REALDOLMEN

Java Servlets and JSP Programming

Java Servlets and JSP Programming

Introduction

Java Servlets and JSP Programming

Learn how to build dynamic web applications with Java using Servlets and JSP

Agenda

- J2EE Overview
- Server Setup and Configuration
- Servlets
- JavaServer Pages
- Architectural Overview
- Additional Concepts

Java Servlets and JSP Programming

J2EE Overview

What is J2EE?

- Challenges for enterprise applications
  - Portability
  - Time-to-Market
  - Diverse Environments
  - Scalability
  - Reliable
  - Secure
  - Easy to maintain
  - Flexible / Easily integrated to older systems
  - High Performance
- Need for a platform that addresses all these issues...

What is J2EE?

- J2EE is an open and standard based platform for developing, deploying and managing n-tier, web-enabled, server-centric, and component-based applications

The Java 2 Platform

- J2ME
  - Java version for small devices such as PDA’s, cell phones, etc, which are typically constrained in memory space and processing power
- J2SE
  - Standard Edition, for desktop and workgroup server environments, which require full feature functionality, including a rich graphical interface

The Java 2 Platform

- J2EE
  - The Java platform for developing and deploying enterprise applications which are typically transactional, reliable and secure
  - Is built on top of J2SE

REALDOLMEN

What do I get from J2EE?

- API and specifications
- Platform for development and deployment
- Reference implementation (standards)
- Compatibility Test Suite
- J2EE brand
- Blueprints
  - Documentation and example implementations with guidelines, best practices, and design patterns for developing J2EE applications

REALDOLMEN

Trends

- Movement from single-tier, two-tier to multi-tier architecture
- Movement from monolitical model to object based application model
- Movement from application-based clients to html-based clients

Single VS Multi-Tier

Single Tier

- No separation between presentation, business logic, database
- Hard to maintain

Multi-Tier

- Separation among presentation, business logic, database
- More flexible to change, i.e. presentation can change without affecting other tiers

Monolytical VS Object-based

Monolytical

- 1 binary file
- Recompiled, relinked, redeployed every time there is a change

Object-based

- Pluggable parts
- Reusable
- Enables better design
- Easier update
- Implementation can be separated from the interface
- Only the interface is published

Open Issues

- Complexity at middle-tier still remains
- Duplicate system services still need to be provided for the majority of enterprise applications
  - Concurrency control, transactions
  - Load balancing, security
  - Resource management, connection pooling
- How to solve this problem?
  - Commonly shared container that handles the above system services
  - Propriety vs open-standard based

Propriety Solution

- Use “component and container” model in which the container provides system services in a well-defined but with propriety manner
- Problem of propriety solution: Vendor lock-in
- Example: .Net

Open-standard Solution

- Use “component and container” model in which container provides system services in a well-defined and as industry standard manner
- J2EE is such a standard solution
- J2EE also provides portability of code because it is based on Java technology and standard based Java programming APIs

Why J2EE?

- Platform value to developers
  - Can use any J2EE implementation for development and deployment
  - Vast amount of community resources (books, articles, quality code, etc)
  - Can use of the shelf 3rd party business components
  - Vendors work together on specifications and then compete in implementations
  - Freedom to innovate while maintaining the portability of applications
  - Do not have to create/maintain their own APIs

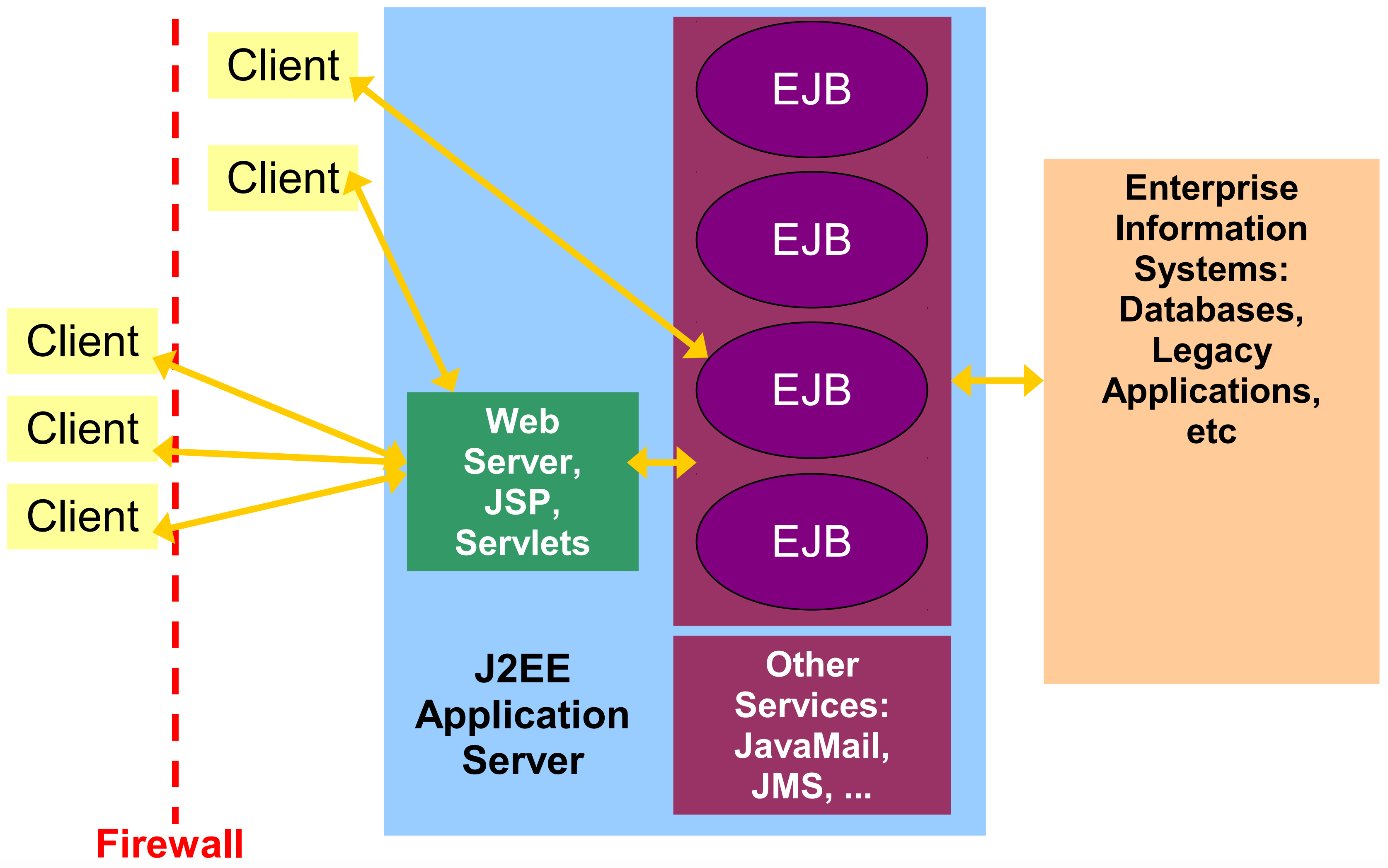
Why J2EE?

- Platform value to business customers
  - Application portability
  - Many implementation choices are possible based on various requirements
    - Price (free to high-end), scalability (single CPU to cluster), reliability, performance, tools, and more
  - Large developer pool

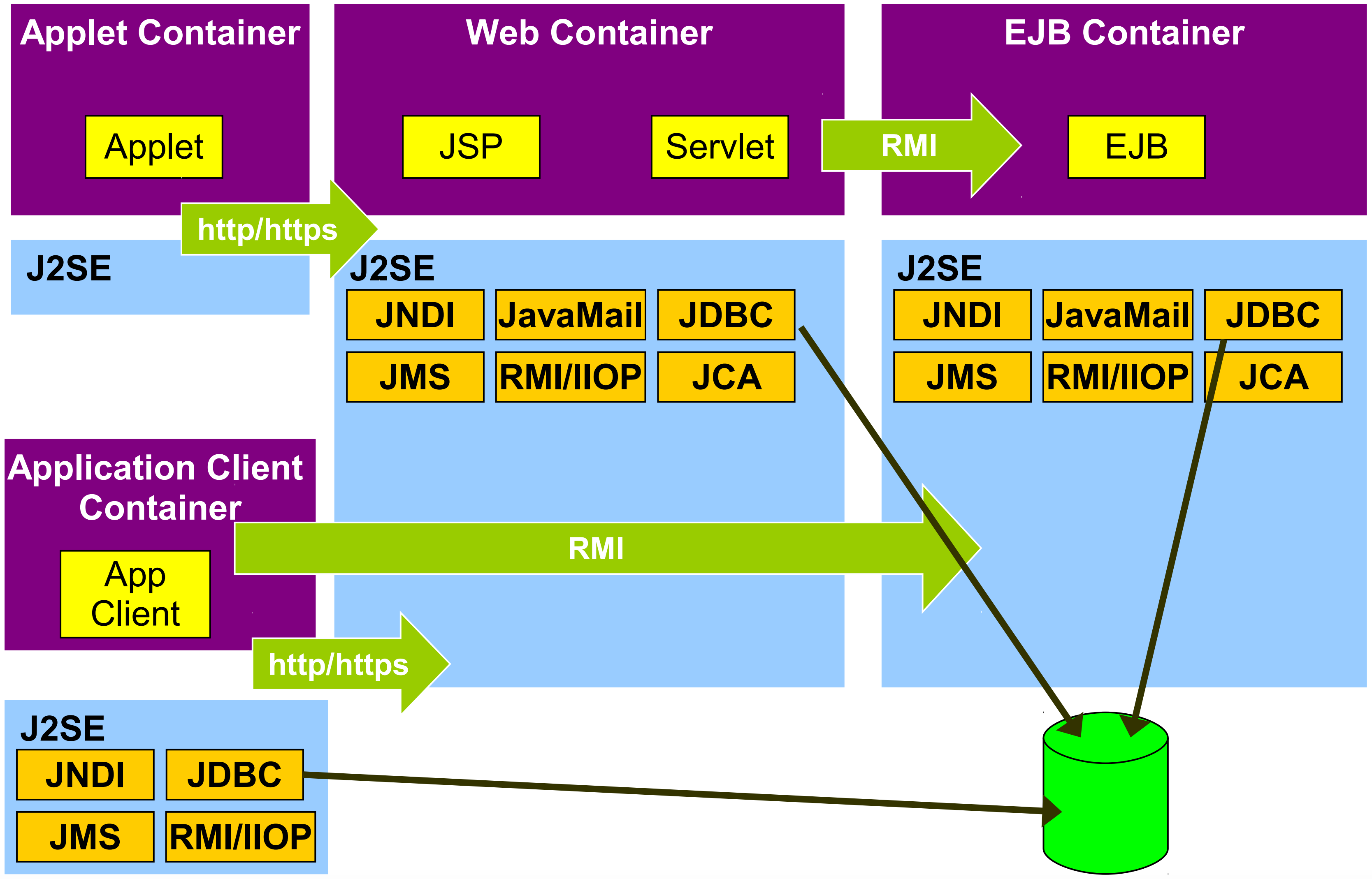
Overview of the J2EE API's

- J2SE
- *Servlets*
- *JSP*
- EJB
- JCA
- JavaMail
- JMS, ...
- Management, *deployment*, web services, ...

