if>...</c:if>`, `<c:forEach>` atd.
    - viz dále

JSP – Princip

- Jednoduchý:
  - .jsp soubor je přeložen do .java zdrojáku servletu
  - s .java souborem se pak už zachází jako se servletem.

Input JSP

```
<%@ page errorPage="myerror.jsp" %>
<%@ page import="com.foo.bar" %>

<html>
<head>
<%! int serverInstanceVariable = 1;%>

<% int localStackBasedVariable = 1; %>
<table>
<tr><td><%= toStringOrBlank( "expanded inline data " + 1 ) %></td></tr>
```

Resulting servlet

```
package jsp_servlet;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

import com.foo.bar; // Imported as a result of <%@ page import="com.foo.bar" %>
import …

class _myservlet implements javax.servlet.Servlet, javax.servlet.jsp.HttpJspPage {
   // Inserted as a
   // result of <%! int serverInstanceVariable = 1;%>
   int serverInstanceVariable = 1;
   …

   public void _jspService( javax.servlet.http.HttpServletRequest request,
   javax.servlet.http.HttpServletResponse response )
   throws javax.servlet.ServletException,
   java.io.IOException
```

redhat

JSP

Input JSP

```
<%@ page errorPage="myerror.jsp" %>
<%@ page import="com.foo.bar" %>

<html>
<head>
<%! int serverInstanceVariable = 1;%>

<% int localStackBasedVariable = 1; %>
<table>
<tr><td><%= toStringOrBlank( "expanded inline data " + 1 ) %></td></tr>
```

Resulting servlet

```
package jsp_servlet;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

import com.foo.bar; // Imported as a result of <%@ page import="com.foo.bar" %>
import …

class _myservlet implements javax.servlet.Servlet, javax.servlet.jsp.HttpJspPage {
   // Inserted as a
   // result of <%! int serverInstanceVariable = 1;%>
   int serverInstanceVariable = 1;
   …

   public void _jspService( javax.servlet.http.HttpServletRequest request,
   javax.servlet.http.HttpServletResponse response )
   throws javax.servlet.ServletException,
   java.io.IOException
   {
       javax.servlet.ServletConfig config = …; // Get the servlet config
       Object page = this;
       PageContext pageContext = …;            // Get the page context for this request
       javax.servlet.jsp.JspWriter out = pageContext.getOut();
       HttpSession session = request.getSession( true );
       try {
           out.print( "<html>\r\n" );
           out.print( "<head>\r\n" );
           …
           // From <% int localStackBasedVariable = 1; %>
           int localStackBasedVariable = 1;
           …
           out.print( "<table>\r\n" );
           out.print( " <tr><td>" );
           // From <%= toStringOrBlank( "expanded inline data " + 1 ) %>
           out.print( toStringOrBlank( "expanded inline data " + 1 ) );
           out.print( " </td></tr>\r\n" );
           …
       } catch ( Exception _exception ) {
           // Clean up and redirect to error page in <%@ page errorPage="myerror.jsp" %>
       }
   }
}
```

redhat

JSP – Syntaxe

- `<% ... %>` Je zkopírováno jako kód v metodě `doGet()`.*
- `<%= ... %>` Musí obsahovat výraz; ten je vyhodnocen a předán do `out.print()`.
- `<%@ ... %>` Parametry stránky, taglibs, importy a includy – viz ukázka.
- `<%! ... %>` Deklarace proměnných třídy.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.*" %>
<%@page session="true" %>
<%@taglib uri="http://www.brno.cz/jsptags" prefix="brno" %>

<%! int pocitadlo = 0; %> <%-- Pozor na vlákna - nefunguje tak, jak byste čekali. --%>

<html>
    <head>
        <%@include file="header-meta.html" %>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP stránka</title>
    </head>
    <body>
        <div>Aktuální čas: <%= new java.util.Date() %></div>
        <div>Počítadlo:    <%= ++pocitadlo %></div>
        <%-- JSP komentář - tento text nebude ve výstupu. --%>

