Java technologie pro webové aplikace

PIA 2011/2012

Téma 6

Základní pojmy

- Java Enterprise Edition
  - "enterprise" část Java technologie; aktuálně JavaEE 6 (2010)
  - jednou z částí servlety a JSP
- Kontejner
  - prostředí pro běh servletů
  - Tomcat (Apache Jacarta projekt), aktuálně v6 (2010)
- Servlet
  - Java třída která umí obsloužit HTTP požadavek
  - aktuální verze specifikace 3.0 v JavaEE v6
- JavaServer Page (JSP)
  - Java jako zapouzdřený HTML skriptovací jazyk
  - aktuální verze specifikace 2.1

Zdroje informací

- Standardy a specifikace
  - většinou fa Sun Microsystems, JCP
  - http://java.sun.com/javaee/ → „Technologies“→ „Web Application“
- Knihy
  - B.Kurniawan: Java for the Web… (New Riders)
  - Bollinger: JSP – JavaServer Pages (Grada)
  - M.Hall: Java servlety a stránky JSP (Neocortex)
- On-line
  - Java EE Tutorial od Sunu
  - BP D.Maixner (slunecnice.cz), články na root.cz a interval.cz

Java servlety: základy

Hello World servlet

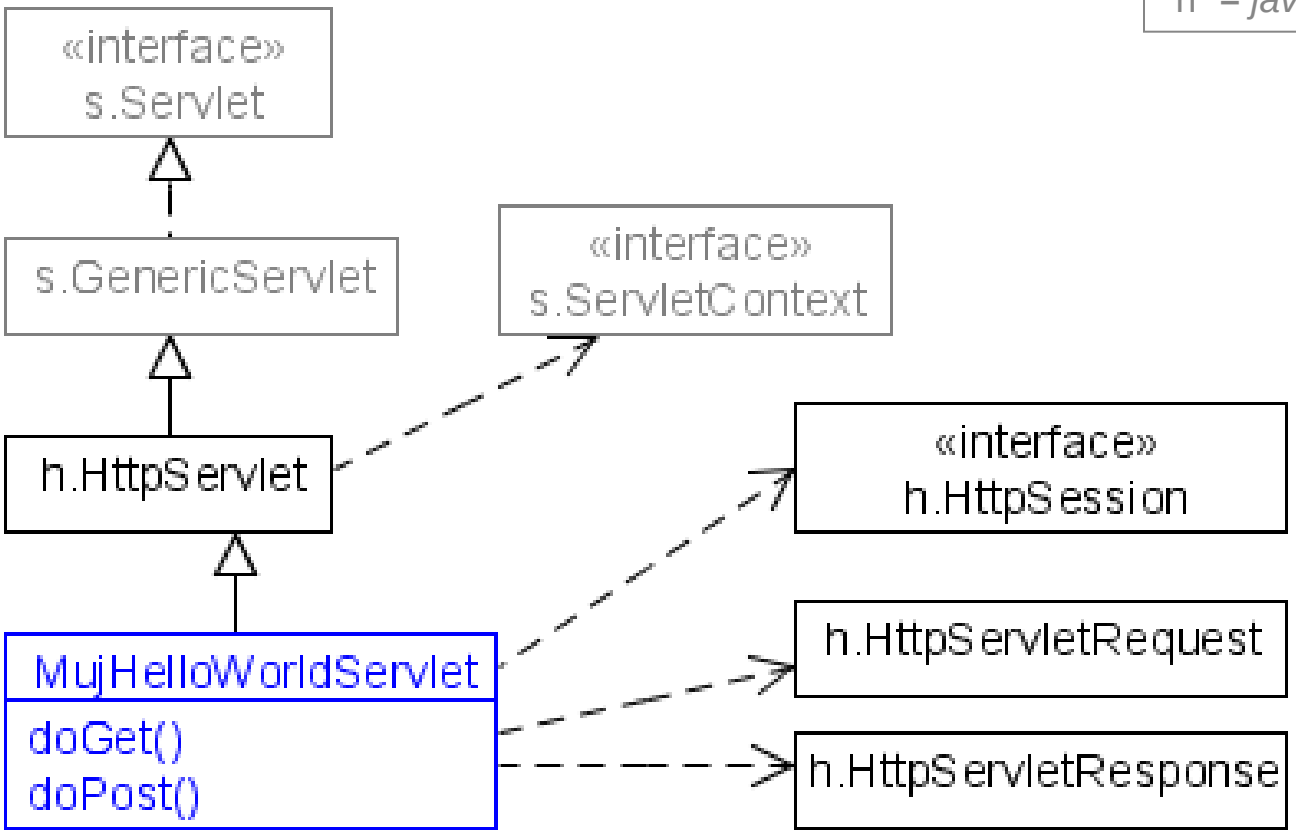
```
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldExample extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<html>\n<head>");
        String title = "helloworld";
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>" + title + "</h1>");
        out.println("</body>\n</html>");
    }
}
```

