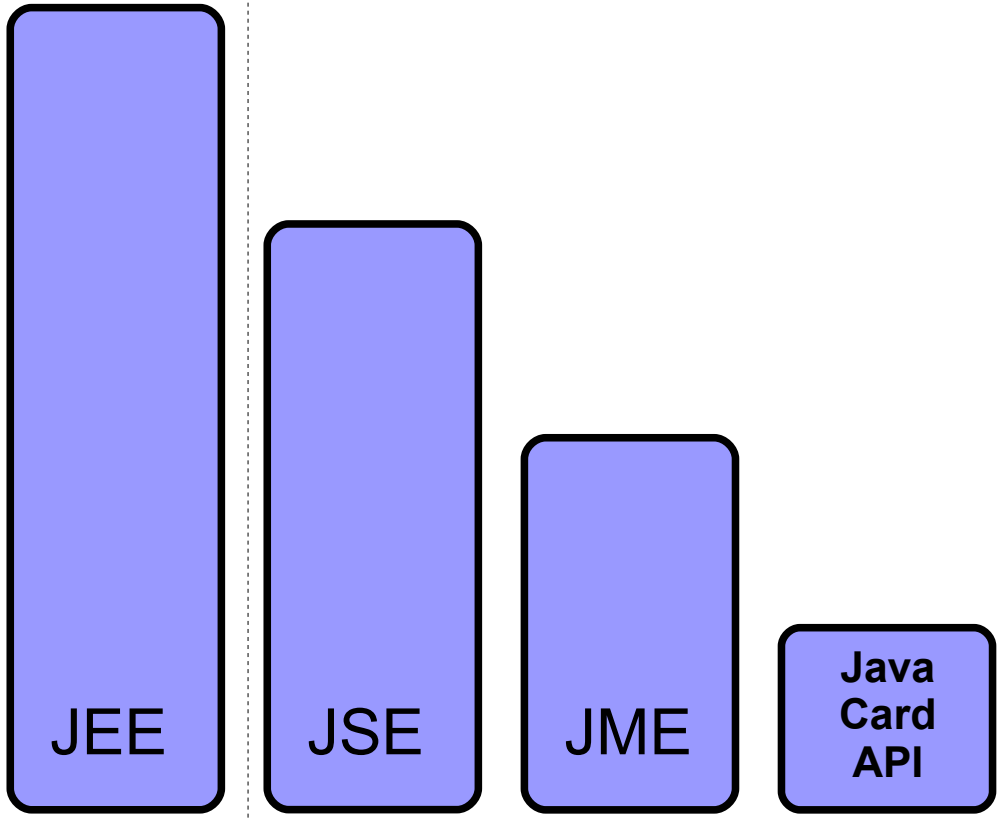
Webové aplikace
Servlety, JSP, JSF

JEE

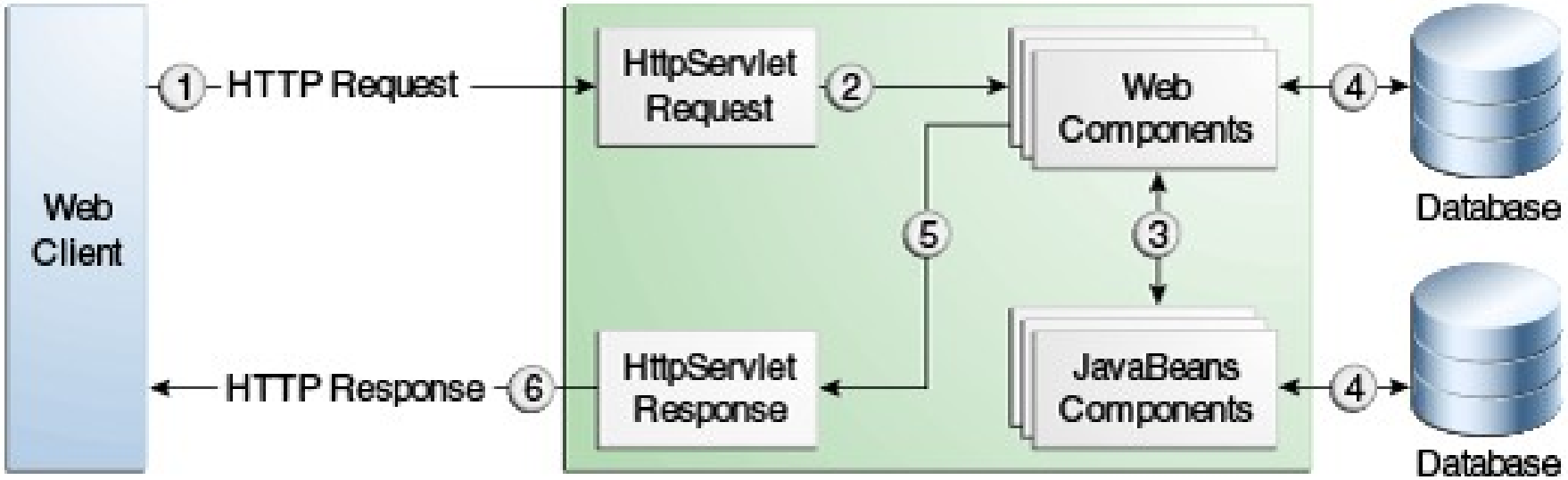
- web aplikace
  - servlety
  - JSP
  - JSF
  - …
- web services
- dependency injection
- EJB
- security
- persistency
- …