J2EE: End-to-End Architecture


Client
Client
Client
Client
Client
Firewall
Web Server, JSP, Servlets
EJB
EJB
EJB
EJB
J2EE Application Server
Other Services: JavaMail, JMS, ...
Enterprise Information Systems: Databases, Legacy Applications, etc

Containers and Components



Containers and Components

Containers handle

- Concurrency
- Security
- Availability
- Scalability
- Persistence
- Transactions
- Management

Components handle

- Presentation
- Business logic

Containers and Components

- Containers do their work invisibly
  - No complicated APIs
  - Control by interposition
- Containers implement the J2EE standards
  - Look the same to components
  - Vendors making the containers have great freedom to innovate

J2EE Development Lifecycle

- Write and compile the component code
  - Servlet, JSP, EJB, ...
- Write deployment descriptors for components
- Assemble components into ready-to-deploy package
- Deploy the package on the server

Deployment Descriptor

- Gives the container instructions on how to manage and control behaviors of the J2EE components
  - Transaction
  - Security
  - Persistence
- Allow declarative customization (as opposed to programming customization)
  - XML file
- Enables portability of code

J2EE Anatomies

- 4 tier J2EE applications
  - HTML client, JSP/Servlets, EJB, JDBC
- 3 tier J2EE applications
  - HTML client, JSP/Servlets, JDBC
- 3 tier J2EE applications
  - EJB standalone application, EJB, JDBC
- B2B J2EE applications
  - J2EE platform to J2EE platform through the exchange of JMS or XML based messages

J2EE Reference

- J2EE has a reference implementation
  - Validates specifications
  - Fully compliant
  - Fully functional
  - Not commercial quality, used for prototyping
- CTS: Compatibility Test Suite
  - Controls compatibility of the J2EE application to the standards
  - WORA (write once, run anywhere) paradigm
- J2EE Application Verification Kit
  - Controls portability of the J2EE application

J2EE BluePrint

- Best practice guidelines, design patterns, design principles
  - MVC pattern
- Covers all tiers
  - Client
  - Web
  - Business Logic (EJB)
  - Database
- Sample code
  - Java Pet Store
  - Java Adventure Builder

Why J2EE for Web Services?

- Web services is just one of many service delivery channels of J2EE
  - No architectural change is required
  - Existing J2EE components can be easily exposed as web services
- Many benefits of J2EE are preserved for web services
  - Portability, reliability, scalability
  - No single-vendor lock-in

J2EE Summary

- J2EE is the platform of choice for developing and deploying n-tier, web-based, transactional and component-based enterprise applications
- J2EE is a standard-based architecture
- J2EE is all about community
- J2EE evolves according to the needs of the industry

Java Servlets and JSP Programming

Server Setup and Configuration

Tomcat

- Tomcat is the most popular reference implementation for JSP and Servlets
- Tomcat is a Java Web Server
- You need this to run your servlets and JSP pages

Tomcat Installation

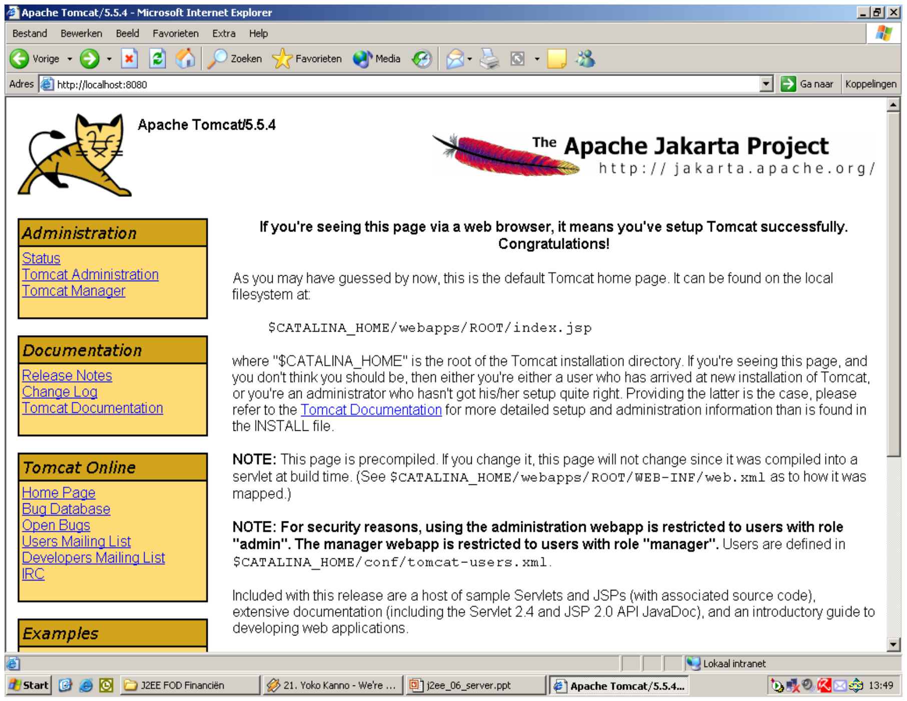
- Go to *http://jakarta.apache.org*
- Follow the download links
- Download the most recent binary for Tomcat
- The Windows versions can either be a .zip or an .exe file

Tomcat Installation

- Run the .exe file / extract the .zip
- Install / copy into a directory called /tomcat
  - The default path name is too long and contains embedded spaces
  - If you use the default installation path name, you have to quote it everytime you use it
- You can start and stop Tomcat from your IDE, using the commandline or from the Control Panel if you installed it (Services)
- Setting up the server in your IDE

Checking the Installation

- Browse to *http://localhost:8080*
- If everything went smoothly, you should see the following:


Apache Tomcat/5.5.4 - Microsoft Internet Explorer
Bestand Bewerken Beeld Favorieten Extra Help
Vorige Zoeken Favorieten Media
Adres http://localhost:8080
Ga naar Koppelingen
The Apache Jakarta Project
http://jakarta.apache.org/
Lokaal intranet
Start J2EE FOD Financiën 21. Yoko Kanno - We're ... j2ee_06_server.ppt Apache Tomcat/5.5.4...
13:49

Apache Tomcat/5.5.4

Administration
Status
Tomcat Administration
Tomcat Manager

Documentation
Release Notes
Change Log
Tomcat Documentation

Tomcat Online
Home Page
Bug Database
Open Bugs
Users Mailing List
Developers Mailing List
IRC

Examples

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.jsp
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: This page is precompiled. If you change it, this page will not change since it was compiled into a servlet at build time. (See `$CATALINA_HOME/webapps/ROOT/WEB-INF/web.xml` as to how it was mapped.)

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation (including the Servlet 2.4 and JSP 2.0 API JavaDoc), and an introductory guide to developing web applications.

REALDOLMEN

Running Examples

- Let’s see if our installation works...
- In the lower left of the page, in the menu, there are two links to the example servlets and JSP pages
- Try them out :-)

Changing the Default Port

- By default, Tomcat runs on port 8080, so users have to type the port number 8080 after the domain name
- The default HTML port is 80
- We can change the port for Tomcat to 80 in the server.xml file, located in the conf directory of the installation
- We need to restart Tomcat for the changes to take effect
- Note:
  - Do not change the port if you are planning to run other web servers together with Tomcat...

Adding the “admin” Package

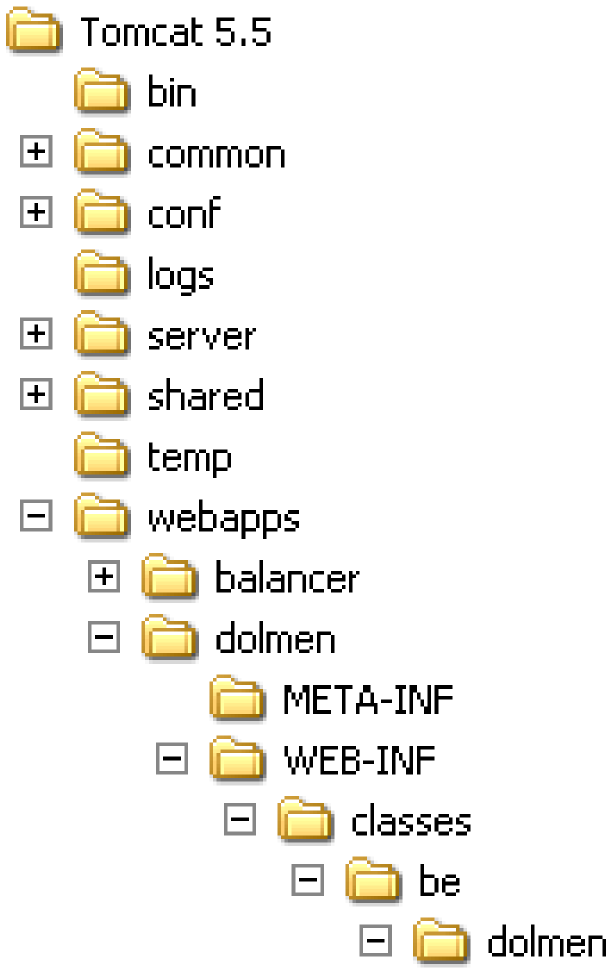
- From the Tomcat website, also download the admin package
- Unzip the package, and copy the files from the directories to the same directories of your Tomcat installation directory
- Restart Tomcat

Tomcat Directories

- bin
  - Startup and Shutdown scripts
- common/classes
  - Unpacked classes which are global to the web applications
- common/lib
  - Common classes
- conf
  - Configuration files
- server
  - Tomcat archive files
- logs
  - Tomcat Log files
- webapps
  - Servlets and JSP applications
- work
  - Resulting servlets from JSP pages translation

A Web Application Directory Structure

- Web applications have a specific directory structure
- In the web application directory, you will see a directory WEB-INF
- Inside WEB-INF, you have a directory classes
- WEB-INF also contains a file web.xml



REALDOLMEN

Server Setup and Configuration Summary

- We are ready to start implementing Java Web Applications!

Exercise

- Set up your server and your development environment to be able to start creating Java web applications

Java Servlets and JSP Programming

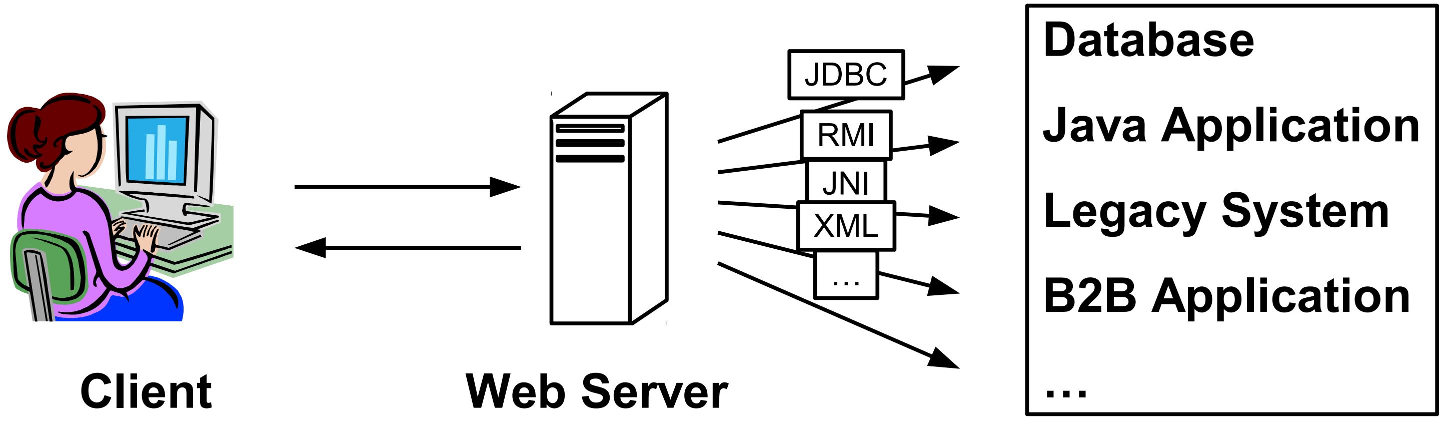
Servlets

What are Servlets?

- Java technology objects which extend the functionality of an HTTP Server
  - Comparable to Netscape’s NSAPI, Microsoft’s ISAPI, or Apache Modules
- Platform & Server Independent
- Why Servlets ?
  - HTTP is the universal transport through restrictive firewalls
  - Browsers, Web Applications, Java technology-based applets and Applications, and other programs can all use HTTP
  - Any kind of data can be transmitted over HTTP (not just HTML)

A Servlet's Job

- Read explicit data sent by client (form data)
- Read implicit data sent by client (request headers)
- Generate the results
- Send the explicit data back to the client
- Send the implicit data to client (status code and response headers)



REALDOLMEN

Why build Web pages dynamically ?

