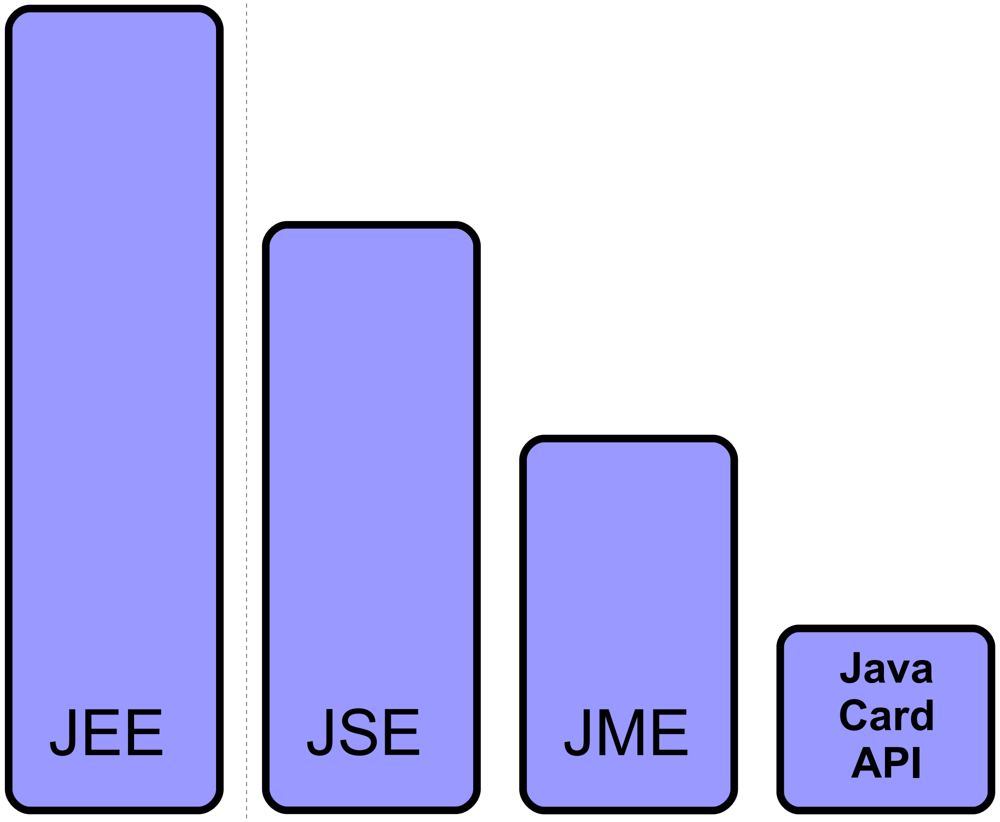
JEE

Web applications
Servlets, JSP, JSF

JEE

- web applications
  - servlets
  - JSP
  - JSF
  - …
- web services
- dependency injection
- EJB
- security
- persistency
- …



Overview

- most of current web pages are *dynamic*
  - technologies and laguages – CGI, PHP, ASP,...
    - server-side dynamicity
- core Java-based technologies

  servlets, Java Server Pages, Java Server Faces
- Servlet
  - a program in Java
  - runs in a server (Java web container)
  - serves requests from a client (browser)
- JSP
  - allows for Java code (plus other elements) directly in HTML source
- JSF
  - a combination of servlets and templates