Přehled

- většina dnešních webových stránek je *dynamická*
  - technologie a jazyky – CGI, PHP, ASP,...
    - dynamicita na straně serveru
- základní Java technologie
  servlety, Java Server Pages, Java Server Faces
- Servlet
  - program v Javě
  - běží uvnitř serveru (Java web container)
  - obsluhuje požadavky od klienta (prohlížeče)
- JSP
  - umožňují přímo do HTML kódu vkládat Java kód plus další elementy
- JSF
  - kombinace servletů a šablon

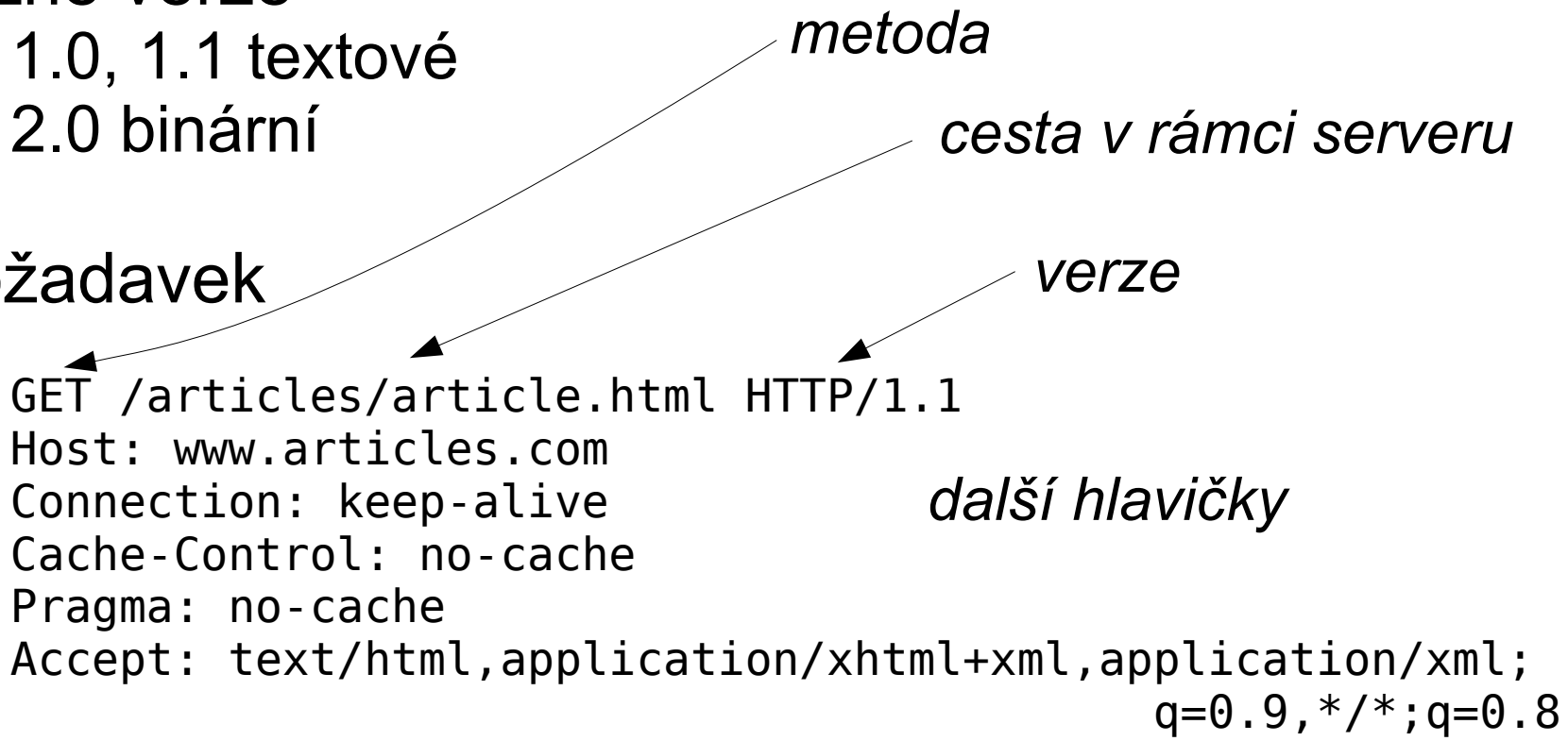
Přehled



zdroj obrázku: https://eclipse-ee4j.github.io/jakartaee-tutorial/webapp001.html

HTTP

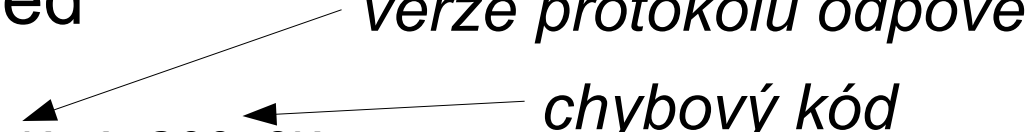
- různé verze
  - 1.0, 1.1 textové
  - 2.0 binární
- požadavek



- metody
  - OPTIONS, HEAD, GET, POST, PUT, DELETE, TRACE

HTTP

- odpověď




```
HTTP/1.1 200 OK
Date: Sun, 09 Apr 2017 12:48:21 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 25503
Cache-Control: no-cache
Content-Encoding: gzip

tělo odpovědi...
```

další hlavičky

- chybové kódy
  - 1xx informační
  - 2xx úspěch
  - 3xx přesměrování
  - 4xx chyby od klienta
  - 5xx chyby serveru

JAVA

Servlety

Struktura servletu

- API
  - javax.servlet
  - javax.servlet.http
- inteface **javax.servlet.Servlet**
  - každý server ho musí implementovat
  - metody
    - `public void init(ServletConfig config) throws ServletException;`
