



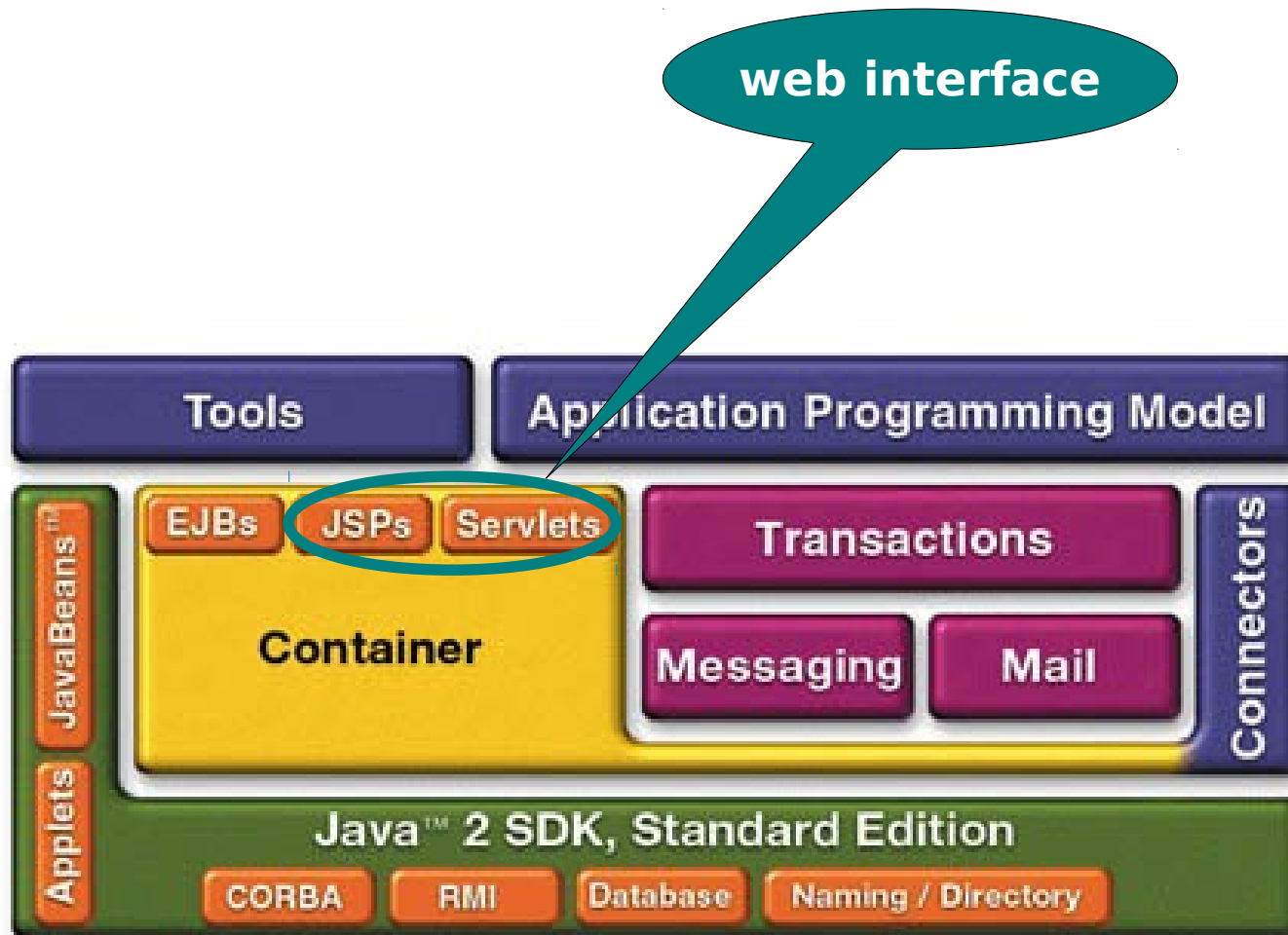
**JSP**

Petr Aubrecht (CA)