Overview

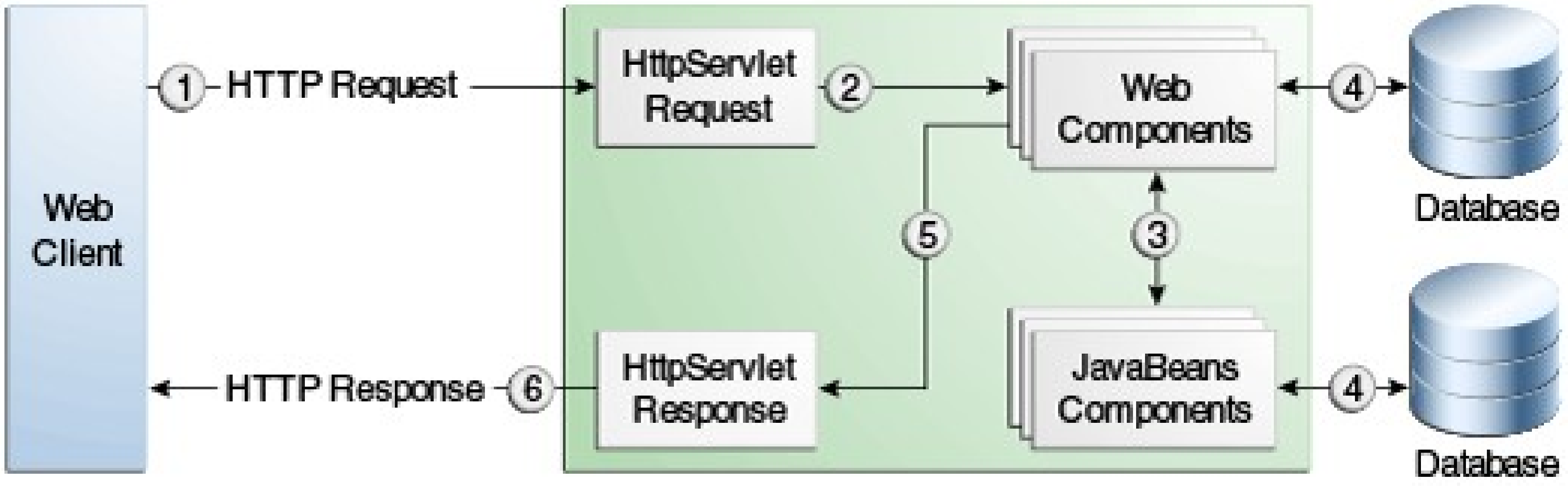


image source: https://eclipse-ee4j.github.io/jakartaee-tutorial/webapp001.html

HTTP

- multiple version
  - 1.0, 1.1 textual
  - 2.0 binary
- request

method

path within the server

version

```
GET /articles/article.html HTTP/1.1
Host: www.articles.com
Connection: keep-alive
Cache-Control: no-cache
Pragma: no-cache
Accept: text/html,application/xhtml+xml,application/xml;
                                         q=0.9,*/*;q=0.8
```

other headers

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

HTTP

- response

protocol version of the response

error code

```
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

content...
```

other headers

- error codes
  - 1xx informational
  - 2xx success
  - 3xx redirection
  - 4xx client errors
  - 5xx server errors

JAVA

Servlets

Servlet structure

- API
  - javax.servlet
  - javax.servlet.http
- inteface **javax.servlet.Servlet**
  - every server must implement it
  - methods

```
• 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();
```

Servlet structure

- the Servlet interface is not typically implemented directly but via the class **javax.servlet.http.HttpServlet**
  - **`protected void service(HttpServletRequest req, HttpServletResponse resp)`**
    - receives an http request
    - calls a particular **`do<something>()`** method
    - typically it is not overridden
      - the **`do<something>()`** methods are overridden
  - **`void doGet(HttpServletRequest req, HttpServletResponse resp)`**
    - serving an http GET request
  - other “do” methods
    - doPost, doDelete, doHead, doPut, doOptions, doTrace
    - the same parameters as **`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

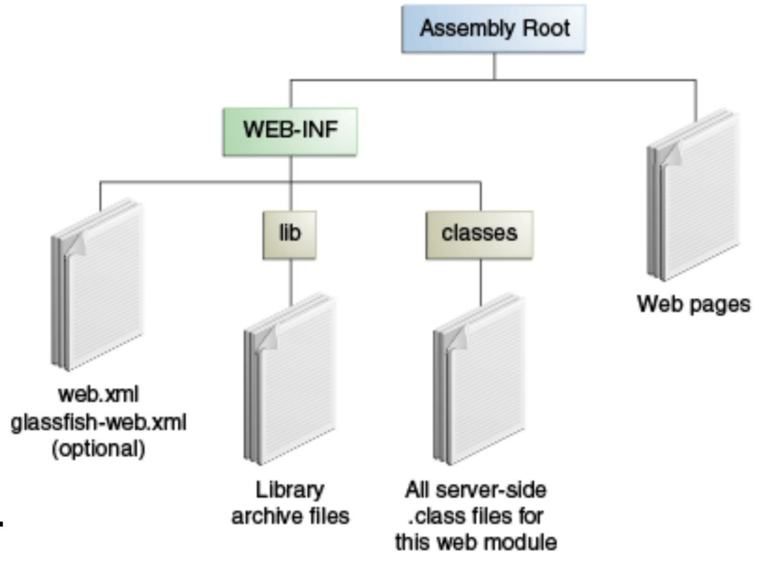
```
<?xml version="1.0" encoding="ISO-8859-2"?>

<!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>
```