        <brno:novinky>
          ...
        </brno:novinky>
</body>
</html>
```

redhat

JSP – Scopes

- V JSP je možné manipulovat s objekty.
- Mohou existovat v “úložištích” ve 4 úrovních:
  - **page scope**: Vytvoří a odstraní se pro jednotlivé stránky.
    - Používaný nepřímo – JSP tagy (viz dále)
  - **request scope**: Existuje po dobu vyřizování HTTP požadavku.
    - Pro předávání hodnot v MVC modelu mezi controller servletem a JSP view
  - **session scope**: Existuje po dobu existence session.
    - Pro data jako přihlášený uživatel, obsah košíku, atd.
  - **application scope**: Existuje po celou dobu běhu webu (“singleton”).
    - Globální objekty a data – cache, počet přihlášených, DB připojení...
- thread-safety

redhat

Servlety vs. JSP – práce s atributy

- **page scope**:
  - N/A — *pageContext*.setAttribute()
- **request scope**:
  - *request*.setAttribute() — *request*.setAttribute()
- **session scope**:
  - *request*.getSession().setAttribute() — *session*.setAttribute()
- **application scope**:
  - getServletContext().setAttribute() — *application*.setAttribute()

redhat

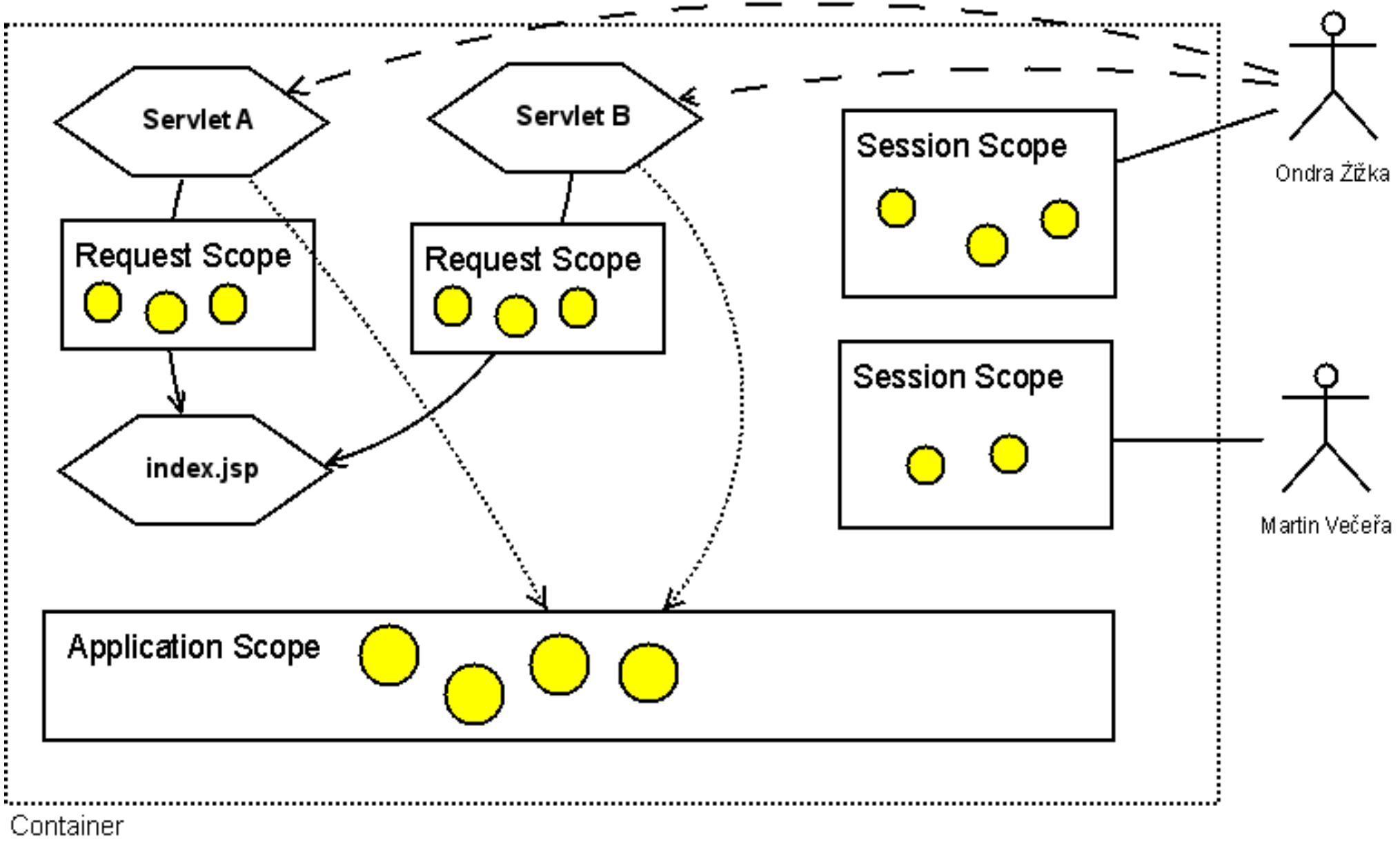
JSP – Beans & scopes

- V JSP stránce je možné objekty (aka beany) vytvořit takto:
  - `<jsp:useBean id="aukceBean" scope="request" class="org.jboss...AukceBean">`
- Poté je možno objekt používat:
  - v Java kódu
    - `<h1><%= aukceBean.getNazev() %></h1>`
  - v EL výrazech
    - `<h1>${aukceBean.nazev}</h1>`
  - v atributech JSP tagů, ...
    - `<c:if test=”aukceBean.uzavrena”>`

redhat

JSP – Beans & scopes – thread safety

Thread safety



redhat

JSP – Předdefinované objekty

- U objektů je uveden jejich scope.
- Některé objekty či jejich obdoby již znáte ze servletů – u těch je *.

- `request* [req]` – viz `HttpServletRequest`
- `response* [page]` – viz `HttpServletResponse`
- `out* [page]` – writer pro výstup, stejné jako u servletů.
- `config* [page]` – pro získání init parametrů aplikace.
- `page` – instance třídy stránky zpracovávající požadavek. Moc se nepoužívá.
- `exception` – definováno jen pro chybové stránky.

redhat

JSP – Předdefinované objekty II

- U objektů je uveden jejich scope.