    - `public ServletConfig getServletConfig();`
    - `public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;`
    - `public String getServletInfo();`
    - `public void destroy();`

Struktura servletu

- interface Servlet se typicky neimplementuje přímo, ale přes třídu **javax.servlet.http.HttpServlet**
  - **`protected void service(HttpServletRequest req, HttpServletResponse resp)`**
    - přijme http požadavek
    - distribuuje volání na **`do<něco>()`** metody
    - obvykle se nepředefinovává
      - předefinovávají se **`do<něco>()`** metody
  - **`void doGet(HttpServletRequest req, HttpServletResponse resp)`**
    - obsluha http GET požadavku
  - ostatní „do“ metody
    - doPost, doDelete, doHead, doPut, doOptions, doTrace
    - stejné parametry jako **`doGet`**
  - **`long getLastModified(HttpServletRequest req)`**

Hello world

```
package prg;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

  protected void doGet(HttpServletRequest req,
                                    HttpServletResponse res)
                            throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><HEAD><TITLE>Hello World!</TITLE>"+
        "</HEAD><BODY><H2>Hello World!</H2></BODY></HTML>");
    out.println("<hr><em>"+getServletInfo()+"</em>");
    out.close();
  }

  public String getServletInfo() {
    return "HelloWorldServlet 1.0";
  }
```