- or directly in the code

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

Servlet on a server



- directory structure
  - ....webapps/*app-name*/
    - META-INF/ – manifest
    - context.xml – applicatior
    - WEB-INF/
      - classes/ – compiled classes
      - lib/ – jar files
      - web.xml
    - static pages, images,....
- the server forbids direct access to the WEB-INF directory
- exact placement of an application depends on a particular server

image source: https://eclipse-ee4j.github.io/jakartaee-tutorial/packaging003.html

Servers for deployment

- Tomcat
  - http://tomcat.apache.org/
  - servlet container
  - “instalation“ of a servlet
    - copy it to the webapps directory and restart
    - use the Tomcat manager
      - also a servlet
- GlassFish
  - https://eclipse-ee4j.github.io/glassfish/
  - not only for servlets
  - “instalation” of a servlet
    - copy it to the *domain-dir*/autodeploy/
    - the as-admin tool
- …

WAR

- Web ARchive (WAR)
  - distributing web-applications, installing on a server,...
  - a JAR file with a web-app directory structure
    - i.e. WEB-INF, web.xml, classes....
- creation
  - manually using jar or zip
  - via Ant
    - the task war
      - e.g.:

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

Servlet life-cycle

- void init(ServletConfig config) throws ServletException
  - called automatically during the servlet start
  - called just once
  - e.g.

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

  - „init“ parameters can be set in 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>
```

Servlet life-cycle

- or, the parameters can be set directly in the code
  - ideal for default values

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

Servlet life-cycle

- void init() throws ServletException
  - init without parameters
  - should be overridden if no parameters are necessary
    - called automatically from init(ServletConfig)
- public void destroy()
  - called during the servlet termination
    - when the servlet is terminated
    - when the servlet is removed from memory
    - when the servlet is terminated from the manager

HttpServletRequest

- represent an http request
  - 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

- ...continuation
  - Cookie[] getCookies()
  - HttpSession getSession()
  - HttpSession getSession(boolean create)
- Cookie
  - constructor
    - Cookie(String name, String value)
  - methods
    - (get|set)Name, (get|set)MaxAge, (get|set)Value
- HttpSession
  - the server automatically decides whether to keep a session via cookies or via URL
  - methods
    - getId, (get|set)Attribute, setMaxInactiveInterval, invalidate

HttpServletResponse

- a set of constants for return codes of responses
  - SC_OK (200), SC_NOT_FOUND (404),...
- methods
  - setContentType, setContentEncoding

  - ServletOutputStream getOutputStream()

  - void setStatus(int sc)
  - void setHeader(String name, String value)

  - String encodeURL(java.lang.String url)
    - adds a session identification to URL
    - if a session is used, all URLs in a resulting page should "go" through this method
  - void addCookie(Cookie cookie)

JAVA

JSP

JSP – overview

- mix of HTML and Java (and special tags)
- JSP code is inserted to HTML via

```
<% JSP code %>
```

- e.g.:

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

- in the WAR structure, JSP pages are in the same place as static elements
  - i.e. not in WEB-INF

JSP – overview

- steps in serving a JSP page
  - during first access, the JSP page is transformed to Java code
    - the resulting servlet is compiled and .class file(s) stored in a special directory
  - a new instance of the servlet is created
  - and then continue as for regular servlet
- during transforming JSP -> Java
  - code between <% %> is “copied”
  - html code is transformed to out.print(“.......“)
- kinds of JSP elements
  - scripting elements
  - directives
  - JSP actions (tags)
  - own (developer-defined) actions (tags)

Scripting elements

- declaration
  - enclosed in `<%!      %>`
  - one or more declaration in Java
  - runs only during first access or when the JSP container re-initializes the page
- expression
  - enclosed in `<%=      %>`
  - a single expression in Java
  - a result is the expression value
  - executed during each access
- scriptlet
  - enclosed in `<%      %>`
  - Java code
  - executed during each access

Examples