- The Web page is based on data submitted by the user
  - E.g. results page from search engines and order-confirmation pages at on-line stores
- The Web page is derived from data that changes frequently
  - E.g. a weather report or news headlines page
- The Web page uses information from databases or other server-side sources
  - E.g. an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale

REALDOLMEN

Servlets Are Lightweight

- Servlets can run in the same server process as the host HTTP server
- Can support a higher user load with less machine resources
- Servlets can be loaded from anywhere:
  - Local Filesystem
  - Remote Website

Advantages of Servlets over “Traditional” CGI

- Efficient
  - Threads instead of OS processes, one servlet copy, persistent
- Convenient
  - Lots of high-level utilities
- Powerful
  - Sharing data, pooling, persistence
- Portable
  - Run on virtually all operating systems and servers
- Secure
  - No shell escapes, no buffer overflows
- Inexpensive
  - There are plenty of free and low-cost servers

Web Applications

- A Web Application is a collection of HTML, Servlets, JSPs, supporting libraries, and other files
- Rooted at a particular URL Prefix
- Can exist in a well defined file system structure or in a Web Application Archive (.war)
- All Servlets in a Web Application share a Servlet Context

Web Applications are easy to develop

- Are written in Java programming language
- Use all benefits of the Java language
  - Object-Oriented (OO)
  - Write Once, Run Anywhere (WORA)
    - Same servlet can run on any brand of servlet enabled server from Apache to Zeus
    - Develop on any small desktop machine
    - Deploy on any server
  - Code reuse
  - Use business logic running in the J2EE architecture

Server Requirements

- The server must be running on a java enabled platform (= platform with JVM)
- Server must be servlet enabled
- Some servers are servlet enabled by nature, no additional software needs to be installed
- Other servers must be servlet enabled by installing a ‘servlet engine’

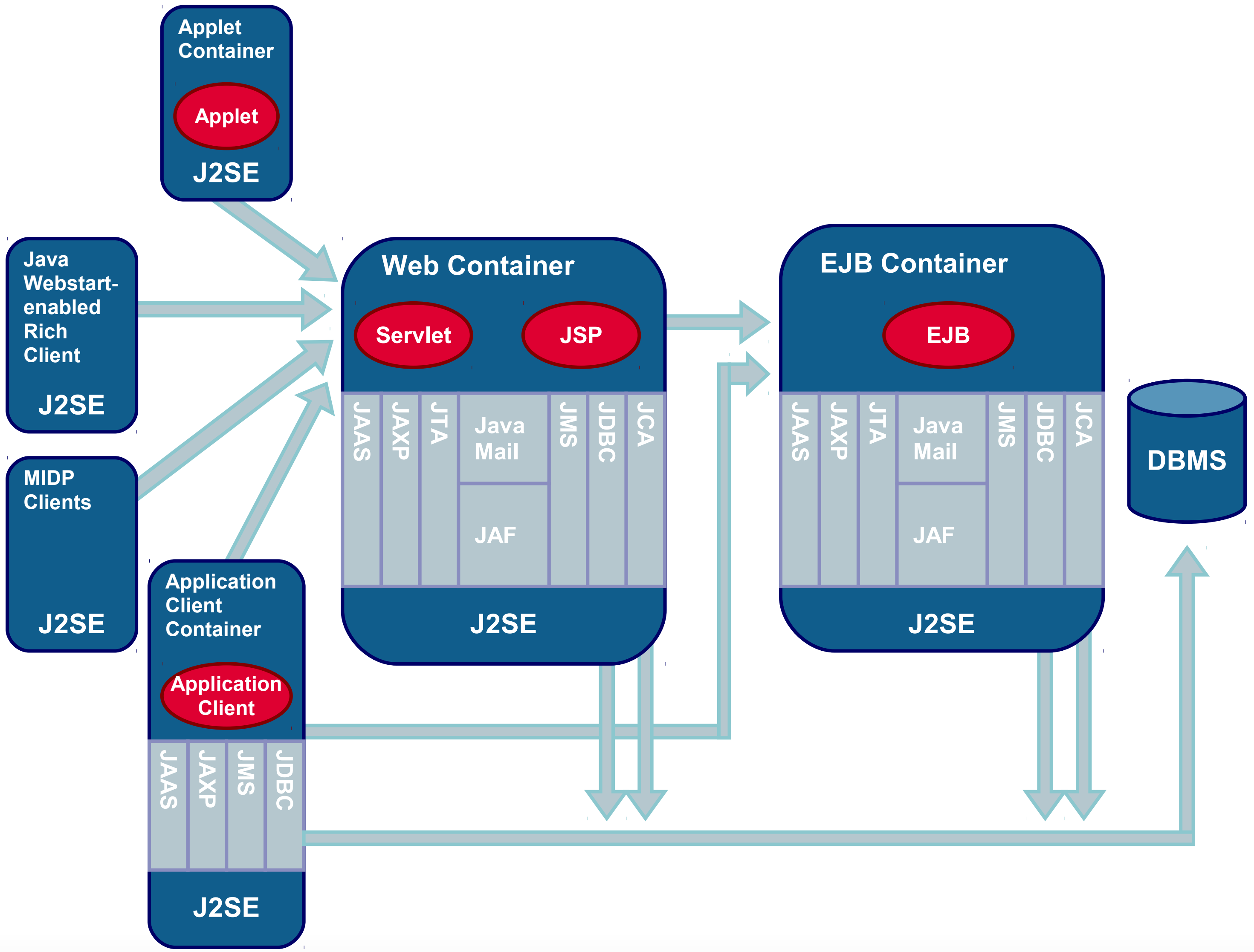
Servlet Engine

- Piece of software that is plugged into the server:
  - Server passes ‘servlet requests’ to the servlet engine
  - Servlet passes the response back to the server
  - Server sends the response back to the user’s browser
- Servlet engine uses the server specific API to talk to the server (NSAPI, ISAPI, …)
- Different engine versions for different servers
- Available servlet engines:
  - ServletExec, JRun, Tomcat, …
- Servlet code does not depend on the type of engine that is being used

Servers Supporting Servlets

- Apache - Tomcat
- ATG - Dynamo
- Caucho - Resin
- Fujitsu - Interstage
- IBM - WebSphere
- Iona - Orbix
- JBoss
- MacroMedia - JRun Server
- New Atlanta Communications - ServletExec
- Novell – exteNd
- ObjectWeb – JOnAS
- Oracle – WebLogic
- Persistence – Power Tier
- Pramati
- SAP AG – Web Application Server
- Sun – Glassfish
- Sybase – EAServer
- Trifork – Enterprise Application Server
- …

Servlets in the Java Platform

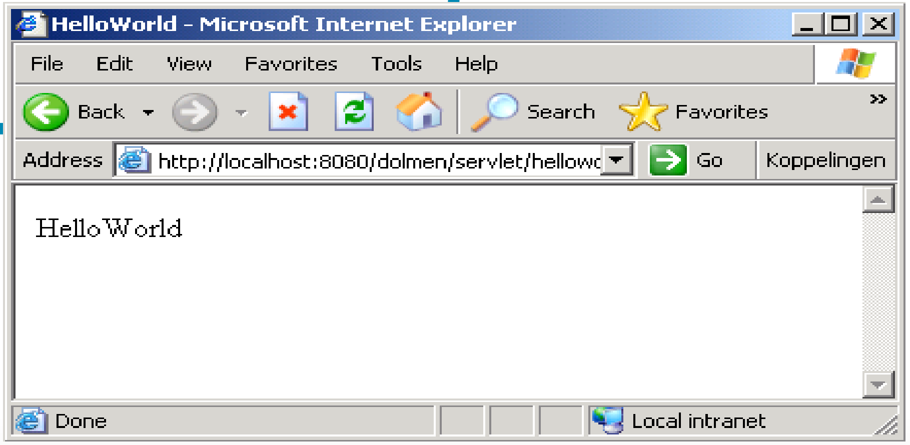


Servlets in the Java Platform

- Servlets are the HTTP speaking middle tier between any kind of client and large Enterprise services exposed via EJBs
- EJBs can only be exposed in environments where RMI works
- EJBs can't be accessed over HTTP only firewalls
- HTML is the perfect lightweight interface to Enterprise Applications

A simple Servlet that generates text

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

public class HelloWorldServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
    // ...
}
```


HelloWorld - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/dolmen/servlet/hellowc
Go Koppelingen
Done
Local intranet

Hello World

- Servlets are implemented as a Java class that extends from HttpServlet

REALDOLMEN

XML based configuration

- Servlets need to be configured in the web application descriptor file
  - /WEB-INF/web.xml

```
<?xml version="1.0" encoding="utf-8"?>
<web-app 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"
        version="3.0">
    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>
            com.realdolmen.servlets.HelloWorldServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/HelloWorldServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Servlets 3.0

- Starting from JEE6, there is a new and more compact way to create a servlet
  - The servlet class implemention is identical
  - It is marked with an annotation, eliminating the need for an web.xml entry
    - In fact, the web.xml file has become optional entirely

```
@WebServlet(urlPatterns = "/myservlet")
public class MyServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException, IOException {
        // ...
    }
}
```

@WebServlet

- This annotation offers all equivalent configuration settings the XML way also offers

```
public @interface WebServlet {
    String name() default "";
    String[] value() default {};
    String[] urlPatterns() default {};
    int loadOnStartup() default -1;
    WebInitParam [] initParams() default {};
    boolean asyncSupported() default false;
    String smallIcon() default "";
    String largeIcon() default "";
    String description() default "";
    String displayName() default "";
}
```

Generating HTML

- Set the Content-Type header
  - Use response.setContentType()
- Output HTML
  - Be sure to include the DOCTYPE
- Use an HTML validation service
  - http://validator.w3.org
  - http://www.htmlhelp.com/tools/validator
  - If your servlets are behind a firewall, you can run them, save the HTML output and use a file upload form to validate

A simple Servlet that generates HTML

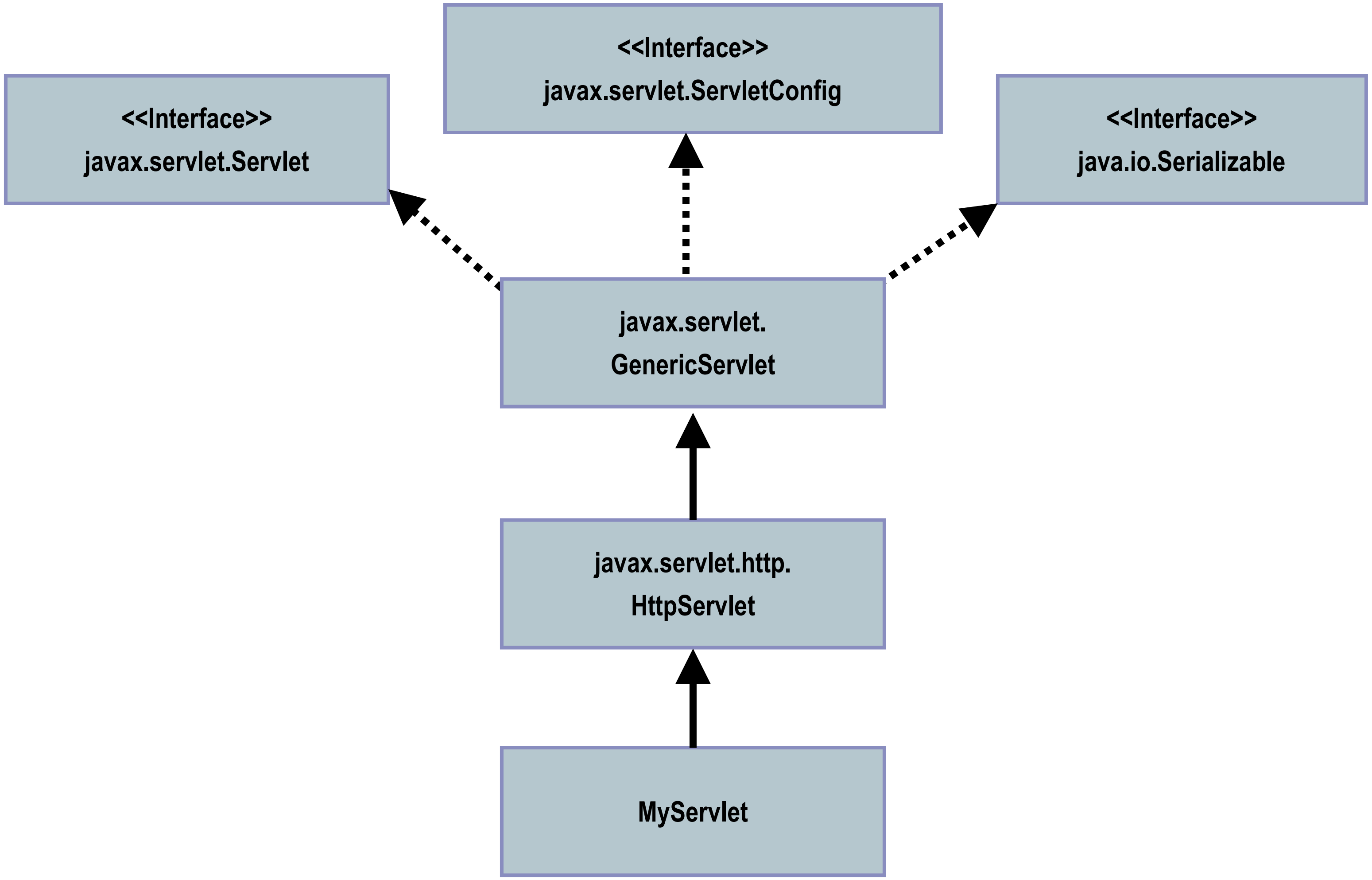
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class HelloWorldServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html";
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Hello World</title></head>");
        out.println("<body bgcolor=\"#ffffff\">");
        out.println("<p>Hello World</p>");
        out.println("</body></html>");
    }
    // ...
}
```

Exercise

- Create a HelloWorldServlet
  - The servlet should print “Hello, World!” in an HTML page

REALDOLMEN

The Servlet Framework



REALDOLMEN

The Servlet Life Cycle

- init()
  - Executed once when the servlet is first loaded by container (not called for each request)
- service()
  - Called in a new thread by server for each request
  - Dispatches to doGet, doPost, etc.
  - Do not override this method
- doGet(), doPost(), doXXX()
  - Handles GET, POST, etc. requests
  - Overrides these to provide desired behavior
- destroy()
  - Called when container deletes servlet instance (not called for each request)

Initializing a Servlet

- Called by the servlet container to indicate to a servlet that the servlet is being placed into service
  - Once after instantiating the servlet
- Use ServletConfig.getInitParameters() to read initialization parameters
  - Set init parameters in web.xml