Hello world – web.xml

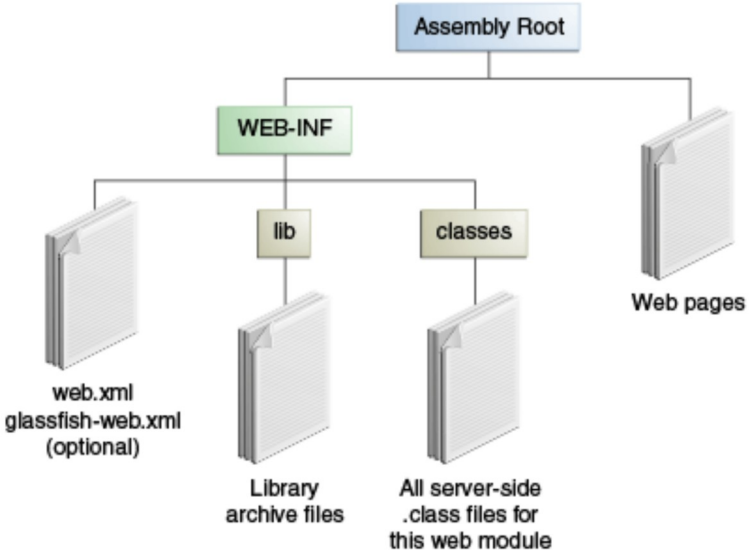
```
<?xml version="1.0" encoding="ISO-8859-2"?>

<!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>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>prg.HelloWorldServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/myHello</url-pattern>
  </servlet-mapping>
</web-app>
```

- nebo přímo v kódu

```
@WebServlet(urlPatterns = { "/myHello" })
public class HelloWorldServlet extends HttpServlet {
    ...
```

Servlet na servru



- adresářová struktura
  - ....webapps/*app-name*/
    - META-INF/ – manifest
    - context.xml – kontext aplikace
    - WEB-INF/
      - classes/ – přeložené třídy
      - lib/ – jar soubory
      - web.xml
    - statické stránky, obrázky,....
- server zakazuje přímý přístup k WEB-INF adresáři
- přesné umístění aplikace je závislé na serveru

zdroj obrázku: https://eclipse-ee4j.github.io/jakartaee-tutorial/packaging003.html

Servery pro nasazení

- Tomcat
  - http://tomcat.apache.org/
  - servlet container
  - „instalace“ servletu
    - nakopírovat do webapps adresáře a restartovat
    - použít Tomcat manager
      - také servlet
- GlassFish
  - https://eclipse-ee4j.github.io/glassfish/
  - nejen pro servlety
  - „instalace“ servletu
    - nakopírovat do *domain-dir*/autodeploy/
    - nástroj as-admin
- …

WAR

- Web ARchive (WAR)
  - distribuce web-aplikací, instalování do serveru,...
  - JAR soubor s adresářovou strukturou web-aplikace
    - tj. WEB-INF, web.xml, classes....
- vytvoření
  - ručně pomocí jar nebo zip
  - pomocí Antu
    - task war
      - př:

```
<target name="war" depends="compile">
   <war destfile="helloworld.war" webxml="web.xml">
    <classes dir="classes"/>
   </war>
 </target>
```

Životní cyklus servletu

- void init(ServletConfig config) throws ServletException
  - volá se automaticky při startu servletu
  - volá se pouze jednou
  - př.

```
public void init(ServletConfig config) throws
  ServletException {
   super.init(config);
   name = config.getInitParameter("name");
}
```

  - „init“ parametry lze nastavit v web.xml

```
<servlet>
   <servlet-name>examplServlet</servlet-name>
   <servlet-class>examplServlet</servlet-class>
   <init-param>
     <param-name>name</param-name>
     <param-value>Petr</param-value>
   </init-param>
</servlet>
```

Životní cyklus servletu

- nebo lze nastavit parametry přímo v kódu
  - vhodné pro implicitní hodnoty

```
@WebServlet(
        urlPatterns = "/uploadFiles",
        initParams = @WebInitParam(name = "location",
                                   value = "/Uploads")
)
public class FileUploadServlet extends HttpServlet {
  ...
```

Životní cyklus servletu

- void init() throws ServletException
  - init bez parametru
  - předefinovat pokud nejsou potřeba init parametry
    - volá se automaticky z init(ServletConfig)
- public void destroy()
  - volá se při ukončení servletu
    - při ukončení servru
    - při automatickém uvolnění servletu z paměti
    - při ukončení z manageru

HttpServletRequest

- reprezentuje http požadavek
  - String getHeader(String name)
  - Enumeration getHeaderNames()
  - StringBuffer getRequestURL()

  - String getScheme()
  - String getServerName()
  - int getServerPort()
  - boolean isSecure()

  - String getQueryString()
  - String getParameter(String name)
  - Map getParameterMap()
  - Enumeration getParameterNames()

HttpServletRequest

- ...pokračování
  - Cookie[] getCookies()
  - HttpSession getSession()
  - HttpSession getSession(boolean create)
- Cookie
  - konstruktor
    - Cookie(String name, String value)
  - metody
    - (get|set)Name, (get|set)MaxAge, (get|set)Value
- HttpSession
  - server automaticky rozhodne, zda se session udržuje přes cookies nebo přes URL
  - metody
    - getId, (get|set)Attribute, setMaxInactiveInterval, invalidate