unzip; strip; touch; finger; mount; fsck; more; yes; unmount; sleep

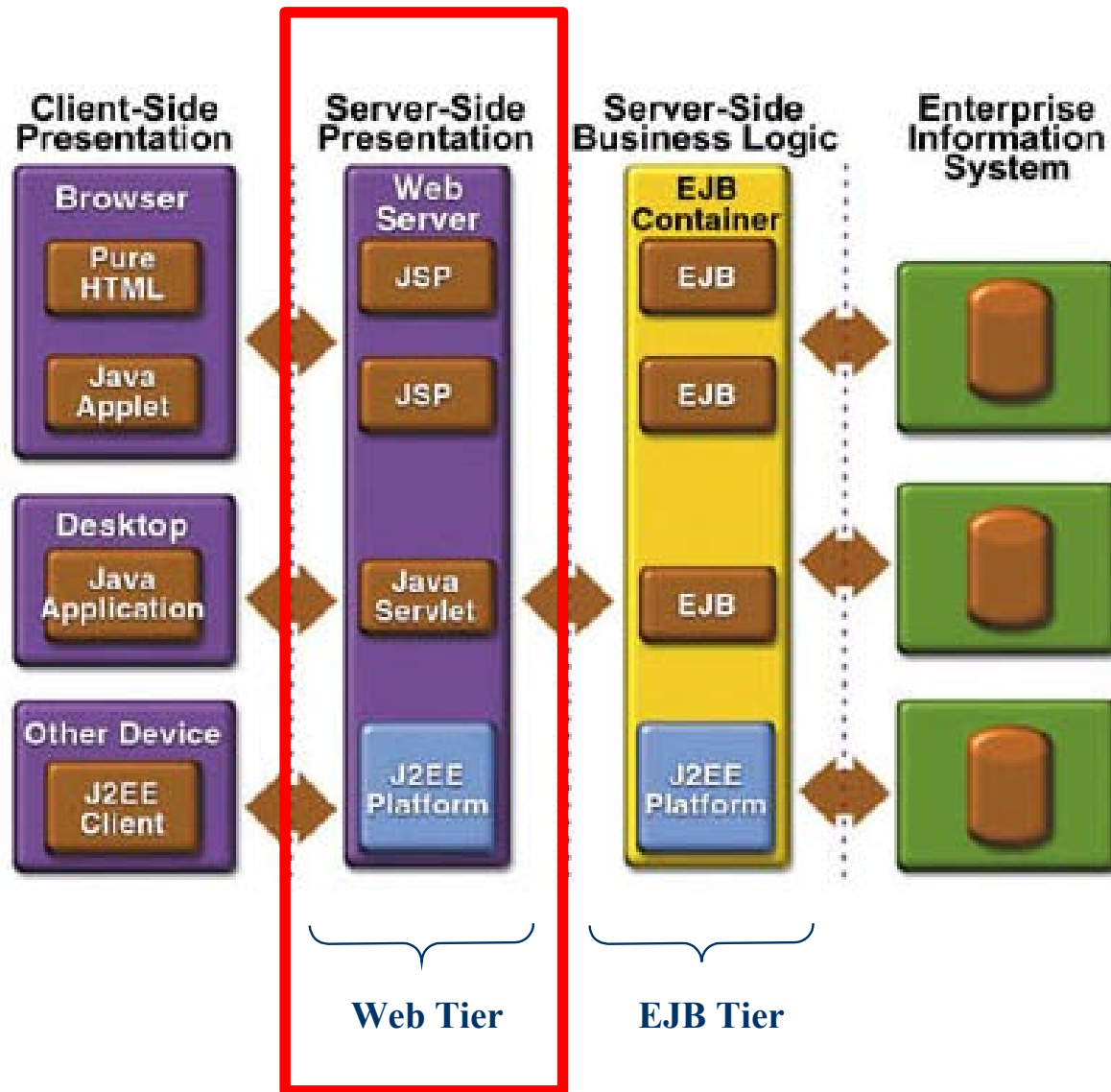
# Servlets and JSP

- kontext:



# Process Flow

- layers



# Servlety nebo PHP?

- potřeba psát HTML stránky pohodlně
- napojení na beany
- stránku napíšeme deklarativně
- nakonec se ještě naučíme, jak dělat aplikace ještě pohodlněji bez hluboké znalosti HTML, CSS a Javascriptu (JSF)

# MVC

- Proč?
- **M**odel – data; entity beany (z databáze)
- **V**iew – nakonec HTML stránka
- **C**ontroller – session beany, zpracování formuláře
- Kam spadá formulář? Znovuzobrazení chybně zadaných dat, řešení navigace, zobrazení výsledků, business logika?
- Jeden vývojář je odborník na business logiku a druhý na HTML/CSS/Javascript? Jak mezi ně rozdělit práci?
- Řešením je rozdělení: webový odborník píše HTML nebo JSP tagy a programátor implementuje tagy.