```
<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>
```

Examples

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

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

Variables in JSP

- created in JSP declaration
  - valid in the whole JSP page
    - defined at the class level
  - created and initialized during initialization of the servlet (which is created from JSP)
- created in scriptlets
  - valid in the particular scriptlet
    - defined on the method level
  - created and initialized during each access

- no method can be defined in scriptlets
  - as the code in scriptlets is inside a method (created during page transformation into the servlet)

Comments in JSP

- Java comments in scriptlets
  - // comment
  - /* comment */
- JSP comment
  - <%-- comment --%>
  - other JSP elements can be commented out

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

- HTML comments
  - <!-- comment -->
  - they will be in the resulting page

Implicit objects in JSP

- automatically created objects
  - can be used in expressions and scriptlets
  - cannot be used in declareation
    - as they are created later

- request
  - an instance of HttpServletRequest
- response
  - an instance of HttpServletResponse
- out
  - output to the resulting page
  - an instance of jsp.JspWriter
- session
  - an instance of HttpSession

Implicit objects in JSP

- application
  - am instance of ServletContext
- config
  - an instance of ServletConfig
- page
  - a reference to the currently processed page
- pageContext
  - an instance of PageContext
  - an environment in which all pages runs

Directives

- influence how the servlet is generated from JSP
- 3 directives
  - page
  - include
  - taglib
- usage
  - <%@ directive attribute1=“value1” .....
    attributeN=“valueN” %>
- include
  - <%@ include file=“relative URL” %>
  - inserts the file at **compile**-time
- taglib
  - “imports“ a user-defined element library
  - <%@ taglib uri=“TLD file” prefix=“prefix” %>

The page directive

- multiple usage
- parameters
  - import
  - errorPage, isErrorPage
  - session
  - info
  - language
  - contentType
  - isThreadSafe
  - buffer
  - autoFlush

The page directive

- import
  - import of classes and packages
  - <%@ page import=package.class“ %>
- errorPage
  - specifies the page, which is used for processing exception non-handled in the current page
  - <%@ page errorPage=“relative URL” %>
- isErrorPage
  - if the current page is error one
    - false by default
- session
  - whether a session should be kept for the page
  - <%@ page session=“false” %>
- info
  - page information – typically author, copyright,...
  - <%@ page info=“Petr, 2013 ” %>

The page directive

- language
  - (programming) language of JSP
  - <%@ page language=“java” %>
- contentType
  - default value text/html; charset=iso-8859-1
  - <%@ page contentType=“ text/plain; charset=utf-8” %>
- autoFlush
  - default true
  - if false, than the buffer is not flushed automatically and IOExcpetion is thrown
    - JspWriter.flush()
  - <%@ page autoFlush=“false” %>
- extends
  - a direct parent for the generated servlet
  - <%@ page extends=“class” %>

JSP actions (tags)

- jsp:include
  - inserts file or result to JSP
    - static file (e.g. html) is inserted
    - dynamic file is executed (e.g. jsp) a and result is inserted
  - executed each access
  - <jsp:include page="hello.jsp"/>
- jsp:param
  - passing parameters for jsp:include

  -
    ```
    <jsp:include page="scripts/login.jsp">
       <jsp:param name="username" value="petr" />
    </jsp:include>
    ```
- jsp:forward
  - forwarding current request to other JSP

  -
    ```
    <jsp:forward page="orderError.jsp" >
      <jsp:param name=„errorType“ value=„badAmount“ />
    </jsp:forward>
    ```

JSP action (tags)

- using JavaBeans
  - jsp:useBean
    - creation an instance
  - jsp:getProperty
    - reading a property
  - jsp:setProperty
    - setting a property
- e.g.:

  -
    ```
    <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="*" />`
    - stores all request parameters as properties
      - names must be the same

Expression Language (EL)

- useBean, (get|set)Property are useful but not very user-friendly
- solution – Expression Language
  - direct usage of objects in a JSP page