HttpServletResponse

- sada konstant pro návratové kódy odpovědí
  - SC_OK (200), SC_NOT_FOUND (404),...
- metody
  - setContentType, setContentEncoding
  - ServletOutputStream getOutputStream()
  - void setStatus(int sc)
  - void setHeader(String name, String value)
  - String encodeURL(java.lang.String url)
    - přidá do URL identifikaci session
    - při používání session by všechny URL ve výsledné stránce měli jít přes tuto metodu
  - void addCookie(Cookie cookie)

JAVA

JSP

JSP – přehled

- mix HTML a Javy (a speciálních tagů)
- JSP kód je v HTML vložen pomocí

  `<% JSP kód %>`
- př:

```
<html><body>
<H1>The time in seconds is:
<%= System.currentTimeMillis()/1000 %></H1>
</body></html>
```

- JSP stránky se do WAR struktury umístí na stejné místo jako normální statické elementy
  - tj. mimo WEB-INF

JSP – přehled

- postup zpracování požadavku na JSP
  - při prvním požadavku na JSP se vytvoří Java kód, který implementuje Servlet
    - servlet je přeložen a .class soubor(y) uložen do spec. adresáře
  - vytvořena instance servletu
  - dále jako u běžného servletu
- při překladu JSP -> Java
  - kód mezi <% %> se „zkopíruje“
  - html kód se přeloží na out.print(“.....“)
- typy JSP elementů
  - skriptovací elementy
  - direktivy
  - JSP akce (tagy)
  - vlastní (vývojářem definované) akce (tagy)

Skriptovací elementy

- deklarace
  - uzavřeno v <%! %>
  - jedna nebo více deklarací v jazyce Java
  - spouští se při první návštěvě stránky nebo v okamžiku, kdy kontejner JSP opětovně inicializuje stránku
- výraz
  - uzavřeno v <%= %>
  - jeden výraz v jazyce Java
  - výsledek je hodnota výrazu
  - spouští se při každém přístupu
- skriptlet
  - uzavřeno v <% %>
  - Java kód
  - spouští se při každém přístupu

Příklady

```
<HTML>
<BODY>
Hello!  The time is now <%= new java.util.Date() %>
</BODY>
</HTML>
```

```
<TABLE BORDER=2>
<%
    for ( int i = 0; i < n; i++ ) {
        %>
        <TR>
        <TD>Number</TD>
        <TD><%= i+1 %></TD>
        </TR>
        <%
    }
%>
</TABLE>
```

Příklady

```
<HTML>
<BODY>
<%!
     int theNumber = 42;
     int getNuber() {
          return theNumber;
     }
%>

Hello <%= getNumber() %>
</BODY>
</HTML>
```

Proměnné v JSP

- vytvořené v deklaraci JSP
  - platné v celé JSP stránce
    - definované na úrovni třídy
  - vytvoří se a inicializuje při instanciování servletu (vytvořeného z JSP)
- vytvořené ve skriptletech JSP
  - platné v daném skriptletu
    - definovaná na úrovni metody
  - vytvoří se a inicializuje při každém přístupu na stránku

- nelze definovat metody ve skriptletech
  - protože kód skriptletů je uvnitř (při překladu do servletu vytvořené) metody

Komentáře v JSP

- Java komentáře ve skriptletech
  - // komentář
  - /* komentář */
- JSP komentáře
  - <%-- komentář --%>
  - lze v nich zakomentovat jiné JSP elementy

```
<%-- Zakomentováno: <%= "Hello" %><br> --%>
```

- HTML komentáře
  - <!-- komentář -->
  - dostanou se do výsledné stránky

Implicitní objekty v JSP

- automaticky vytvářené objekty
  - lze je použít ve výrazech a skriptletech
  - nelze je použít v deklaracích
    - vytvářejí se až později

- request
  - instance HttpServletRequest
- response
  - instance HttpServletResponse
- out
  - výstup na výslednou stránku
  - instance jsp.JspWriter
- session
  - instance HttpSession

Implicitní objekty v JSP

- application
  - instance ServletContext
- config
  - instance ServletConfig
- page
  - reference na aktuálně zpracovávanou stránku
- pageContext
  - instance PageContext
  - popis prostředí, v němž jsou všechny stránky spuštěny