# Struts

- První široce přijímaný framework pro Javu
- Mnoho aplikací v Javě (do)dnes používá Struts 1
- implementuje MVC
- každé stránce odpovídá „action“, hlavní servlet zpracovává události a vyvolává akce
  - spravuje action bean, jejich automatické naplnění, transformace a validace
  - deklarativně popisuje tok stránek. Pro stránku existují forward labely (např. „success“, „fail“, které jsou v konfiguraci mapovány na konkrétní stránky)

# Struts example

```
public class SampleAction extends Action {  
    public ActionForward execute(ActionMapping mapping,  
ActionForm form, HttpServletRequest request, HttpServletResponse  
response) {  
    String param = request.getParameter("param1");  
    if(param.equals("")) {  
        return mapping.findForward("fail");  
    }  
    ...processing...  
    request.setAttribute("result", result);  
    return mapping.findForward("success");  
}
```

# JSP intro

- JSP se inspirovalo v některých ohledech frameworkem Struts
- větší tlak na MVC, ačkoliv stále ještě nedotažené
- orientuje se na view, jde především o HTML výstup
- JSP se překládá na servlet, ale není potřeba ho uvádět ve web.xml – kontejner udělá všechnu otrockou práci za nás



# JSP – jak to vypadá

## Java uvnitř HTML

```
<html><head>...</head><body>
```

```
<h1>Násobilka 7</h1>
```

```
<table>
```

```
<% for(int i=1;i<=10;i++) { %>
```

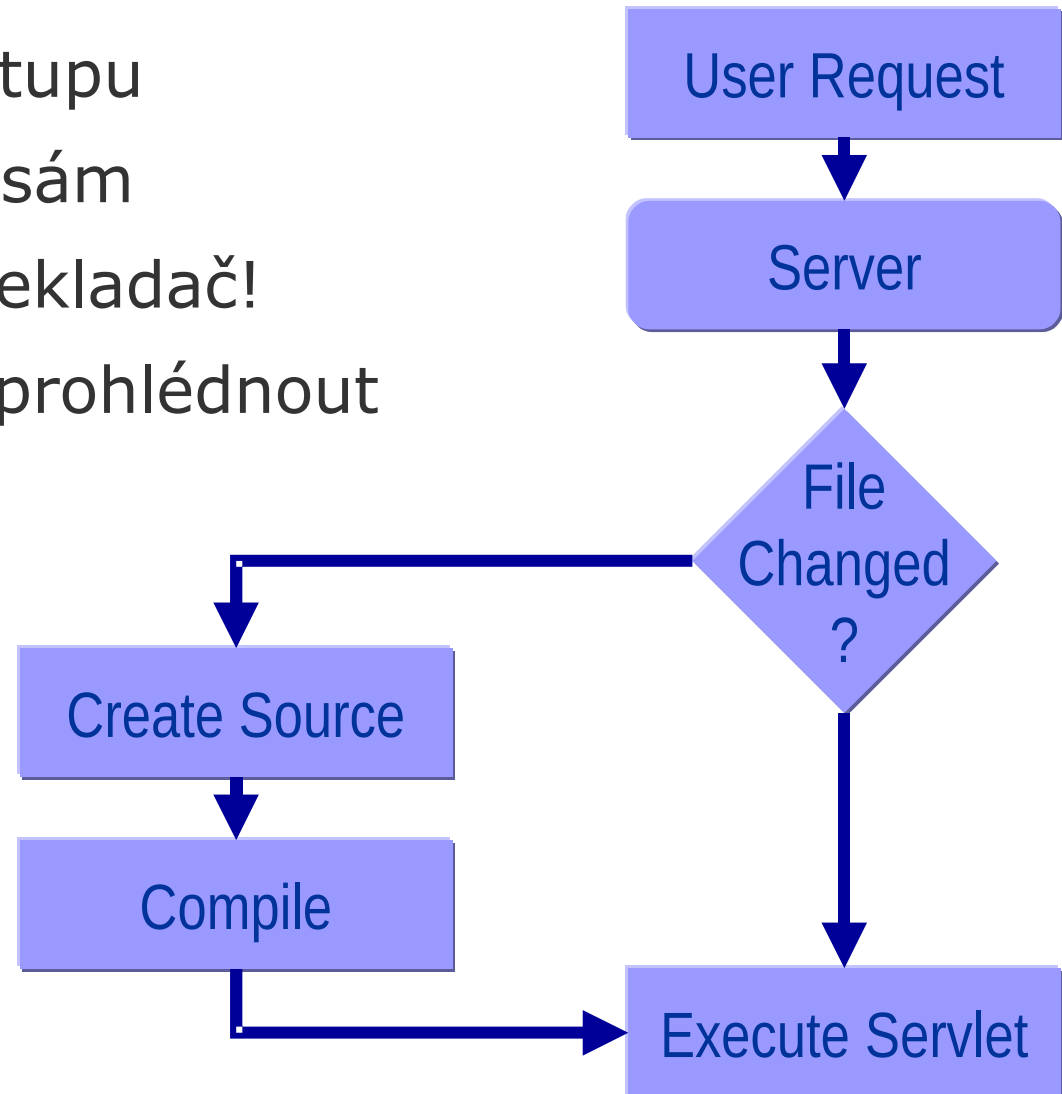
```
<tr><td><%= i %></td><td><%=i*7%></td></tr>
```

```
<% } %>
```

```
</table></body></html>
```