```
public void init(ServletConfig cnf) throws ServletException {
    super.init(cnf);
    String message = cnf.getInitParameter("message");
 }
```

  - This is typically done to allow deploy-time configuration of a servlet

Providing init parameters

- Init parameters can be specified in the web.xml file

```
<servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>
        com.realdolmen.servlets.HelloWorldServlet
    </servlet-class>
    <init-param>
        <param-name>message</param-name>
        <param-value>Hello!</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/HelloWorldServlet</url-pattern>
</servlet-mapping>
```

Destroying a Servlet

- Called by the Servlet container to indicate that the servlet is being taken out of service
  - The destroy method is only called once all threads within the servlet's service method have exited or after a timeout period has passed
  - The method gives the servlet an opportunity to clean up any resources that are being held

```
public void destroy() {
    // Close the connection (add try/catch)
    // Usually not the place to manage connections
    if(dbConn != null) {
        dbConn.close();
        dbConn = null;
    }
}
```

Handling Requests

- Requests come in through the *service()* method
- By default the *service()* method dispatches the request to the corresponding method:
  - GET requests *doGet()* method
  - POST requests *doPost()* method
  - PUT request *doPut()* method
  - DELETE requests *doDelete()* method
- Don’t override the *service()* method, instead override the different doXXX() methods
- *doGet()* and *doPost()* are most frequently used

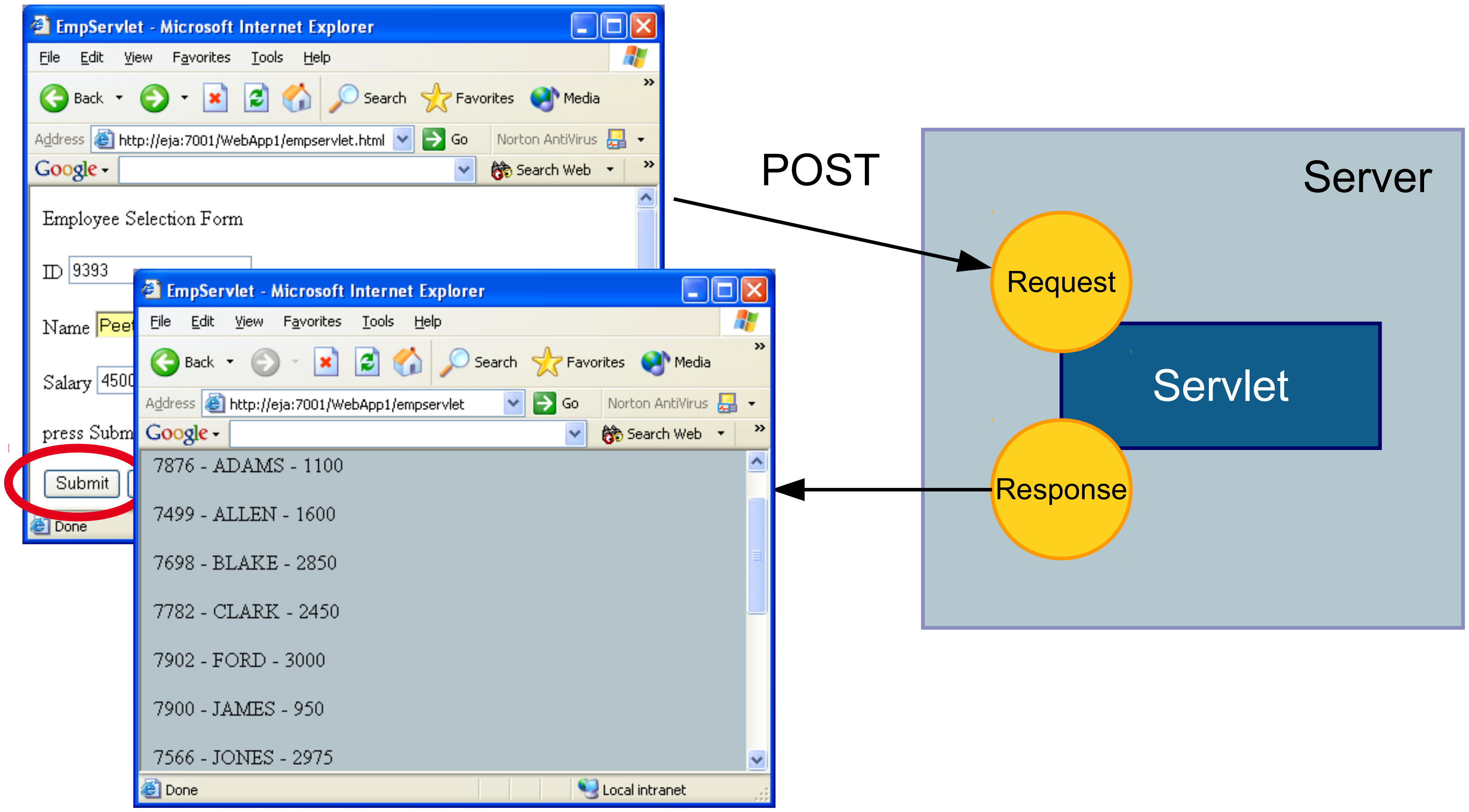
Anatomy of a Request

- A client makes a request on a server using a specific URL:

```
http://localhost:8080/MyWeb/HelloWorld
http://localhost:8080/MyWeb/HelloWorld?name=duke
http://localhost:8080/MyWeb/HelloWorld?name=duke%20java
    &email=duke@sun.com
```

  - The request is resolved to a servlet by the container
  - The servlet’s service method is called with a Request and Response object
  - The servlet provides a response to the request

Requests in Action


EmpServlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
http://eja:7001/WebApp1/empservlet.html
Go
Norton AntiVirus
Google
Search Web
Submit
Done
EmpServlet - Microsoft Internet Explorer
http://eja:7001/WebApp1/empservlet
Done
Local intranet
POST
Server
Request
Servlet
Response

Employee Selection Form

ID 9393

Name Pee

Salary 4500

press Subm

7876 - ADAMS - 1100

7499 - ALLEN - 1600

7698 - BLAKE - 2850

7782 - CLARK - 2450

7902 - FORD - 3000

7900 - JAMES - 950

7566 - JONES - 2975

REALDOLMEN

The Request Object

- *javax.servlet.http.HttpServletRequest*
- Encapsulates all information from the client
- Access to request headers
- Access to an InputStream or Reader
- Access to parameters passed along with the URL

```
http://localhost:8080/MyWebApp/HelloWorldServlet?name=duke%20java
    &email=duke@sun.com
```

- Access to form data specified in a HTML document

Frequently Used Request Methods

```
javax.servlet.ServletRequest {
    Enumeration getParameterNames();
    String getParameter(String paramName);
    String getRemoteAddr();
}

javax.servlet.http.HttpServletRequest {
    String getRequestURI();
    Enumeration getHeaderNames();
    String getHeader(String headerName);
    HttpSession getSession();
    Cookie[] getCookies();
}
```

Exercise

- Write a HelloNameServlet
  - This servlet should accept accept a name which it prints out
  - Use the getParameter method on the HttpServletRequest object to obtain the name provided in the URL

The Response Object

- *javax.servlet.http.HttpServletResponse*
- Encapsulates all communication to client
- Access to response headers
- Access to an OutputStream or Writer
- Access to setting cookies
- Convenience methods for sending redirects, error pages, etc.

Frequently Used Response Methods

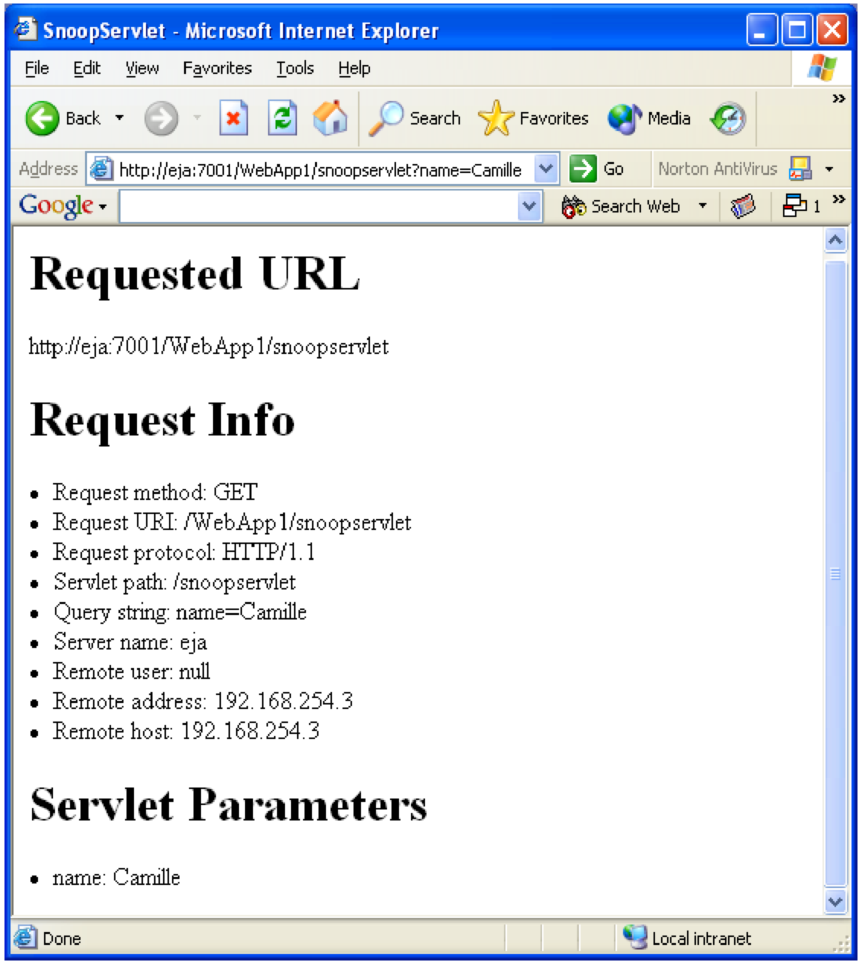
```
javax.servlet.ServletResponse {
    ServletOutputStream getOuputStream();
    PrintWriter getWriter();
    void setContentType(String type);
    void setContentLength(int length);
}

javax.servlet.http.HttpServletResponse {
    void addCookie(Cookie cookie);
    void setStatus(int statusCode);
    void sendError(int statusCode);
    void sendRedirect(String url);
}
```

Handling GET & POST requests

- GET and POST requests can be handled separately
  - It is recommended to factor out common code into one method
  - Let doPost call doGet!