- Některé objekty či jejich obdoby již znáte ze servletů – u těch je *.

- `pageContext [page]`
  - Úložiště pro objekty používané stránkou
    - `setAttribute(), getAttribute(), findAttribute(), removeAttribute(), ...`
  - Zkratkové metody [`getOut(), getException(), getPage() getRequest(), getResponse(), getSession(), getServletConfig()  a getServletContext()`].
- `session* [sess]` – přístup k objektu `HttpSession`
- `application* [app]` – kontext aplikace.
  - Stejné jako volání `getServletConfig().getContext()` v servletu.

redhat

JSP – <%@page ... %>

- extends="className" – od které třídy bude tento servlet dědit.
- import="importList" – defaultně: `java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*`
- session="true|false" – podílí se / potřebuje tato stránka session?
- buffer="none|*N*kb" – velikost výstupního bufferu.
- autoFlush="true|false" – automatické vysypání bufferu.
- isThreadSafe="true|false" – je stránka schopná paralelního zpracování?
- info="*popis*" – užívané v různých admin/devel aplikacích.
- errorPage="*url*" – url chybové stránky (pokud by nastala výjimka).
- isErrorPage="true|false" – jedná se o stránku pro ošetření chyb?
- contentType="..." – defaultně "text/html".
- PageEncoding="..." – defaultně "ISO-8859-1".

redhat

Expression Language (EL)

- Stručnější vypisování hodnot vlastností bean
  - Místo `<p><%=aCustomer.getAddress().getCountry()%><p>`
    píšete `<p>${aCustomer.address.country}</p>`

- Bezpečnější – omezený přístup k objektům
- Deffered evaluation - #{...}
  - umožňuje i zápis hodnoty
  - některé knihovny tagů (JSF) jej používají
  - možno volat `public` ne-`void` metody

redhat

Expression Language - možnosti

- “Traverzování” po vlastnostech: `${aCustomer.address.country}`
- Přístup do `Map`y: `${colors[“red”]}`
- Pole[ ] nebo `List`: `${customer.orders[1]}`
- Výrazy:
  - `${customer.age + 20.5 / 4 >= 50 }`
  - `${customer.age == 20}`
- Seznam operátorů a další možnosti viz Java EE tutoriál.
- Možno použít předdefinované objekty (viz dříve):
  - `${param['item_id']}`
  - `${sessionScope.cart.numberOfItems % 2}`

redhat

JSP tagy

- JSP podporuje tagy ve stylu ColdFusion CFML
- XML syntaxe
- Podpora Expression Language - `${...}`