# JSP lifecycle

- překlad při prvním přístupu
- refresh dělá kontejner sám
- kontejner potřebuje překladač!
- výsledný servlet si lze prohlédnout
- NB umí krokovat i JSP



# JSP scriptlets

- **<% code %>**
  - vložení Java kódu, do výsledného servletu se prostě okopíruje do zpracující metody
- **<%= expression %>**
  - ve výsledku se stane argumentem `out.println(expr)`
- **<%! init-block %>**
  - inicializační blok, stane se součástí třídy, používá se pro definici data memberů a metod
- **<%@directive %>**
  - direktivy ovlivňující zpracování zprávy
- **<%-- comment --%>** - není součástí výstupu

# JSP – init block

```
<%!  
    private BookDBAO bookDBAO;  
  
    public void jspInit() {  
        ...getServletConfig()...  
        bookDBAO = new BookDBAO();  
    }  
  
    public void jspDestroy() {  
        bookDBAO.cleanup();  
    }  
%>
```

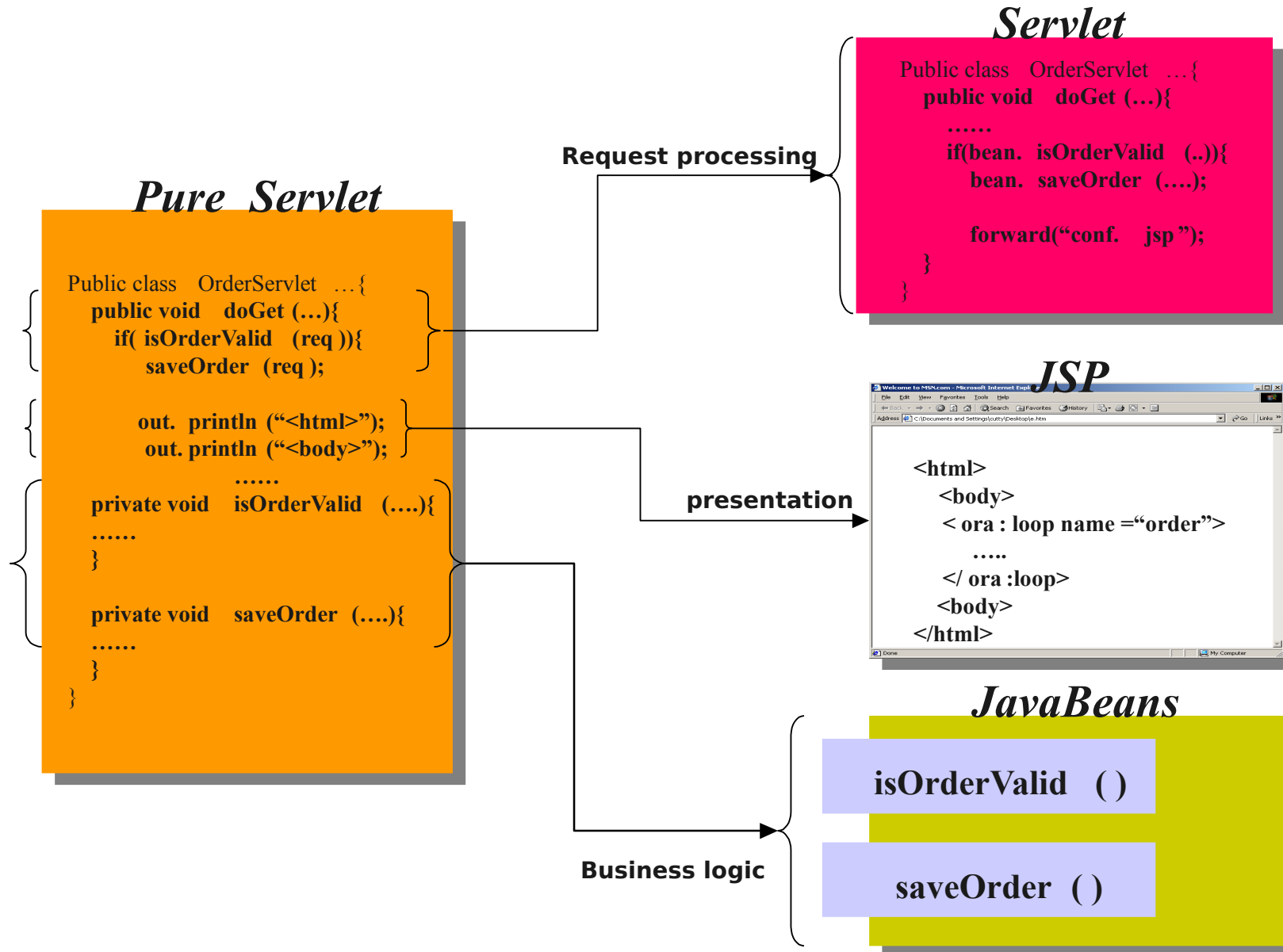
# JSP directives

- `<%@page import="java.util.*,java.math.*" %>`
  - zpracování aktuálního souboru, viz následující slide
- `<%@ page errorPage="errorpage.jsp" %>`
  - je to pohodlnější a flexibilnější než ve web.xml
- `<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
  - použití knihoven (musí být k dispozici odpovídající knihovna, která registruje dané url)