```
public class OrderServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>Hello</TITLE></HEAD>");
        // ...
    }
    public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String order = request.getParameter("order");
        out.println("<HTML><HEAD><TITLE>Confirm</TITLE></HEAD>");
        out.println("<h3>Your order " + order + " is accepted");
        // ...
    }
}
```

SnoopServlet Example


SnoopServlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://eja:7001/WebApp1/snoopservlet?name=Camille
Go
Norton AntiVirus
Google
Search Web
Done
Local intranet

Requested URL

http://eja:7001/WebApp1/snoopservlet

Request Info

- Request method: GET
- Request URI: /WebApp1/snoopservlet
- Request protocol: HTTP/1.1
- Servlet path: /snoopservlet
- Query string: name=Camille
- Server name: eja
- Remote user: null
- Remote address: 192.168.254.3
- Remote host: 192.168.254.3

Servlet Parameters

- name: Camille

SnoopServlet Example

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

public class SnoopServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html";
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>SnoopServlet</title></head>");
        out.println("<body bgcolor=\"#ffffff\">");
        out.println("<h1>Requested URL</h1>");
        out.println(request.getRequestURL());
        // ...
```

SnoopServlet Example

```
// ...
out.println("<h1>Request Info</h1>");
out.println("<li>Request method: " +
    request.getMethod() + "</li>");
out.println("<li> Request URI: " +
    request.getRequestURI() + "</li>");
out.println("<li> Request protocol: " +
    request.getProtocol() + "</li>");
out.println("<li> Servlet path: " +
    request.getServletPath() + "</li>");
out.println("<li> Query string: " +
    request.getQueryString() + "</li>");
out.println("<li> Server name: " +
    request.getServerName() + "</li>");
out.println("<li> Remote user: " +
    request.getRemoteUser() + "</li>");
out.println("<li> Remote address: " +
    request.getRemoteAddr() + "</li>");
out.println("<li> Remote host: " +
    request.getRemoteHost() + "</li>");
// ...
```

SnoopServlet Example

```
        // ...
        out.println("<h1>Servlet Parameters</h1>");
        // Returns a non-generic Enumeration
        Enumeration e = request.getParameterNames();
        while (e.hasMoreElements()) {
            String name = (String) e.nextElement();
            String value = request.getParameter(name);
            out.println("<li>" + name + ": " + value + "</li>");
         }
         out.println("</body></html>");
        }
    }
```

HTTP Status Codes

- Example HTTP 1.1 Response

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE ...>
<HTML>
<!-- ... -->
</HTML>
```

- Changing the status code lets you perform a number of tasks not otherwise possible
  - Forward client to another page
  - Indicate a missing resource
  - Instruct browser to use cached copy
- Set status before sending document

HTTP Status Codes

- *response.setStatus(int statusCode)*
  - Use constant for the code, not an explicit int
    - Constants are in HTTPServletResponse
  - Names derived from standard message
    - E.g. SC_OK, SC_NOT_FOUND, etc.
- *response.sendError(int code, String message)*
  - Wraps message inside small HTML document
- *response.sendRedirect(String URL)*
  - Relative URLs permitted in 2.2 and later
  - Sets Location header also

HTTP Status Codes

- 200 (OK)
  - Everything is fine; document follows (default for servlets)
- 204 (No Content)
  - Browser should keep displaying previous document
- 301 (Moved Permanently) / 302 (Found)
  - Requested document temporarily moved elsewhere
  - Browsers go to new location automatically
  - Servlets should use sendRedirect(), not setStatus()
- 401 (Unauthorized)
- 404 (Not Found)
- 501 (Not Implemented)

Exercise

- Create a Servlet that outputs an HTML form
  - Make sure there are two text fields (firstName and lastName), a submit button and a reset button
  - First use GET as method attribute in the form tag, and leave the action attribute empty for now (action=””)
  - Look at the URL in your browser when you press the submit button
  - Change the action attribute in the form tag in POST
  - Look at the URL again in your browser if you press the submit button
  - Now process the request parameters
    - Read all parameters from the form, and transmit an HTML response with all the parameter names and their values

The Potential of Cookies

- Idea
  - Servlet sends a simple name and value to client
  - Client returns same name and value when it connect to same site (or same domain, depending on cookie settings)
- Typical Uses of Cookies
  - Identifying a user during an e-commerce session
    - Servlets have a higher-level API for this task
  - Avoiding username and password
  - Customizing a site
  - Focusing advertising

Some Problems with Cookies

- The problem is privacy, not security
  - Servers can remember your previous actions
  - If you give out personal information, servers can link that information to your previous actions
  - Servers can share cookie information through use of a cooperating third party such as *ads.doubleclick.net*
  - Poorly designed sites store sensitive information like credit card numbers directly in cookie
- Moral for servlet authors
  - If cookies are not critical to your task, avoid servlets that totally fail when cookies are disabled
  - Don’t put sensitive info in cookies
- Web Browser is expected to support 20 cookies per host of at most 4KB each

Sending Cookies to Browser

- Create a cookie (*javax.servlet.Cookie*)

```
Cookie cookie = new Cookie(name, value);
```

- Setting / changing cookie attributes
  - setComment()
  - setMaxAge()
    - setMaxAge(60*60*24*7)   // 7 days
    - setMaxAge(0)   // remove cookie
    - setMaxAge(-1)   // session cookie
  - setValue()
- Send cookie to user
  - response.addCookie(cookie);

Sending Cookies to Browser

```
public void doGet (HttpServletRequest req,
                   HttpServletResponse res)
                   throws ServletException, IOException {
    // ...
    String bookId = req.getParameter("bookId");
    Cookie cookie = new Cookie("Buy", bookId);
    cookie.setComment("User wants to buy this book " +
                   "from the bookstore.");
    cookie.setMaxAge(60*60*24*7);
    res.addCookie(cookie);
    // ...
}
```

Reading Cookies from Browser

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
                  throws ServletException, IOException {
    // ...
    Cookie[] cookies = req.getCookies();
    for(int i = 0; i < cookies.length; i++) {
        Cookie thisCookie = cookies[i];
        if (thisCookie.getName().equals("Buy")) {
            thisCookie.setMaxAge(0);
        }
     }
    // ...
 }
```

Methods in the Cookie API

- getDomain() / setDomain()
  - Specify domain to which cookie applies
- getMaxAge() / setMaxAge()
  - Cookie expiration time (in seconds)
- getName()
- getPath() / setPath()
  - Specify path to which cookie applies
- getSecure() / setSecure()
  - Only for SSL or all connections
- getValue() / setValue()
  - Associate value with cookie

Exercise

- Write a RepeatVisitorServlet
  - The servlet prints out the text “Welcome Aboard” when detecting a first visit by a client
  - For subsequent visits by a client, the servlet displays “Welcome Back”
  - Use cookies

Exercise

- Write a RegistrationFormServlet
  - The servlet prints a form that allows a client to enter his first name, last name and email address
  - The servlet is capable of detecting fields that were previously entered, and shows the data from those fields in the form
  - Use cookies

Why Session Tracking

- When clients at on-line store add items to their shopping card, how does the server know what's already in the cart ?
- When clients decide to proceed to checkout,
  how can server determine which previously created cart is theirs ?

REALDOLMEN

Implementing your own Session Tracking: Cookies

- Idea: associate cookie with data on server

```
String sessionID = makeUniqueString();
HashMap<String, Object> sessionInfo = new Hashtable<String, Object>();
HashMap<String, HashMap<String, Object>> globalMap =
    findTableStoringSessions();
globalMap.put(sessionID, sessionInfo);
Cookie sessionCookie = new Cookie("JSESSIONID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

- Still to be done:
  - Extracting cookie that stores session identifier
  - Setting appropriate expiration time for cookie
  - Associating the hash tables with each request
  - Generating the unique session identifiers

REALDOLMEN

Implementing your own Session Tracking: URL-Rewriting

- Idea:
  - Client appends some extra data on the end of each URL that identifies the session
  - Server associates that identifier with data it has stored about the session

```
http://localhost:8080/MyWeb/file.html?jsessionid=1234
```

- Advantage
  - Works even if cookies are disabled or unsupported
- Disadvantage
  - Lots of tedious processing
  - Must encode all URLs that refer to your own site
  - Links from other sites and bookmarks can fail

REALDOLMEN

Implementing your own Session Tracking: Hidden Form Fields

- Idea:

```
<INPUT TYPE="HIDDEN" NAME="session" VALUE="..."/>
```

- Advantage
  - Works even if cookies are disabled or unsupported
- Disadvantage
  - Lots of tedious processing
  - All pages must be the result of form submissions

The Session Tracking API

- Session objects live on the server
- Automatically associated with client via cookies or URL-rewriting
  - Obtain a HttpSession by calling getSession() on a HttpServletRequest object
  - Using getSession() always returns a HttpSession, a new one is created is no session exists
  - Using getSession(false) only returns an existing HttpSession
- Hashtable-like mechanism lets you store arbitrary objects inside session
  - setAttribute(name, value) to store information
  - getAttribute(name) to retrieve information

Session Example

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

public class OrderServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                HttpServletResponse response)
                throws ServletException, IOException {
        // Get the user's session
        HttpSession session = request.getSession();
        // add a value to the session
        String orderId = request.getParameter("orderId");
        session.setAttribute("order", orderId);
        // ...
    }
}
```

HttpSession Methods

- getAttribute() / setAttribute()
- removeAttribute()
  - Removes values associated with name
- getAttributeNames()
  - Returns names of all attributes in the session
- getId()
  - Returns the unique identifier
- isNew()
  - Determines if session is new to client (not to page)

HttpSession Methods

- getCreationTime()
  - Returns time at which session was first created
- getLastAccessedTime()
  - Returns time at which session was last sent from client
- getMaxInactiveInterval() / setMaxInactiveInterval()
  - Gets or sets the amount of time session should go without access before being invalidated
- invalidate()
  - Invalidates the session and unbinds all objects associated with it

REALDOLMEN

Exercise

- Adapt your RegistrationFormServlet
  - Keep the previously entered information in the HttpSession instead of cookies

Servlets are Not Just for HTML

- Servlets are usually used to generate dynamic HTML pages
- They can also be used to:
  - Generate images using AWT and Java 2D API
  - Generate custom data formats for use by applets or other apps
  - Send serialized objects to and read serialized objects from apps

Image Generation Example

```
public void doGet(HttpServletRequest req,
                HttpServletResopnse res)
                throws ServletException, IOException {
    // do logic to build image
    Image image = buildImage(req);
    res.setContentType("image/gif");
    ServletOutputStream out = res.getOutputStream();
    // use acme.com's Gif Encoder
    GifEncoder encoder = new GifEncoder(image, out);
    encoder.encode();
}
// code omitted

/*
 * acme: American Company Making Everything (Warner Brothers, Looney
 * Tunes), also providers of some open source utilities
 */
```

REALDOLMEN

HTTP Tunneling

- Servlets can be used to service clients that are sitting behind a firewall
- Allows for client / server communication using:
  - Text based messaging (XML, java.util.Properties)
  - Binary messaging using object serialization
  - Other or custom data formats
- Text based formats lend to easier cross platform inter-operation
- Serialization allows for complex data structures to be transmitted

Using Properties in Client & Server Communication

```
// Servlet side
java.util.Properties props = new java.util.Properties();
props.put("result", "true");
ServletOutputStream out = response.getOutputStream();
props.save(out, "Servlet Property Stream");

// Client side
URL url = new URL("http://myserver:8080/myweb/resultservlet");
InputStream in = url.openStream();
Properties props = new Properties();
props.load(in);
String result = props.getProperty("result");
```

Using Serialization in Client & Server Communication

```
// Servlet side
Result result = getResult(); // generate object
ServletOutputStream out = response.getOutputStream();
ObjectOutputStream obj = new ObjectOutputStream(out);
obj.writeObject(result);

// Client side
URL url = new URL("http://myserver:8080/myweb/resultservlet");
InputStream in = url.openStream();
ObjectInputStream objIn = new ObjectInputStream(in);
Result result = (Result) objIn.readObject();
```

What are the Servlet Listeners

- A set of interfaces for managing the ServletContext-related events and HTTP session events
- Defined in javax.servlet & javax.servlet.http packages
  - ServletContextListener
  - ServletContextAttributesListener
  - HttpSessionListener
  - HttpSessionAttributeListener
- More listeners are available in the packages...