Pracovní cyklus servletu

- Vytvoření, kompilace
  - kód servletu, podpůrný kód; *servlet-api.jar*
  - deployment descriptor
- Packaging
  - nepovinné, .war soubor (viz dále)
- Nasazení, konfigurace
  - vložení do kontejneru, informování kontejneru, konfigurace (staging, role)
  - kontejner natáhne, instancuje a inicializuje servlet
- Čekání, obsluha požadavku (cyklicky)
  - kontejner odchytí HTTP požadavek
  - určí, který servlet jej zpracuje (server config)
  - spustí obslužnou metodu servletu
  - servlet obdrží data požadavku, generuje odpověď
- Ukončení
  - kontejner spustí finalizační metodu servletu

Základní třídy a rozhraní

"s" = *javax.servlet* Java package
"h" = *javax.servlet.http* package



Pomocné třídy a rozhraní

- „Prostředí“ servletu
  - rozhraní *s.ServletContext* a *s.ServletConfig*
    - obvykle získány pomocí metod *h.HttpServlet*
  - třída *s.ServletOutputStream* resp. *java.io.PrintWriter*
    - instance získána obvykle z HttpServletResponse
  - rozhraní *h.HttpSession*
    - instance získána obvykle z HttpServletRequest
- Pomocné třídy
  - třída *h.Cookie*
  - výjimky *s.ServletException, java.io.IOException*

Obsluha požadavku

- Obslužné metody *doGet* | *doPost* | …

  *protected void do*Method*(HttpServletRequest req, HttpServletResponse resp)*

  – zavolány kontejnerem podle HTTP metody
  – konvence

```
void doPost(…)  { this.doGet(…); }
```

Kroky při obsluze požadavku

- Určit, zda je HTTP metoda implementována
  - » nechci obsluhovat ⇒ neimplementuji doXxx()
- Získat vstupní parametry/data požadavku
- Získat proud (stream) pro tvorbu těla odpovědi
- Nastavit content type odpovědi
- Generovat data odpovědi
  - nebo delegovat na jiný servlet/JSP
- Zapsat odpověď do proudu

  nebo

  nastavit chybový kód

Získání dat požadavku

- Rozhraní *s.ServletRequest* a *h.HttpServletRequest*
  - druhé = formální parametr obslužných metod
- Hrubá data
  - *Enumeration getHeaderNames()*
    + String getHeader(String name)
  - *StringBuffer getRequestURL(); String getQueryString();*
- Předzpracovaná data
  - *String getRemoteAddr(); getContentType();*
  - *Locale getLocale();*
  - *String getPathInfo(); boolean isSecure(); …*
  - *String getParameter(String name);*
  - *Cookie[] getCookies();*

Vytváření odpovědi

- Rozhraní *s.ServletResponse* a *h.HttpServletResponse*
- Výstupní proudy
  - *ServletOutputStream getOutputStream();* pro binární data
  - *java.io.PrintWriter getWriter();* pro textová (HTML)
    - oba mají přetížené metody *print()* a *println()*
- Nastavení stavového kódu
  - *void sendError(int sc)* + konstanty *HttpServletResponse.SC_**
- Nastavování hlaviček
  - *void setContentType(String type);*
    - » podobně délka, charset, …
  - *void setHeader(String name, String value);*

Pozor na výstup

- Generování HTML
  - připravit data, zavolat generující metodu
  - nejlépe přes JSP apod.

- „Pozdní“ hlavičky
  - bufferování výstupu *by default* vypnuto
    $\Rightarrow$ data jsou posílána okamžitě (propustnost)
  - *boolean isCommitted();* když chci zjistit, zda není pozdě *+ void resetBuffer();*
  - *void setBufferSize(int size);* když chci poslat chybový kód nebo hlavičky „až po těle“

Servlet v aplikaci

… s úvodem ComplexHelloExample

Složky servletové aplikace

- Servlety
  - přeložený kód
- JSP a HTML stránky
  - view vrstva
- Popis aplikace
  - deployment descriptor

Za run-time:
- celá aplikace (kontext)
- uživatelská relace
- požadavek
- zdroje (db, soubory, prostředí)

Adresářová struktura
- kořenový adresář → statické/JSP soubory na / URL
- podadresáře → ditto pro vnořené úrovně URL
- *WEB-INF/web.xml* = deployment descriptor
  - *WEB-INF/classes/* → servlety a pomocné třídy
  - *WEB-INF/lib/*.jar* → Java archivy se servlety, beany,…