- `<%@include file="response.jsp" %>`
  - vkládání

# JSP page directive

- `<%@ page contentType="text/html; charset=utf-8" %>`
  - MIME typ návratové stránky a znaková sada!
- `<%@ page import="java.util.*" %>`
  - Which classes are imported
- `<%@ page isThreadSafe="true" %>` `<%!--Default --%>`
- `<%@ page isThreadSafe="false" %>`
  - How multithreading is handled

# Servlety a JSP vs MVC



# JSP po překladu

```
<h1>Hello World!</h1>
```

```
<p>It's <%= new Date() %></p>
```

```
PageContext pageContext = _jspxFactory.getPageContext(this, request,  
    response, null, true, 8192, true);
```

```
HttpSession session = pageContext.getSession();
```

```
ServletContext application = pageContext.getServletContext();
```

```
ServletConfig config = pageContext.getServletConfig();
```

```
JspWriter out = pageContext.getOut();
```

```
Object page = this;
```

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

```
out.write("    <h1>Hello World!</h1>\n");
```

```
out.write("    <p>It's ");
```

```
out.print( new Date() );
```

```
out.write("</p>\n");
```



# Objekty k dispozici

- request (HttpServletRequest)
- response (HttpServletResponse)
- session (HttpSession)
- application(ServletContext)
- out (of type JspWriter)
- config (ServletConfig)
- pageContext