ServletContextListener

- Manages the life cycle-related events of the servlet context
  - When a Web application is created or destroyed

```
public class SCListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent sce) {
    }

    public void contextDestroyed(ServletContextEvent sce) {
    }
}
```

ServletContextAttributeListener

- Used to handle events when attributes are added(), deleted() or replaced() in the ServletContext

```
public class SCAListener implements ServletContextAttributeListener {

    public void attributeAdded(ServletContextAttributeEvent scae) {
    }

    public void attributeRemoved(ServletContextAttributeEvent scae) {
    }

    public void attributeReplaced(ServletContextAttributeEvent scae) {
    }
}
```

REALDOLMEN

HttpSessionListener

- Manages the life cycle-related events of the HTTP session state
  - When HTTP session is created or invalidated

```
public class HttpSListener implements HttpSessionListener {

    public void sessionCreated(HttpSessionEvent se) {
    }

    public void sessionDestroyed(HttpSessionEvent se) {
    }
}
```

HttpSessionAttributeListener

- Used to implement a listener to handle events when attributes are added(), delete() or replaced() in the HTTP session object

```
public class HttpSAListener implements HttpSessionAttributeListener {

    public void attributeAdded(HttpSessionBindingEvent se) {
    }

    public void attributeRemoved(HttpSessionBindingEvent se) {
    }

    public void attributeReplaced(HttpSessionBindingEvent se) {
    }
}
```

Listeners in web.xml

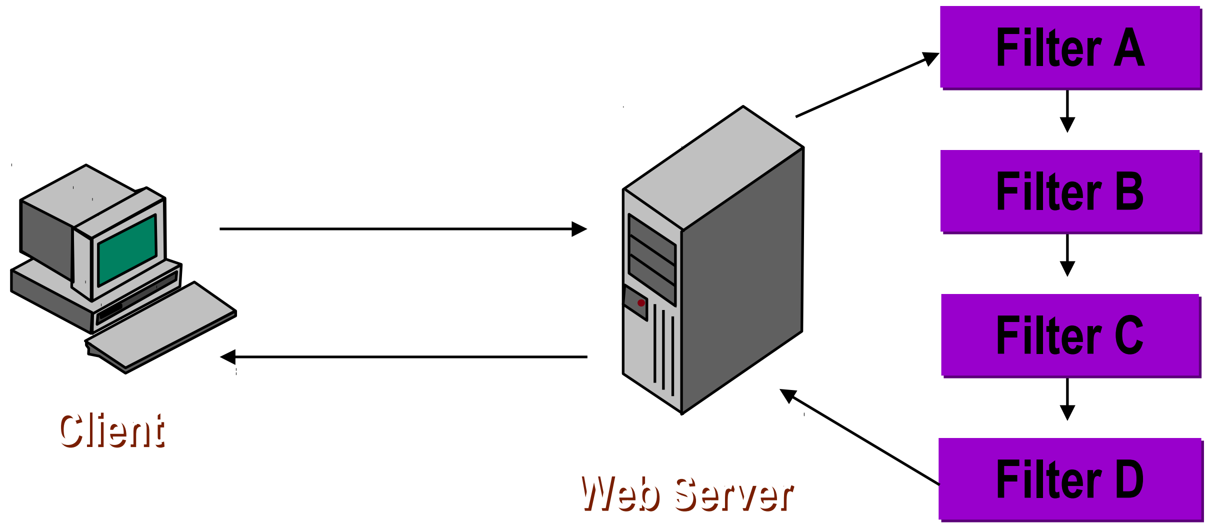
```
<web-app>
    <listener>
        <listener-class>
            com.realdolmen.servlets.HttpSAListener
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>
            com.realdolmen.servlets.HelloWorldServlet
        </servlet-class>
    </servlet>
    <!-- ... →
</web-app>
```

Exercise

- Write an HttpSessionListener
  - Keep track of the number of sessions that have been created / destroyed
  - Keep the counter on the application scope (ServletContext)
  - Write a Servlet that prints out the number of active sessions
  - Configure the listener in the web.xml

What is a Servlet Filter ?

- A Filter is a servlet-like container-managed object that can be declaratively inserted within the HTTP request-response process



Possibilities of Servlet Filters

- Validate HTTP request
  - Each filter has access to the HTTP request object and can validate the contents of the HTTP request
- Log HTTP request
  - Implement custom access logging for all web resources
- Authorize HTTP request
  - Implement custom authorization code, security checks
- Content Management
  - Prepare content for web container
- Provide custom HTTP environment
  - Modify the request and response objects

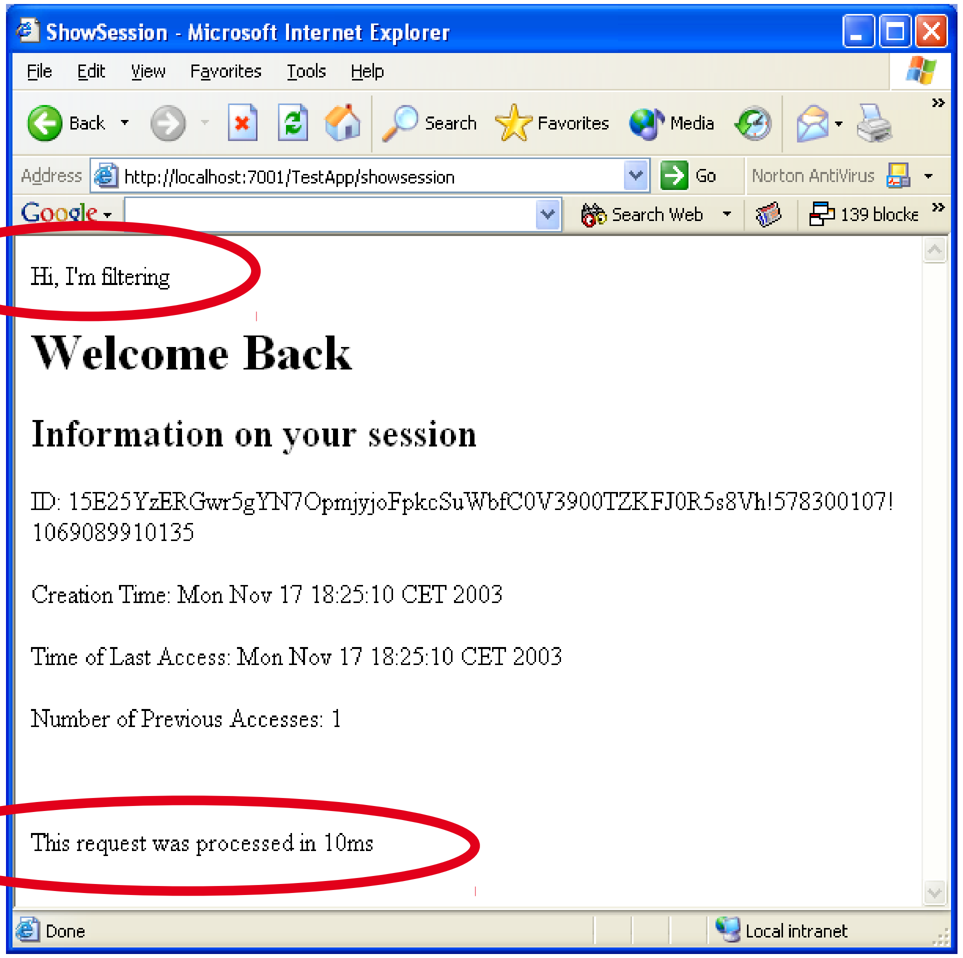
Servlet Filtering Example

```
public class HiFilter implements Filter {
    private FilterConfig filterConfig;
    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }
    public void doFilter(ServletRequest request, ServletResponse
                    response, FilterChain filterChain) {
        try {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.print("Hi, I'm filtering");
            filterChain.doFilter(request, response);
        }
        catch(ServletException sx) {
            filterConfig.getServletContext().log(sx.getMessage());
        }
        catch(IOException iox) {
            filterConfig.getServletContext().log(iox.getMessage());
        }
    }
    public void destroy() {}
}
```

Servlet Filtering Example

```
public class TimerFilter extends HttpServlet implements Filter {
    private FilterConfig filterConfig;
    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }
    public void doFilter(ServletRequest request, ServletResponse
                    response, FilterChain filterChain) {
        try {
            long begin = System.currentTimeMillis();
            filterChain.doFilter(request, response);
            long end = System.currentTimeMillis();
            response.getWriter().print("<br/><br/>This request was
                processed in " + (end - begin) + "ms");
        } catch(ServletException sx) {
            filterConfig.getServletContext().log(sx.getMessage());
        } catch(IOException iox) {
            filterConfig.getServletContext().log(iox.getMessage());
        }
    }
    public void destroy() {}
}
```

Servlet Filtering Example


ShowSession - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:7001/TestApp/showsession Go
Norton AntiVirus
Google Search Web 139 blocke
Done
Local intranet

Hi, I'm filtering

Welcome Back

Information on your session

ID: 15E25YzERGwr5gYN7OpmjyjoFpkcSuWbfC0V3900TZKFJ0R5s8Vh!578300107!1069089910135

Creation Time: Mon Nov 17 18:25:10 CET 2003

Time of Last Access: Mon Nov 17 18:25:10 CET 2003

Number of Previous Accesses: 1

This request was processed in 10ms

Access Log Servlet Filter Example

```
public class AccessLogFilter implements Filter {
    protected PrintWriter log;
    public void init(FilterConfig config) throws ServletException {
        try {
            File f = new File(config.getServletContext().
                getServletContextName() + "_access.log");
            if (!f.exists())
                f.createNewFile();
            log = new PrintWriter(new FileWriter(f.getName(), true));
        }
        catch (IOException ioe) {
            throw new ServletException(ioe);
        }
    }
    public void destroy() {
        // Closing log (add try-catch!)
        if (log != null)
            log.close();
    }
    // ...
```

Access Log Servlet Filter Example

```
// ...
    public void doFilter(ServletRequest req, ServletResponse res,
        FilterChain chain) throws IOException, ServletException {
        StringBuffer sb = new StringBuffer();
        sb.append("[");
        sb.append(new java.util.Date());
        sb.append("]Request by ");
        sb.append(req.getRemoteHost());
        sb.append(" for ");
        sb.append(((HttpServletRequest)req).getRequestURI());
        sb.append(".");
        log.println(sb.toString());
        log.flush();
        chain.doFilter(req, res);
    }
}
```

Configuring Filters in web.xml

- Filters should be put before any listener declarations, but after the context-param elements in web.xml

```
<filter>
  <filter-name>myFilter</filter-name>
  <display-name>My Filter</display-name>
  <description>This is my filter</description>
  <filter-class>examples.myFilterClass</filter-class>
</filter>
<filter-mapping>
  <filter-name>myFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Filter mappings can be applied to servlets and make it possible to define filter chains