```
<%@ taglib prefix="c"
  uri="http://java.sun.com/jsp/jstl/core" %>
...
<c:set var="browser" value="${header['User-Agent']}"/>
<c:out value="${browser}"/>
```

- Standardní knihovna JSTL – JSP Standard Tag Library (dále)
- + mnoho knihoven (gůglete “jsp tag library”)
- Částečně oddělují logiku od prezenční vrstvy
- Umožňují přesun Java kódu do business vrstvy
- Možnost jednoduše naprogramovat vlastní tagy - “komponenty”

redhat

JSP – předdefinované tagy

- `<jsp:forward page="relativní-URL">`
  - přesměruje na html/jsp/servlet
  - ukončí se vykonávání aktuální JSP stránky.
- `<jsp:include page="url" flush="true|false">`
  `<jsp:param .../> </jsp:include>`
  - Začlení obsah z dané URL.
  - Provede se extra požadavek -- může být neefektivní.
- `<jsp:useBean>` - definice objektu, viz dříve
- `<jsp:getProperty name="bean" property="propertyName">`
  - vypíše danou vlastnost daného objektu.
- `<jsp:setProperty name="bean" property="prop" param="paramName" />`
  - nastaví danou vlastnost daného objektu podle parametru požadavku.
  - Trik: `<jsp:setProperty name="beanName" property="*" />`
    - Vyhledá parametry podle jmen vlastností beany.
- `<jsp:setProperty name="bean" property="prop" value="<%=1+1%>" />`
  - Nastaví vlastnost na danou hodnotu.

redhat

JSP – předdefinované tagy II

- `<jsp:attribute>`
- `<jsp:element>`
- `<jsp:body>`
- `<jsp:invoke>`
- `<jsp:doBody>`
- `<jsp:text>`
- `<jsp:output>`
- ... používané pro vlastní tagy.

redhat

JPS - includování

- Dvě možnosti:
  - <%@ include file=”header.jsp” %>
    - V podstatě vložení JSP kódu z daného souboru.
    - Neplést s <%@ page include %>.
    - Provádí staticky se při kompilaci!
  - <jsp:include page=”header.jsp” />
    - Provedení kódu z daného souboru.
    - Provádí se dynamicky při zpracování každého požadavku!
    - Vnitřně funguje přes `RequestDispatcher#include()`.
  - Ještě existuje tag <c:import>, který umí načíst i z externí URL.

redhat

JSP Standard Tag Library – JSTL

- Standardní knihovna tagů
  - Core – variables, flow control, `<c:url>`, `<c:out>`
  - XML – utils, transformation (XSLT)
  - I18N – formátování zpráv, čísel a dat, lokalizace
  - SQL
  - Funkce – manipulace se stringy, délky kolekcí

redhat

JSTL - Core

- `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>`

- Variables