Direktivy

- ovlivňují jak se má vygenerovat servlet z JSP
- 3 direktivy
  - page
  - include
  - taglib
- použití
  - <%@ direktiva atribut1="hotnota1" ..... atributN="hodnotaN" %>
- include
  - <%@ include file=„relativní URL" %>
  - vloží soubor v době ***překladu*** stránky
- taglib
  - „importuje" knihovnu uživatelsky-definovaných elementů
  - <%@ taglib uri="soubor TLD" prefix="předpona" %>

Direktiva page

- různé použití
- parametry
  - import
  - errorPage, isErrorPage
  - session
  - info
  - language
  - contentType
  - isThreadSafe
  - buffer
  - autoFlush

Direktiva page

- import
  - import tříd a balíčků
  - <%@ page import=balíček.třída“ %>
- errorPage
  - specifikuje stránku, která slouží ke zpracování výjimek nezachycených na aktuální stránce
  - <%@ page errorPage=„relativní URL“ %>
- isErrorPage
  - zda aktuální stránka je chybová
    - implicitně false
- session
  - zda se pro stránku má udržovat session
  - <%@ page session=“false” %>
- info
  - informace o stránce – typicky autor, copyright,...
  - <%@ page info=“Petr, 2012 ” %>

Direktiva page

- language
  - (programovací) jazyk JSP
  - <%@ page language="java" %>
- contentType
  - implicitní hodnota text/html; charset=iso-8859-1
  - <%@ page contentType=" text/plain; charset=utf-8" %>
- autoFlush
  - implicitně true
  - při false se po naplnění buffer nevyprázdní, ale bude vyvolána výjimka IOException
    - JspWriter.flush()
  - <%@ page autoFlush="false" %>
- extends
  - přímý předek pro vygenerovaný servlet
  - <%@ page extends="třída" %>

JSP akce (tagy)

- jsp:include
  - vloží soubor nebo výsledek do JSP
    - statický soubor (př. html) se vloží
    - dynamický se provede (př. jsp) a vloží se výsledek
  - provede se při každém požadavku na stránku
  - <jsp:include page="hello.jsp"/>
- jsp:param
  - přidání parametrů k jsp:include
  - <jsp:include page="scripts/login.jsp">
    <jsp:param name="username" value="petr" />
    </jsp:include>
- jsp:forward
  - předání aktuálního požadavku jiné JSP
  - <jsp:forward page=“orderError.jsp” >
    <jsp:param name=„errorType“ value=„badAmount“ />
    </jsp:forward>

JSP akce (tagy)

- používání JavaBeans
  - jsp:useBean
    - vytvoření instance
  - jsp:getProperty
    - čtení property
  - jsp:setProperty
    - nastavení property
- př:
  - ```
    <jsp:useBean id="checking" scope="session"
                                          class="bank.Checking" >
       <jsp:setProperty name="checking" property="balance"
                                                 value="0.0" />
    </jsp:useBean>
    ```
  - **`<jsp:setProperty name="mybean" property="*" />`**
    - uloží všechny parametry požadavku jako property
      - jména musejí souhlasit
    ```

Expression Language (EL)

- useBean, (get|set)Property jsou užitečné, ale špatně použitelné
- řešení – Expression Language
  - přímé použití objektů v JSP stránce

    ${item}

- lze použít nejen na JavaBeans
- na vlastnosti bean se odkazuje tečkovou notací
  - ${checking.balance}
  - alternativně lze použít i ${checking["balance"]}
    - vhodné pokud je potřeba zkonstruovat jméno vlastnosti dynamicky

Expression Language (EL)

- EL lze používat s operátory

  ${ 1 + 2 * 3 }

- operátory
  - aritmetické + - * / div
  - relační == eq != ne < lt > gt <= le >= ge
  - logické && and || or ! not
  - empty
  - ternární ${ test ? expr1 : expr2 }
  - lambda ->
  - přiřazení =
  - středník ;

- zakázání EL na stránce

```
<%@ page isELEnabled="false" %>
```

Expression Language (EL)

- odložené vyhodnocení

  #{item}

- může být vyhodnoceno v jiných fázích životního cyklu stránky
  - podle technologie, co stránku používá

Tag libraries

- direktiva taglib
  - „importuje“ knihovnu uživatelsky-definovaných elementů
- př:

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

<tlt:tag>
   body
</tlt:tag>

<tlt:greetings/>
```

- vytváření vlastních tagů
  - potomci javax.servlet.jsp.tagext.TagSupport
  - metody
    - doStartTag(), doEndTag(),...