```
<filter-mapping>
  <filter-name>myFilter</filter-name>
  <servlet-name>myServlet</servlet-name>
</filter-mapping>
```

Exercise

- Write a Filter
  - The Filter increments a Session parameter for every request to the server (per user)
  - The parameter “requestCount” can be used in a servlet to display the current number of requests for that user
  - Do not forget to include your filter in your web.xml

Java Servlets and JSP Programming

JavaServer Pages (JSP)

The Need for JSP

- Servlets excel in programming or data processing tasks
  - Manipulating request parameters, HTTP request headers and HTTP response data
  - Cookie and session-tracking
  - Talk to relational databases with JDBC (usually by delegating to underlying services)
- But Servlets are not so good with presentation issues
  - It is hard to write and maintain the HTML
  - You cannot use standard HTML tools
  - The HTML is inaccessible to non-Java developers

The JSP Framework

- Idea:
  - Use regular HTML for most of page
  - Mark servlet code with special tags
  - Entire JSP page gets translated into a servlet (once) and servlet is what actually gets invoked (for each request)
- Simple example:
  - JSP

    Thanks for ordering <b><%= request.getParameter(“title”) %></b>
  - URL

    http://localhost:8080/MyWeb/OrderConfirmation.jsp?title=JSP %20Programming
  - Result

    Thanks for ordering **JSP Programming**

Benefits of JSP

- JSP provides the following benefits over Servlets
  - It is easier to write and maintain the HTML
    - Static code is ordinary HTML
  - You can use standard Web-site development tools
    - E.g. MacroMedia DreamWeaver, Frontpage, NVU, ...
  - You can divide up your development team
    - Java programmer – dynamic code
    - Web developer – presentation layer
- JSP & Servlets are complementary technologies!

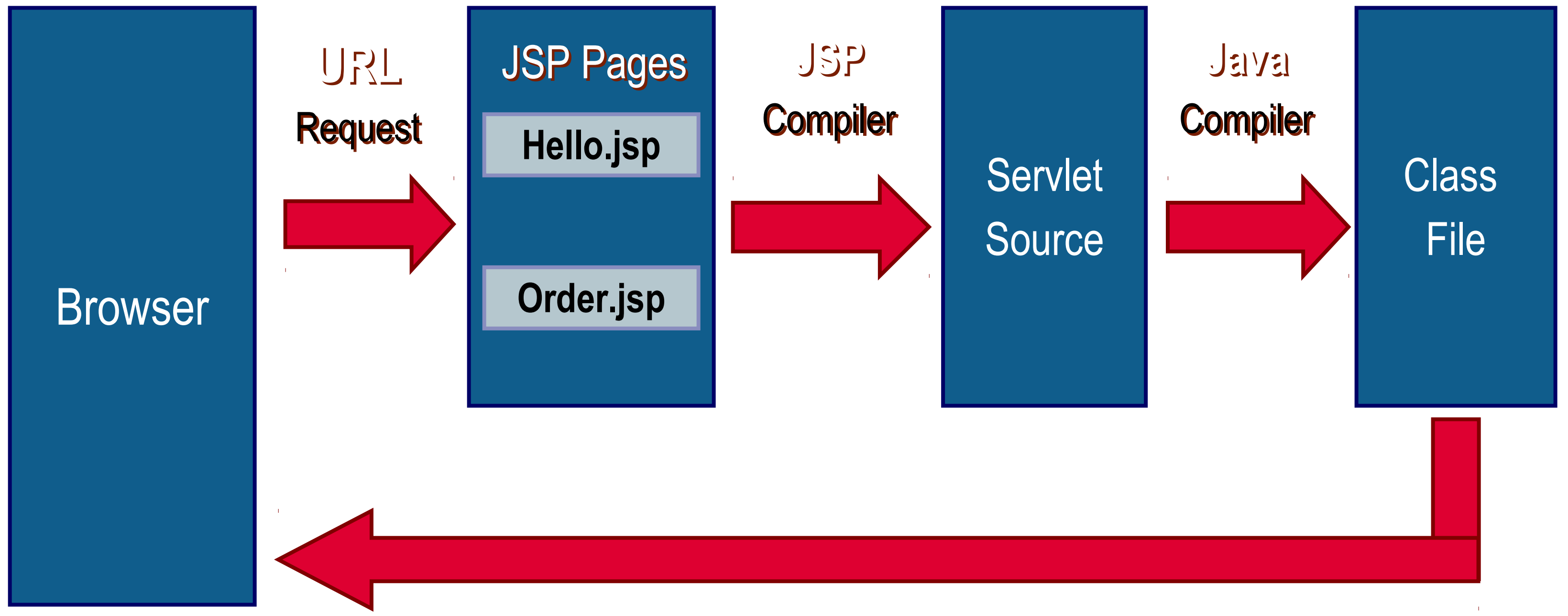
JSP Example

```
<%@ page import="java.util.Date" %>
<html>
    <head><title>CurrentDate</title></head>
    <body bgcolor="#ffffff">
        <h1>Current Date</h1>
        The current time is <%= new Date().toString() %><br>
        Your hostname is <%= request.getRemoteHost() %><br>
        Your session ID is <%= session.getId() %><br>
    </body>
</html>
```

Exercise

- Write a JSP page that displays “HelloWorld” and the current time
- Deploy the JSP page and test

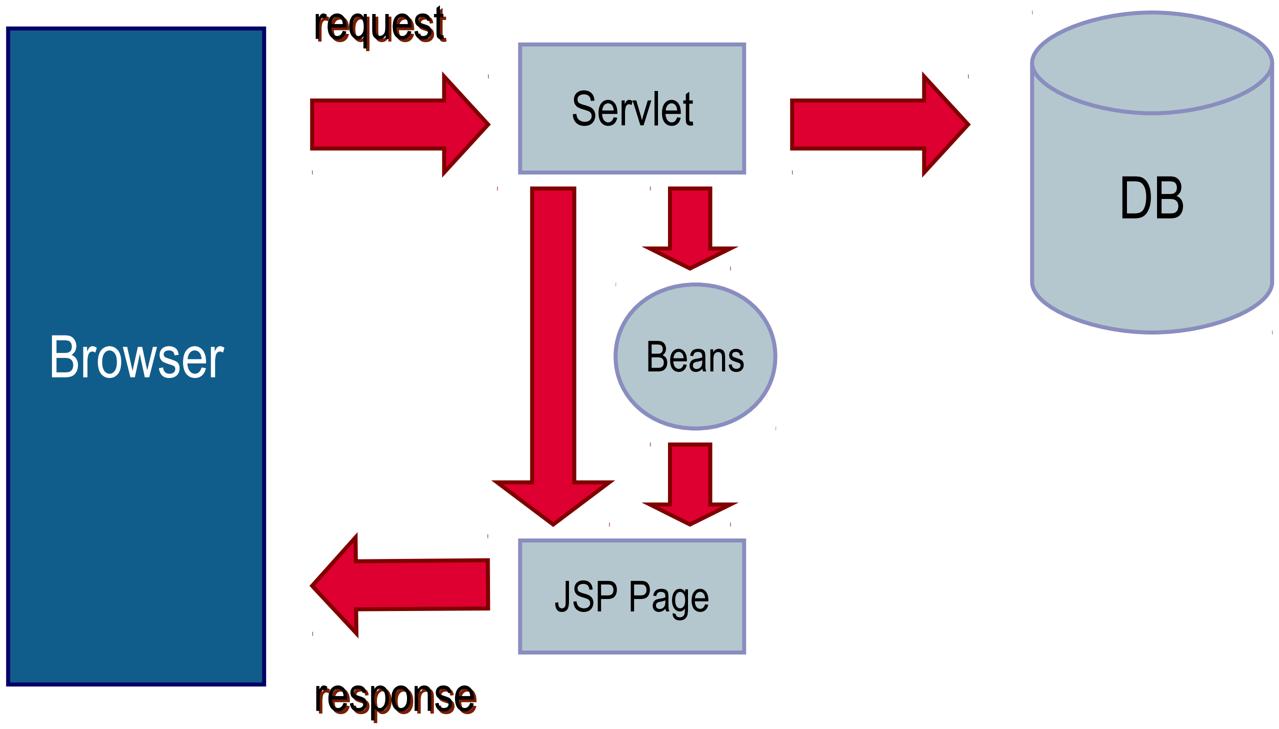
Creation of a JSP Class File



REALDOLMEN

JSP Access Model

- More about this model later...



JSP History

- JSP 1.0 (1999)
- JSP 1.1
  - Custom Tag extensions
- JSP 1.2
  - Assumes Java 2 Platform
  - Requires Servlet 2.3 API
- JSP 2.0
  - Part of J2EE 1.4
  - Requires Servlet 2.4 API
- JSP 2.1 (2006)
  - Part of J2EE 1.5
  - Requires Servlet 2.5 API

JSP Tags

- JSP Tags can be expressed in two forms
  - Short-hand
  - XML equivalent
- Each form has several types
  - Scriptlet - Java code fragments
  - Expression - produce a value to be inserted
  - Declaration - declare variable or methods
  - Directive - directives to the server
  - Action - more advanced feature

REALDOLMEN

JSP Tags - Scriptlet

- Embeds Java source code within your HTML
- The Java code is executed and its output is inserted in sequence with the rest of the HTML in the page
- Begins with <% followed by space and ends with %>
- Does not need to be complete !
- May not include method definitions or inner classes !

```
<%  java_code %>

// XML equivalent:

<jsp:scriptlet>
    java_code
</jsp:scriptlet>
```

REALDOLMEN

Scriptlet Example

```
<font color="navy">
<%
 for (int i=0;i<10;i++) {
   out.println("<b>Hello World. This is a scriptlet test"
        + i + "</b><br>");
   System.out.println("This goes to the System.out stream " + i);
 }
%>
</font>
```

Predefined Variables

- JSP provides eight predefined variables

request	has information from the client
response	has ways to interact with client
out	an output stream for HTML
pageContext	wrapper for servlet-related functions
session	has session information
application	data about the servlet engine
config	has instance initialization parameters
page	current object, same as this

Predefined Variables Example

```
<HTML>
<BODY>
<H1>Server Information</H1>
<%
    out.println("<BR>Server:" + request.getServerName());
    out.println("<BR>Port:" + request.getServerPort());
    out.println("<BR>Port:" + request.getRemoteHost());
    out.println("<BR>Date:" + new Date().toString());
    out.println("<BR>Session ID:" + session.getId());
%>
</BODY>
</HTML>
```

JSP Tags - Expression

- Defines a Java expression that is evaluated at page request time, converted to a String, and sent inline to the output stream of the JSP response
- May not be terminated with a semicolon!
- Must return a value which can be cast to a String !

```
<%= expression%>

// XML equivalent:

<jsp:expression>
    expression
</jsp:expression>
```

Scriptlet & Expression Example

```
<font color="navy">
<% for (int i=0; i<10; i++) { %>
     <b>Hello World. This is a scriptlet test <%= i%> </b><br/>
<%
     System.out.println("This goes to the System.out stream " + i);
   }
%>
</font>
<i>Date:  <%=  new java.util.Date().toString()%> </i><br/>
```

Using Scriptlets to Make Conditional JSP Files

- Point
  - Scriptlets are inserted into servlet exactly as written
  - Need not be complete Java expressions
  - Complete expressions are usually clearer and easier to maintain, however
- Example

```
<% if (Math.random() < 0.5) { %>
Have a <b>nice</b> day !
<% } else { %>
Have a <b>lousy</b> day !
<% } %>
```

REALDOLMEN

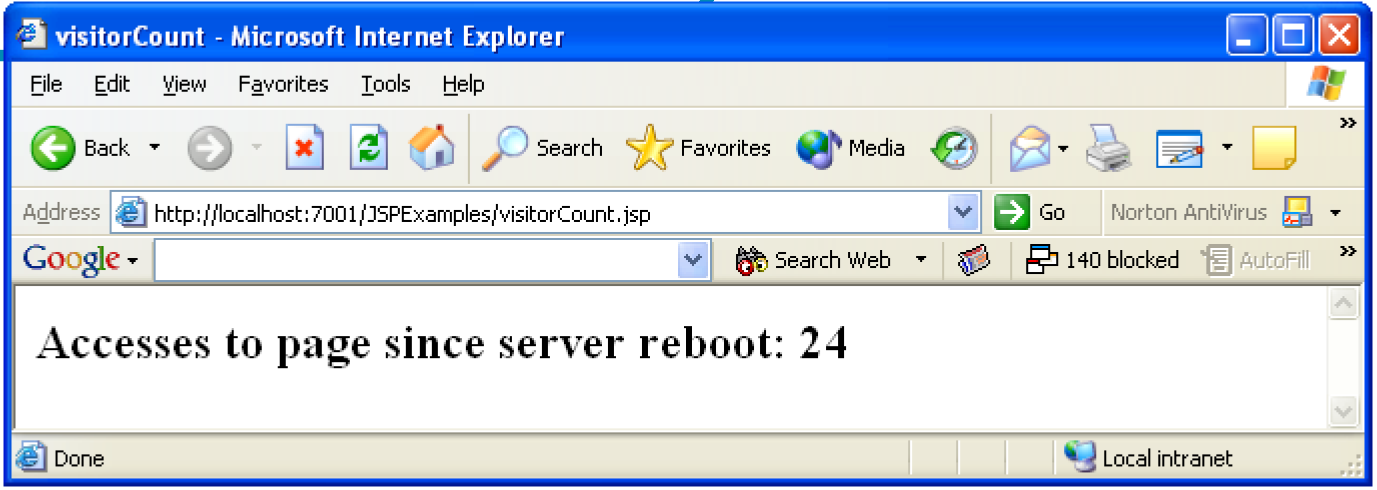
JSP Tags - Declaration

- Declares a variable or method that may be referenced by other declarations, scriptlets, or expressions in the page
- Begins with <%! followed by space and ends with %>
- Needs to be complete !
- Not recommended !