```
<c:set var="bookId" value="${param.Remove}"/>
<c:remove var="cart" scope="session"/>
```

- Flow control

```
<c:forEach var="item" items="${sessionScope.items}"> ...
<c:if test="${!empty param.Add}"> ...  </c:if>
<c:choose>
  <c:when test="${cust.category=='trial'}"> ...
  <c:when test="${cust.category=='paid'}"> ...
</c:choose>
```

- `<c:out>` - oproti ${...} umí číst z `Reader`u, je možné ovládat `escapeXml`
- `<c:catch>` - zachytává výjimky

redhat

JSTL - Internationalization

- `<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>`

- `<fmt:bundle> ... </fmt:bundle>`
- `<fmt:setBundle var=”bundle1” basename=”str.properties” />`
- `<fmt:message />`

- Lokalizace bude probrána v dalších hodinách
  - Stačí si zapomatovat, že v JSTL je podpora.

redhat

JSTL – SQL

```
<sql:transaction>
  <sql:query var="books" sql="SELECT * FROM aukce">
    <sql:param value="${bookId}" />
  </sql:query>
  <c:forEach var="bookRow" begin="0" items="${books.rowsByIndex}">
     ...
  </c:forEach>
</sql:transaction>
```

- Opět – stačí vědět, že cosi takového v JSTL je.
  - Dnes už jsou mnohem lepší způsoby práce s databází.

redhat

JSP – Podružnosti

- Umožňuje alternativní zápis v XML-validních souborech - syntaktické konstrukce mají XML alternativy:
  - `<% ... %>` `<jsp:scriptlet>...</jsp:scriptlet>`
  - `<%= ... %>` `<jsp:expression>...</jsp:expression>`
  - `<%@ ... %>` `<jsp:declaration>...`
- XML zápis: validovatelnost vs. stručnost

- Escapování:
  - Pro zapsání `%>` se píše `%\>`
  - Pro zapsání `<%` se píše `<\%`
  - Znaky `'"\` se escapují jako `\' \" \\` (jen v atributech)

redhat

JSP – cvičení 1

- Zazálohujte si předchozí stav projektu Aukce.
- Přepište servlety, které převážně vypisují HTML, do JSP.
- Pokud jste v některých servletech zpracovávali požadavek a zároveň vypisovali výstup, je na čase tyto části oddělit.
  - Tip:
    - Můžete zpracovat požadavek v servletu,
    - hodnoty potřebné k zobrazení uložit do *request scope*,
    - použít přesměrování ze servletu na JSP stránku,
    - a v ní vypsat připravené hodnoty.
  - Druhá možnost je přesunout zpracování do speciálního objektu – request scope beany, které zavoláte vhodnou metodu v JSP stránce.
  - Pokročilejší možností je přetížit základní třídu JSP stránky.

redhat

JSP – cvičení 2

- K aktuálnímu stavu aplikace přidejte jednoduché přihlašování.
  - Vytvořte JSP soubor, ve kterém bude s přihlašovacím formulářem. Tento soubor potom includujte ve všech stránkách aplikace.
  - Jednodušší alternativou je vytvořit přihlašovací stránku a na ni ve všech stránkách vytvořit odkaz.
  - Po přihlášení pochopitelně zobrazujte odkaz “odhlásit”; ověřte jeho funkčnost.
  - Je možné využít standardní Java EE metody. Druhou možností je použít session.

redhat

JSP – výhody oproti servletům

- Snazší psaní, čitelnější
- Není třeba kompilovat – stačí .jsp soubor vložit kamkoliv do adresáře aplikace a je připravený k použití.

JSP – nevýhody

- Oddělení prezenční a business vrstvy je pracné.
- Někdy obskurní řešení webových problémů (přesměrování, opětovné POST požadavky, historie procházení).
- Bez dobrého IDE se těžko debuguje.

redhat

JDBC – Java Database Connectivity

- Standardní Java API pro komunikaci s databázemi.
- Umožnuje dotazování a manipulaci s daty pomocí SQL.
- Podporuje prepared statements.
- Všechny běžné databáze mají implementaci JDBC.
- Kroky práce s DB – nic nečekaného:
  - Získat připojení (nemusí znamenat otevřít nové spojení)
  - Zahájit transakci
    - Připravit a provést dotazy
    - Přečíst vrácená data
  - Uzavřít transakci
  - Uvolnit spojení (opět, nemusí znamenat zavřít)
- Chování JDBC-based aplikací silně záleží na implementaci DBMS.

redhat

JDBC – Ukázka

```
// "Connect" to the db.
Class.forName("org.hsqldb.jdbc.JDBCDriver");
Connection conn = DriverManager.getConnection("jdbc:hsqldb:file:hsqldb/dat


List<Aukce> aukce = new ArrayList();

// Perform the user-provided SQL statement.
String sql = "SELECT nazev, zadal, cena, popis, vytvoreno FROM aukce ORDER
PreparedStatement ps = conn.prepareStatement( sql );
ResultSet rs = ps.executeQuery();

while( rs.next() ){
  aukce.add( new Aukce(
       rs.getString("nazev")
      ,rs.getString("zadal")
      ,rs.getString("popis")
      ,rs.getDate("vytvoreno")
      ,rs.getInt("cena")
  ) );

}
```

redhat

JDBC - cvičení

- Vaše aplikace momentálně používá data zapsaná jednoduše ve zdrojovém kódu.
- Podle příkladu uvedeného výše zkuste aplikaci předělat tak, aby data načítala a ukládala do databáze.
- Nedoporučujeme použít JSTL.
- Nápověda: Jde skutečně o jednoduché zasílání SQL do DB.
  - SQL nelepte ze stringů, použijte “prepared statement”.
  - `SELECT --> conn.executeQuery()`
  - `INSERT, UPDATE, DELETE --> conn.executeUpdate()`
- Odkaz na dokumentaci JDBC naleznete na Jboss.org stránce JEE

ozizka@redhat.com | www.redhat.com