Kontext servletu

- Kontext = webová aplikace (≤1 v kontejneru)
  - dovoluje servletu komunikovat s kontejnerem
    - inicializační parametry, atributy
    - další zdroje
    - logování
  - definovaná adresářem, v němž je servlet nasazen, a deployment descriptorem
- Přístup ke kontextu
  - přes rozhraní *javax.servlet.ServletContext*
  - přes metodu *getServletContext()* z *GenericServlet*

Example:


```
<Context
 docBase="d:/www/tomcat/pokusy"
 path="/pokusy">
</Context>
```

nadpis

Konfigurace aplikace

- Parametry celé aplikace v definici kontextu
  - přiřazeny v deployment descriptoru

```
<web-app>
  <context-param>
        <param-name>app name</param-name>
        <param-value>My Appli</param-value>
  </context-param>
  …
```

  - přístup přes *ServletContext* interface
    - *Enumeration getInitParameterNames();*
      String getInitParameter(String name);
    - *String getServletContextName(); String getServerInfo();*
    - viz *ServletConfig.getServletContext()*

Inicializace servletu

- Při natažení (instanciaci) kontejnerem
- Typické akce
  - načíst konfigurační data
  - otevřít spojení (db), připojit se ke zdrojům
  - inicializovat lokální data
- Metoda

  void init(ServletConfig config)
  - pomocné → *ServletConfig* interface
    - *Enumeration getInitParameterNames();*
      String getInitParameter(String name);
    - *String getServletName();*

Nastavení konfig. parametrů

- Deployment descriptor

```
<servlet>
  <init-param>
        <param-name>default-login</param-name>
        <param-value>guest</param-value>
  </init-param>
  …
```

  - přístup přes *ServletConfig* interface
  - servlet musí znát typy/třídy datových položek

Předávání hodnot v aplikaci

- Komunikace mezi servlety
  - přes objekty v různých vrstvách aplikace
  - různé rozsahy platnosti předávaných dat
- Obecné rozhraní, obecný mechanismus
  - atributy objektů, get/set metody
    - *Enumeration getAttributeNames();*
    - *Object getAttribute(String name);*
    - *void setAttribute(String name, Object object);*
    - *void removeAttribute(String name);*

Rozsahy platnosti

- Objekty reprezentující rozsahy
  - rozhraní *HttpServletRequest* | *HttpSession* | *ServletContext*
  - požadavek (request)
  - aktuální servlet
  - session
  - aplikace (context)

Bylo by dále potřeba (viz Seam):
- konverzace
- stav záložek
- business proces

- Získání objektu
  - *HttpSession HttpServletRequest.getSession();*
  - *ServletContext GenericServlet.getServletContext();*

Sessions, správa relací

- Primitivní metody
  - skryté prvky formuláře, parametry URL, cookies
- Objekt relace (rozhraní *HttpSession*)
  - reprezentuje relace, obsahuje její data
  - získaný přes metody *HttpServletRequest*
    - *getSession()* → vrací aktuální, nebo vytváří novou relaci
    - *getSession(boolean create)* → "false" znamená "nevytvářet ani pokud neexistuje"
      - » obě pouze pokud tělo není ve stavu commited
        - Víte proč?

Vlastnosti relace

- Konfigurace
  - *get|setMaxInactiveInterval(int seconds);*
- Vlastnosti
  - *String getId();*
  - *long getCreationTime();*
  - *get/setAttribute* – viz objekty pro rozsahy platnosti
  - platnost relace
    - *HttpServletRequest* :: *boolean isRequestedSessionIdValid()*
    - *HttpSession* :: *void invalidate()*
    - nejlépe ověřit zjištěním hodnoty nějakého atributu

Odkazování na jiné zdroje

- Zdroj = jiný servlet, jakýkoli jiný obsah/objekt
  - JSP, HTML, …
- Nepřímý odkaz
  - klientovi pošleme redirect: *resp.sendRedirect(String location);*
  - přenos stavových informací pouze v URL

Odkazování na jiné zdroje (2)

- Přímé odkazování na zdroje
  - » in-process, atributy přes request objekt
  - rozhraní *ServletRequest* a *ServletContext*: *RequestDispatcher getRequestDispatcher(String path) a getNamedDispatcher(String name);*
  - přesměrování: *disp.forward(req,resp);*
    - » `RequestDispatcher rd = request.getRequestDispatcher("/barvy.jsp");`
    - » `rd.forward(request, response);`
    - předání řízení, bez možnosti návratu
    - tělo odpovědi nesmí být ve stavu *isCommited()*
      - » hlavičky nevadí
    - cesta upravena kontejnerem v req parametru
  - vložení: *disp.include(req,resp);*
    - návrat zpět do servletu
    - vkládaný servlet/objekt nemůže měnit hlavičky

Chybové stránky

- Možnost specifikovat odpověď klientovi při chybě
  - deployment descriptor: *<error-page>* element
  - příčina chyby, URL stránky
- Příčiny chyby
  - výjimka v aplikaci
  - volání *response.sendError(code [, msg])*