    ```
    ${item}
    ```

- can be used not only for JavaBeans
- bean's properties are accessed via “dot” notation
  - ${checking.balance}
  - alternatively ${checking["balance"]} can be used
    - suitable if the property name has to be constructed dynamically

Expression Language (EL)

- EL can be used with operators

  ${ 1 + 2 * 3 }

- operators

– arithmetic	+ - * / div
– relational	== eq != ne < lt > gt <= le >= ge
– logic	&& and || or ! not
– empty	
– ternary	${ test ? expr1 : expr2 }
– lambda	->
– assignment	=
– semicolon	;

- forbidding EL in a page

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

Expression Language (EL)

- deferred evaluation

  #{item}

- can be evaluated at other phases of a page lifecycle
  - as defined by whatever technology is using the expression

Tag libraries

- the taglib directive
  - “imports” a library with user-defined elements
- e.g.:

```
<%@ taglib uri=“/tlt” prefix=„tlt“ %>

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

<tlt:greetings/>
```

- creating own tags
  - extending javax.servlet.jsp.tagext.TagSupport
  - methods
    - doStartTag(), doEndTag(),...

Own tag

- a class implementing javax.servlet.jsp.tagext.Tag
  - typically extending TagSupport or BodyTagSupport
  - overridding methods
    - doStartTag(), doEndTag(),...
- an xml file describing the library
  - mapping names to classes

Own tag – example

```
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);
  }
}
```

Own tag – example

```
<?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>
  <!-- next tagy... -->
</taglib>
```

Own tag – example

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

Connecting JSP and servlets

- servlets
  - ideal for complex code
  - not ideal for HTML generation
- JSP
  - vice-versa
- solution – use both
  - servlet for “business” logic of an application
  - JSP for HTML generation
  - similarly to MVC
    - model – beans
    - view – JSP
    - controller – servlet

Connecting JSP and servlets

- Example
- 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" />
```

Connecting JSP and servlets

- the previous example – sharing data between the servlet and JSP only within a single request

- 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}
```

Connecting JSP and servlets

- Sharing data in the session scope

- 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" />
```

Connecting JSP and servlets

- Sharing data in the application scope

- 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

Overview

- a component framework
  - composing applications from reusable components
- „replacement“ for JSP
  - JSP is still part of JEE

- similar to the combination of JSP and servlets on the previous slides

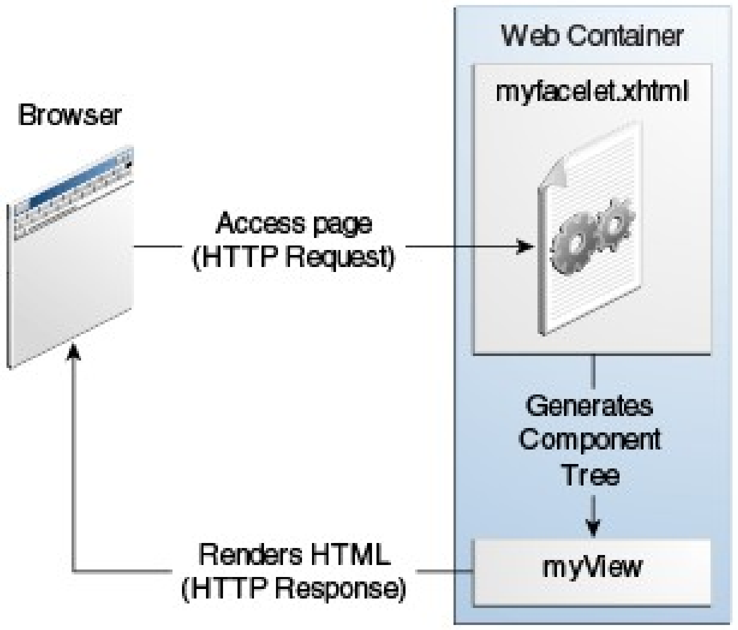


image source: https://javaee.github.io/tutorial/jsf-intro002.html

JSF application

- a web page composed of components
  - facelets
    - a declarative language for page definition (templates)
      - older versions of JSF used JSP
    - XHTML, expression language, tag libs
- managed beans with data and methods
  - Java Beans
- FacesServlet
  - predefined servlet
  - requests mapped to it

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>
```

Composing components

- creating components (templates) from existing ones

```
<!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

- ...