Vlastní tag

- třída implementující javax.servlet.jsp.tagext.Tag
  - typicky potomci od TagSupport nebo BodyTagSupport
  - předefinování metod
    - doStartTag(), doEndTag(),...
- xml soubor popisující knihovnu tagů
  - mapování jmen na třídy

Vlastní tag – příklad

```
public class ExampleTag extends TagSupport {
  public int doStartTag() throws JspException {
    try {
      JspWriter out = pageContext.getOut();
      out.print("Hello world");
    } catch(IOException e) {
      throw new JspException(e.getMessage());
    }
    return(SKIP_BODY);
  }
}
```

Vlastní tag – příklad

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>vsjava</shortname>
  <urn></urn>
  <info>Our HelloWorld library</info>
  <tag>
    <name>example</name>
    <tagclass>vsjava.jsp.tags.ExampleTag</tagclass>
    <info>HelloWorld tag</info>
    <bodycontent>EMPTY</bodycontent>
  </tag>
  <!-- další tagy... -->
</taglib>
```

Vlastní tag – příklad

```
<html>
  <head>
    <%@ taglib uri="vsjava-taglib.tld" prefix="vsjava" %>
    <title><vsjava:example /></title>
  </head>
  <body>
   <vsjava:example />
  </body>
</html>
```

Propojení JSP a servletů

- servlety
  - výhodné pro složitý kód
  - nevýhodné pro generování HTML
- JSP
  - obráceně

- řešení – použít oboje
  - servlet pro „business“ logiku aplikace
  - JSP pro generování HTML

  - ala MVC
    - model – beans
    - view – JSP
    - controller – servlet

Propojení JSP a servletů

- Příklad

- Servlet

```
ValueObject value = new ValueObject(...);
request.setAttribute("key", value);
RequestDispatcher dispatcher =
        request.getRequestDispatcher("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- JSP Page

```
<jsp:useBean id="key" type="somePackage.ValueObject"
scope="request" />
<jsp:getProperty name="key" property="someProperty" />
```

Propojení JSP a servletů

- Předchozí příklad – sdílení dat mezi servletem a JSP jen v rámci jednoho požadavku

- Servlet

```
ValueObject value = new ValueObject(...);
request.setAttribute("key", value);
RequestDispatcher dispatcher =
        request.getRequestDispatcher("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- JSP Page

```
<jsp:useBean id="key" type="somePackage.ValueObject"
                                                scope="request" />
<jsp:getProperty name="key" property="someProperty" />
```

nebo

```
${key.someProperty}
```

Propojení JSP a servletů

- Sdílení dat v rámci session

- Servlet

```
ValueObject value = new ValueObject(...);
HttpSession session = request.getSession();
session.setAttribute("key", value);
RequestDispatcher dispatcher =
        request.getRequestDispatcher("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- JSP Page

```
<jsp:useBean id="key" type="somePackage.ValueObject"
                                                scope="session" />
<jsp:getProperty name="key" property="someProperty" />
```

Propojení JSP a servletů

- Sdílení dat v rámci aplikace

- Servlet

```
ValueObject value = new ValueObject(...);
getServletContext().setAttribute("key", value);
RequestDispatcher dispatcher =
        request.getRequestDispatcher("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- JSP Page