# Použití JavaBean

- **POJO** (nikoliv JEE bean)
- **<jsp:useBean id="cart" class="cart.ShoppingCart" scope="session"/>**
  - daná beana bude k dispozici v daném scope – kontejner ji vytvoří s pravuje sám
- **<jsp:getProperty name="cart" property="price" />**
  - `out.println(cart.getPrice());`

# Porovnání použití beany

```
<%  
    ShoppingCart cart = (ShoppingCart)session.getAttribute("cart");  
    if (cart == null) {  
        cart = new ShoppingCart();  
        session.setAttribute("cart", cart);  
    }  
%>
```

**versus**

```
<jsp:useBean id="cart" class="cart.ShoppingCart"  
            scope="session"/>
```

# Zpracování formuláře

```
<jsp:setProperty name="beanName"  
property="propName"/>
```

```
<%
```

```
    String bookId = request.getParameter("bookId");
```

```
    bookDB.setBookId(bookId);
```

```
%>
```

```
<jsp:setProperty name="beanName"  
property="*/>
```

- inspirace Struts, všechny property dané beany jsou načteny (existují-li dané parametry)

# Složitější příklad

```
<jsp:useBean id="locales" scope="application" class="MyLocales"/>
<form name="localeForm" action="index.jsp" method="post">
<select name="locale">
<%
    Iterator i = locales.getLocaleNames().iterator();
    String selectedLocale = request.getParameter("locale");
    while (i.hasNext()) {
        String locale = (String)i.next();
        if (selectedLocale != null && selectedLocale.equals(locale) ) { %>
            <option selected><%=locale%></option>
<% } else { %> <option><%=locale%></option> <% }
        }
    %>
</select>
<input type="submit" name="Submit" value="Get Date">
</form>
```

# Vkládání dalších stránek

- include directive

- `<%@ include file="banner.jsp" %>`

- zpracování během překladu

- jsp:include element

- `<jsp:include page="date.jsp"/>`

- zpracování za běhu

- pomalejší, ale dynamičtější

# Forward

```
<jsp:forward page="..." >
```

```
  <jsp:param name="param1" value="value1"/>
```

```
</jsp:forward>
```

- proč se parametry zadávají takto?
- url rewriting

# JSP 2.0

- expression language

- rozpoznán ve statickém textu a attributech tagu

```
<%=item.getName()%>
```

```
${item.name}
```

```
<% Map m = (Map)pageContext.getAttribute("state");
```

```
State s = ((State)m.get("CZ"));
```

```
if( s != null ) { %>
```

```
    <%= s.getCapitol() %> <%
```

```
} %>
```

```
${state["CZ"].capitol}
```



# JSTL

- JSTL zjednodušují tyto oblasti:
- Core (c): proměnné, podmínky, iterace, podpora práce s URL
- I18n (ftm): locale, formátování (datum, čísla, ...)
- XML (x): přístup k XML datům, transformace
- Funkce (fn): kolekce, manipulace se stringy, ...
- DB (sql): přístup k databázím
  - rychlé prototypování, neukážeme, aby vás to nelákalo použít

# JSTL – Core

- Jiné cykly a iterace
- **<c:forEach var="customer" items="{customers}">**
  - <c:if test="{customer.address.country == 'USA'}">**
    - {customer}<br>**
  - </c:if>**
- </c:forEach>**
- **<c:forEach var="item" items="{enumeration}"**
  - begin="2" end="10" step="2">**
    - <c:out value="{item}"/><br>**
  - </c:forEach>**
- **<c:forEach var="token" items="bleu,blanc,rouge">**
  - <c:out value="{token}"/><br>**
  - </c:forEach>**

# JSTL – Core

- Rozdělení textu na tokeny

```
<c:forTokens var="token" items="one,two,three"  
  delims=",">
```

```
  <c:out value="{token}"/>
```

```
</c:forTokens>
```

- Složitější výstupy

```
<c:out value="{customer.phoneCell}"
```

```
  escapeXml="false">
```

```
    <font color="red">no cell phone specified</font>
```

```
  </c:out>
```

```
<c:out value escapeXml default="def. value"/>
```

- Je-li u c:out parametr java.io.Reader, přečtou se z něj data a vloží se do výstupu