Servletová aplikace (2)

- Deployment descriptor
  - popisuje součásti a nastavení aplikace
  - XML soubor

```
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
```

  - viz dále
- Distribuce aplikace: WAR file
  - JAR soubor s příponou .war
  - obsahuje uvedenou adresářovou strukturu, navíc META-INF adresář

Deployment Descriptor

```
<web-app xmlns="..." ... >

<display-name>A Simple
    Application</display-name>
<context-param>
  <param-name>webmaster</param-name>
  <param-value>webmaster@my.com</param-
    value>
</context-param>

<servlet>
  <servlet-name>catalog</servlet-name>
  <servlet-class>com.my.CatalogServlet
      </servlet-class>
  <init-param>
    <param-name>catalog</param-name>
    <param-value>Spring</param-value>
  </init-param>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>catalog</servlet-
    name>
  <url-pattern>/catalog/*</url-
    pattern>
</servlet-mapping>
<session-config>
  <session-timeout>30</session-
    timeout>
</session-config>

<welcome-file-list>
  <welcome-
    file>index.jsp</welcome-file>
  <welcome-
    file>index.html</welcome-file>
</welcome-file-list>
<error-page>
  <error-code>404</error-code>
  <location>/404.html</location>
</error-page>

</web-app>
```

Několik speciálních témat

Thread Safe servlety

- Java web applikace jsou od přírody vícevláknové
  - nový požadavek = nové obslužné vlákno (kontejner)
- Ruční řešení
  - používat *sychronized* metody či bloky
- Jednoduché řešení
  - servlet implementuje rozhraní *SingleThreadModel*

  ⇒ kontejner zaručuje serializaci přístupu k metodám
  - neřeší sdílené zdroje

Filtry

- Článek zpracování požadavku
  - nevytváří, jen transformuje
  - autentikace, logování, komprese, …
  - filtry spojeny do řetězu
- Rozhraní s.Filter
  - metoda *doFilter()*
  - inicializace, ukončení

```
public class AuthFilter implements Filter {

public void doFilter(ServletRequest
request, ServletResponse response,
FilterChain chain) {
  if (request.getParameter("user") == null)
    response.sendError(response.
SC_FORBIDDEN, "No login specified");
  else
    chain.doFilter(request,response);
  }

...

}
```

Filtry – konfigurace

- Deployment descriptor
  - podobné servletu
- Mapování
  - na kterých URI
  - v jakém okamžiku
    REQUEST
    FORWARD
    INCLUDE
    ERROR

```
<filter>
<filter-name>AuthBlocker</filter-name>
<filter-class>cz.zcu.AuthFilter</filter-class>
 <init-param>
  <param-name>group</param-name>
  <param-value>administrators</param-value>
 </init-param>
</filter>

<filter-mapping>
  <filter-name>AuthBlocker</filter-name>
  <url-pattern>/admin/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Listenery („posluchači“)

- Reakce na události v aplikaci
  - návrhový vzor Observer či Listener
  - třídy *XxxListener* a *XxxEvent*
- Úrovně událostí (Xxx = …)
  - aplikace: *ServletContext*
  - relace: *HttpSession, HttpSessionAttribute, …*
  - požadavek: *ServletRequest*
- Metody posluchače
  - ContextInitialized(ServletContextEvent sce)
  - requestDestroyed(ServletRequestEvent rre)
  - attributeAdded(HttpSessionBindingEvent se)
  - …
- Metody události: obvykle vrací objekt dané úrovně

web.xml

```
<web-app>
<listener>
  <listener-class>
cz.zcu.ObjCounter
  </listener-class>
</listener>
…
```

Logování

- Možno psát na stdout/stderr
  - System.out.println(msg)
  - vypisuje se do konzole spuštění serveru
- Perzistentní hlášení = do logu
  - přes kontext servletu
    - context.log(String)
    - context.log(String,Throwable)
  - využít logovací knihovny
    - commons logging, log4j, java.util.logging apod.
- Soubor s logem nohup.out
  - Pro primitivní ladění použijete: tail –f nohup.out

Práce s databázemi

- Přes JDBC
  - » java.sql.* , driver class
- Spojení do db
  - *Class.forName("com.mysql.jdbc.Driver");*
    conn = DriverManager.getConnection(dbUri);
    - » jdbc:mysql://jumbo.fav.zcu.cz/database?user=name&password=..
  - *Connection.connect(db, user, pass);*
- Dotazy
  - *String sql = "SELECT * FROM brada_pokus";*
  - *Statement st = conn.createStatement();*
  - *ResultSet rs = st.executeQuery(sql); | execute(sql);*
    rs.getInt("ID"); rs.getString("name"); ...
  - *catch (SQLException e)*