```
<jsp:useBean id="key" type="somePackage.ValueObject"
                                                scope="application" />
<jsp:getProperty name="key" property="someProperty" />
```

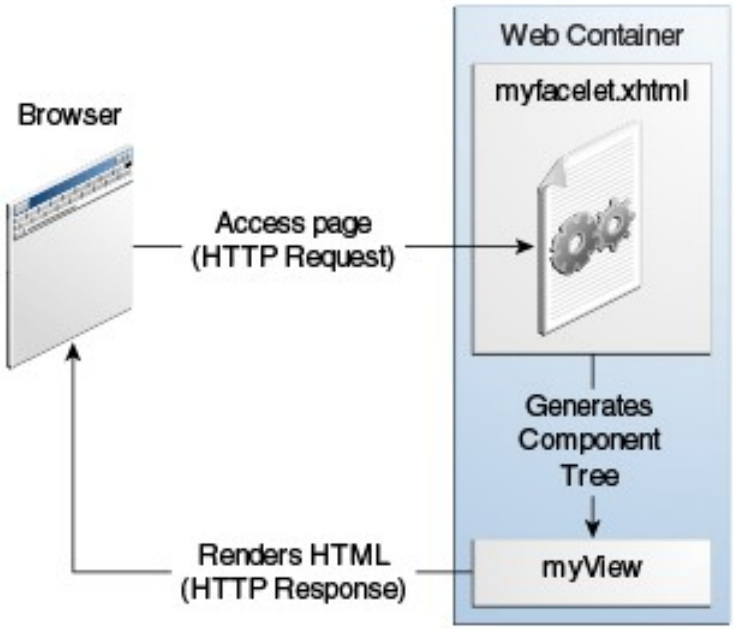
JAVA

JSF

Přehled

- komponentový framework
  - skládání aplikace z hotových komponent
- „náhrada“ za JSP
  - JSP je stále součástí JEE

- podobné jako kombinace JSP a servletů na předchozích slidech



zdroj obrázku: https://javaee.github.io/tutorial/jsf-intro002.html

JSF aplikace

- webová stránka složená z komponent
  - facelets
    - deklarativní jazyk pro definici stránek (šablony)
      - starší verze JSF používaly JSP
    - XHTML, expression language, tag libs
- „managed beans“ s daty a metodami
  - Java Beans
- FacesServlet
  - předdefinovaný servlet
  - požadavky směřují na něj

Facelets

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head> <title>Facelets Hello Greeting</title>
  </h:head>
  <h:body>
    <h:form>
      <h:graphicImage url="#{resource['images:duke.waving.gif']}"
                                        alt="Duke waving his hand"/>
      <h2>Hello, my name is Duke. What's yours?</h2>
      <h:inputText id="username" title="My name is: "
                   value="#{hello.name}" required="true"
                   requiredMessage="Error: A name is required."
                   maxlength="25" />
      <p></p>
      <h:commandButton id="submit" value="Submit"
                           action="response"> </h:commandButton>
      <h:commandButton id="reset" value="Reset" type="reset">
      </h:commandButton>
      </h:form>
    ...
```

Managed beans

```
@Named
@RequestScoped
public class Hello {

    private String name;

    public Hello() {
    }

    public String getName() {
        return name;
    }

    public void setName(String user_name) {
        this.name = user_name;
    }
}
```

```
@SessionScoped
@ApplicationScoped
```

Servlet mapping

- web.xml

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>

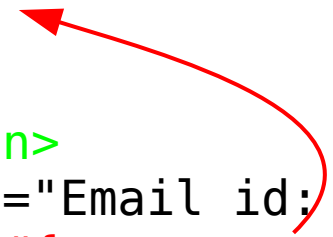
<welcome-file-list>
  <welcome-file>index.xhtml</welcome-file>
</welcome-file-list>
```

Skládání komponent

- tvorba komponent (šablon) z existujících

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:composite="http://xmlns.jcp.org/jsf/composite"
  xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>This content will not be displayed</title>
    </h:head>
    <h:body>
        <composite:interface>
            <composite:attribute name="value" required="false"/>
        </composite:interface>

        <composite:implementation>
            <h:outputLabel value="Email id: "></h:outputLabel>
            <h:inputText value="#{cc.attrs.value}"></h:inputText>
        </composite:implementation>
    </h:body>
</html>
```



Converters

```
<h:outputText value="#{cashierBean.shipDate}">
    <f:convertDateTime type="date" dateStyle="full" />
</h:outputText>

<h:outputText value="#{cart.total}">
    <f:convertNumber currencySymbol="$" type="currency"/>
</h:outputText>
```

- NumberConverter
- DateTimeConverter
- EnumConverter
- BooleanConverter
- ShortConverter
- …

Listeners

```
<h:inputText id="name"
             size="30"
             value="#{cashierBean.name}"
             required="true"
             requiredMessage="#{bundle.ReqCustomerName}">
    <f:valueChangeListener type="my.app.listeners.NameChanged" />
</h:inputText>


<h:commandLink id="Duke" action="bookstore">
    <f:actionListener type="my.app.listeners.LinkBookChange" />
    <h:outputText value="#{bundle.Book201}"/>
</h:commandLink>
```

Validators

```
<h:inputText id="quantity" size="4" value="#{item.quantity}">
    <f:validateLongRange minimum="1"/>
</h:inputText>
<h:message for="quantity"/>
```

- LengthValidator
- RequiredValidator
- RegexValidator
- ...

JSF

- ...