# JSTL – Core

- Větvení (switch)

```
<c:forEach var="customer" items="{customers}">
```

```
<c:choose>
```

```
<c:when test="{customer.address.country == 'USA'}">
```

```
<font color="blue"> </c:when>
```

```
<c:when test="{customer.address.country == 'Canada'}">
```

```
<font color="red"> </c:when>
```

```
<c:otherwise>
```

```
<font color="green">
```

```
</c:otherwise>
```

```
</c:choose>
```

```
{customer}</font> <br>
```

```
</c:forEach>
```

# JSTL – XML

- Práce s XML

```
<c:set var="xmlText">
```

```
  <a>
```

```
    <b><c>foo</c></b>
```

```
    <d>bar</d>
```

```
  </a>
```

```
</c:set>
```

```
<x:parse var="a" doc="{xmlText}" />
```

```
<x:out select="$a//c[@id='123']"/>
```

```
<x:out select="$a/a/d"/>
```

- x:out funguje podobně jako c:out (navíc XPath konverze)

# JSTL – I18n a formátování

- Z requestu lze zjistit preferované locale uživatele (ukázka nastavení ve Firefoxu)
- Nastavení locale
  - `<fmt:setLocale>` ... přepíše nastavení klienta
  - `<fmt:requestEncoding>` ... prepíše encoding klienta
- Lokalizace textů
  - `<fmt:bundle>` ... resource bundle pro stránku
  - `<fmt:message key="...">` s `<fmt:param>` podtagem `<fmt:setBundle>`

# JSTL – I18n a formátování

## - Formátování čísel a data

- `<fmt:formatNumber>`, `<fmt:parseNumber>`

- `<fmt:formatDate>`, `<fmt:parseDate>`

- `<fmt:setTimeZone>`, `<fmt:timeZone >`

## - Příklad použití

```
<fmt:setLocale value="de"/>
```

```
<fmt:bundle basename="cz.myapp.resources">
```

```
  <fmt:message>greetingMorning</fmt:message>
```

```
</fmt:bundle>
```

# Reference na knihovny tagů

- web.xml

```
<jsp-config>
```

```
  <taglib>
```

```
    <taglib-uri>/tlt</taglib-uri>
```

```
    <taglib-location>/WEB-INF/lib.tld</taglib-location>
```

```
  </taglib>
```

```
</jsp-config>
```

```
<%@ taglib prefix="tlt" uri="/tlt"%>
```

- Přímá reference

```
<%@ taglib prefix="tlt" uri="/WEB-INF/iterator.tld"%>
```

- Absolutní reference

```
<%@ taglib prefix="core"
```

```
  uri="http://java.sun.com/jsp/jstl/core"%>
```



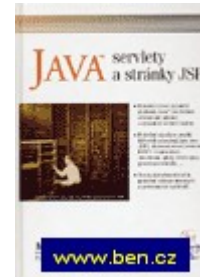
# Summary

- JSP slouží jako prezentační vrstva pro webovou aplikaci.
- Seznámili jste se se základy JSP, JSTL.
- Nyní byste měli umět view, alespoň pokud ovládáte HTML+CSS.
  - Model obstarávají entity beany (aka JPA).
  - Controller si probereme příště (session beans).
- Pokud nechcete zabředávat do problémů JSP a chcete dělat „cool“ aplikace, přes příští přednáška bude o JSF.

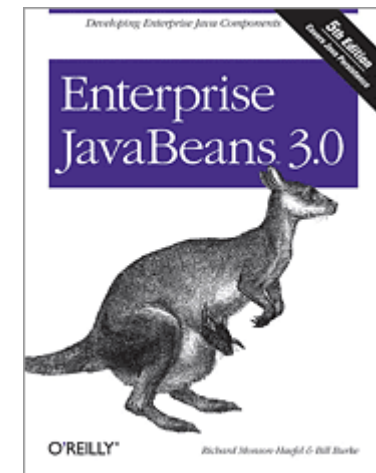
# Materiály k předmětu

- [java.sun.com](http://java.sun.com) - hromady materiálů, tutoriálů
- [www.javapassion.com](http://www.javapassion.com) - výborný zdroj informací, slidy\*
- <http://java.sun.com/products/jsp/>

- Java servlety a stránky JSP



- Enterprise JavaBeans 3.0, O'Reilly, 2006



\* některé obrázky a slidy byly převzaty odsud