```
<%! declaration %>

// XML equivalent:

<jsp:declaration>
    declaration;
</jsp:declaration>
```

Declaration Example

```
<html>
<head><title>visitorCount</title></head>
<body bgcolor="#ffffff">
<h1>
<%-- not the best access counter --%>
<%! private int accessCount = 0; %>
</h1>
<h2>Accesses to page since server reboot:
<%= ++accessCount %></h2>
</body>
</html>
```


visitorCount - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:7001/JSPExamples/visitorCount.jsp Go
Norton AntiVirus
Google Search Web 140 blocked AutoFill
Done Local intranet

Accesses to page since server reboot: 24

REALDOLMEN

Exercise

- Write a JSP page that prints out all the parameters of the request
- The page displays a counter value that is incremented each time the page is accessed

Exercise

- Write a JSP that displays a table
  - The parameter “format” will define the display format
    - format = “excel”: show as excel sheet
    - format = “html”: show as html page
  - Add the following for sending an excel sheet:
    - response.setContentType("application/vnd.ms-excel");
    - response.setHeader("Content-disposition", "attachment;filename=exceldata.xls");

JSP Page directives

- Gives a directive to the Application Server
- You can insert a directive into the JSP page anywhere
- The dir_type determines the type of directive being given, which can accept a list of directives given as name="quotedValue" pairs separated by white space

```
<%@ page dir_type dir_attr %>

// XML equivalent:

<jsp:directive.dir_type dir_attr />
```

REALDOLMEN

JSP Page directives

- The following page directives exist in JSP
  - <%@ page autoFlush= %>
  - <%@ page buffer= %>
  - <%@ page contentType= %>
  - <%@ page errorPage= %>
  - <%@ page extends= %>
  - <%@ page import= %>
  - <%@ page info= %>
  - <%@ page isThreadSafe= %>
  - <%@ page language= %>
  - <%@ page session= %>

JSP Page directives

- The “autoFlush” page directive

```
<%@ page autoFlush="true|false" %>
```

  - Specifies whether the output buffer to the client is flushed automatically when it is full
    - true = the buffer is automatically flushed
    - false = an exception is raised (buffer overflow)
    - The default is true
  - It is possible to set autoFlush to false when buffer=”none”

JSP Page directives

- The “buffer” page directive

```
<%@ page buffer="none" %>
<%@ page buffer="sizekb" %>
```

  - Specifies the buffering model for the initial “out” JspWriter
    - “none” = no buffering, all output is written directly through to the ServletResponse PrintWriter
    - “buffer size”= output is buffered with a buffer size not less than that specified
  - Example:

```
<%@ page buffer="16kb" %>
```

REALDOLMEN

JSP Page directives

- The “contentType” page directive

```
<%@ page contentType="MIME-type" %>
<%@ page contentType="MIME-type;charset=Character-Set" %>
```

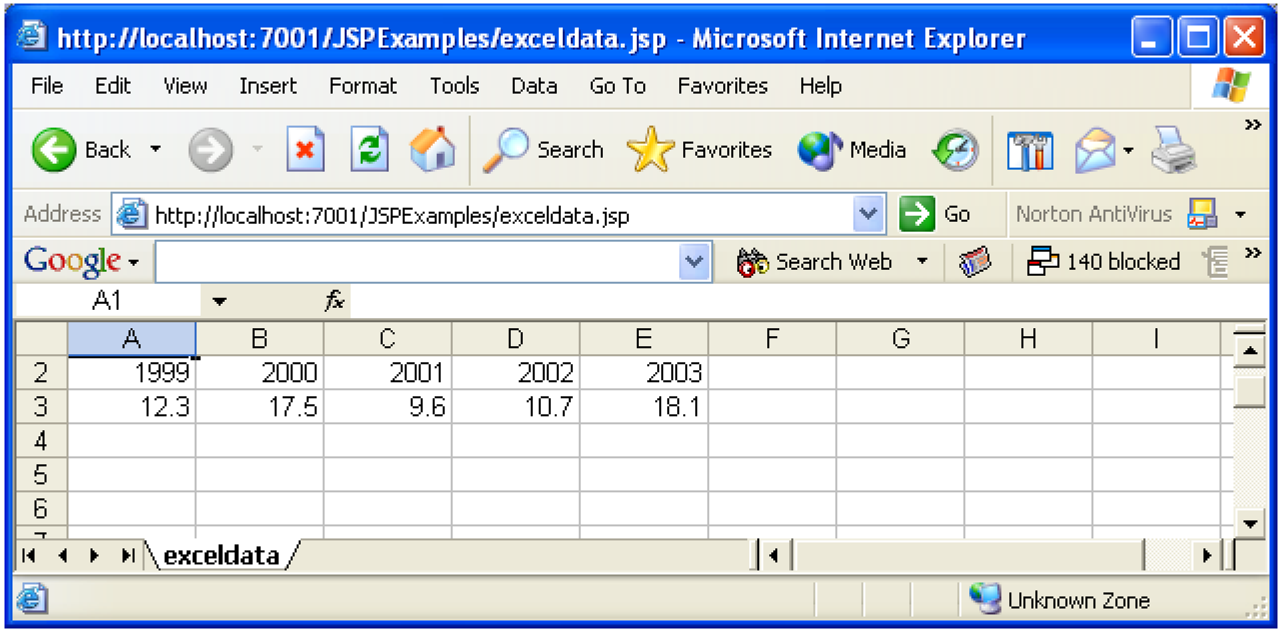
  - Specifies the MIME type of the page generated by the servlet
  - Default value for the MIME-type : “text/html”
  - Example:

```
<%@ page contentType="text/html" %>
```

JSP Page directives

- This is an example of the “contentType” page directive

```
<%@ page contentType="application/vnd.ms-excel" %>
1999     2000     2001     2002     2003
12.3     17.5     9.6      10.7     18.1
```


http://localhost:7001/JSPExamples/exceldata.jsp - Microsoft Internet Explorer
File Edit View Insert Format Tools Data Go To Favorites Help
Back Search Favorites Media
Address http://localhost:7001/JSPExamples/exceldata.jsp Go
Norton AntiVirus
Google Search Web 140 blocked
A1
exceldata
Unknown Zone

	A	B	C	D	E	F	G	H	I
2	1999	2000	2001	2002	2003				
3	12.3	17.5	9.6	10.7	18.1				
4									
5									
6									

JSP Page directives

- The “errorPage” page directive

```
<%@ page errorPage="Relative URL" %>
```

  - Defines a URL to a resource to which any Java exception no caught by the page are forwarded to for error processing
  - The error page mechanism is very useful and avoids the need for developers to write code to catch unrecoverable exceptions in their JSP pages

JSP Page directives

- The “extends” page directive

```
<%@ page extends="package.class" %>
```

  - Name the superclass of the class to which this JSP page is transformed
  - This attribute should normally be avoided and only used with extreme caution, because JSP engines usually provide specialized superclasses
  - Not recommended!
  - Example:

```
<%@ page extends="com.realdolmen.jsp.MySuperJSP" %>
```

REALDOLMEN

JSP Page directives

- The “import” page directive

```
<%@ page import="package.class" %>
<%@ page import="package.class1,...,package.classN" %>
```

  - Import one or more packages
  - Packages imported by default:
    - java.lang.*
    - javax.servlet.*
    - javax.servlet.jsp.*
    - javax.servlet.http.*
  - Examples:

```
<%@ page import="java.util.Date" %>
<%@ page import="java.rmi.*,java.util.*" %>
```

JSP Page directives

- The “info” page directive

```
<%@ page info="Info-text" %>
```

  - Defines an arbitrary string that is incorporated into the translated page
  - Can subsequently be obtained from the page’s implementation of the Servlet.getServletInfo() method
  - Not very useful...
  - Example:

```
<%@ page info="My latest JSP Example V2.0" %>
```

REALDOLMEN

JSP Page directives

- The “isThreadSafe” page directive

```
<%@ page isThreadSafe="true|false" %>
```

  - Indicates the level of thread safety implemented in the page
    - false = the JSP container shall dispatch multiple outstanding client requests, one at a time, in the order they were received
    - true = the JSP container may choose to dispatch multiple outstanding client requests simultaneously
    - Default is true
  - Like implementing the *javax.servlet.SingleThreadModel* interface

REALDOLMEN

JSP Page directives

- Consider this example of threading is JSP

```
<%-- not the best id generator --%>
<%! private int idNum = 0; %>
<%  String userID = "userID" + idNum;
    out.println("Your ID is " + userID + ".");
    idNum = idNum + 1;
%>
```

What is wrong with this code ?

```
<%! private int idNum = 0; %>
<%
synchronized (this) {
    String userID = "userID" + idNum;
    out.println("Your ID is " + userID + ".");
    idNum = idNum + 1;
}
%>
--> Totally safe, better performance in high-traffic environments
--> There are better ways to assign userIDs!
```

isTreadSafe=”true”

JSP Page directives

- The “language” page directive

```
<%@ page language="Java" %>
```

  - Defines the scripting language to be used in the scriptlets, expression scriptlets, and declarations
  - With JSP, the only defined and required scripting language is “java”

JSP Page directives

- The “session” page directive

```
<%@ page session="true|false" %>
```

  - Indicates that the page requires participation in an (http) session
    - true = the implicit script language variable named “session” of type javax.servlet.http.HttpSession references the current/new session for the page
    - false = the page does not participate in a session; the “session” implicit variable is unavailable
    - Default is true, meaning that each jsp page creates a session by default

Including Files at Request Time

```
<jsp:include page="relative URL" flush="true" />
```

- Purpose
  - To reuse JSP, Servlet, HTML or plain text content
  - JSP content cannot affect main page: only output of included JSP page is used
  - To permit updates to the included content without changing the main JSP page(s)

Including Files at Request Time

```
...
<body>
<p>
Here is a summary of our four most recent news stories:
<ol>
    <li><jsp:include page="news/item1.html" flush="true" />
    <li><jsp:include page="news/item2.html" flush="true" />
    <li><jsp:include page="news/item3.html" flush="true" />
    <li><jsp:include page="news/item4.html" flush="true" />
<ol>
</body>
</html>
```

Including Files at Page Translation Time

```
<%@ include file="relative URL" %>
```

- Purpose
  - To reuse JSP content in multiple pages, where JSP content affects the main page
- Example

```
<%@ include file="copyright.html" %>
```

REALDOLMEN

Differences Between jsp:include and @include

- jsp:include includes the output of the designated page
  - @include includes the actual code
- jsp:include occurs at request time
  - @include occurs at page translation time
- With jsp:include, the main page and the included page become two separate servlets
  - With @include, they become parts of a single servlet
- jsp:include automatically handles changes to the included file
  - @include might not (big maintenance problem!)

Exercise

- Extend one of your existing JSP pages
- Include
  - copyright.snippet at translation time
  - currentdate.jsp at request time

Including Applets for the Java Plugin

```
<jsp:plugin type="applet" code="MyApplet.class" width="475"
    height="350">
</jsp:plugin>
```

- Other attributes:
  - codebase, align, archive, jreversion, iepluginurl, nspluginurl

```
<jsp:plugin type="applet" code="Demo.class" codebase="./" >
    <jsp:params>
        <jsp:param name="p1" value="v1"/>
    </jsp:params>
    <jsp:fallback>
        <p> unable to start plugin </p>
    </jsp:fallback>
</jsp:plugin>
```

<jsp:forward

```
<jsp:forward page="Forward URL" />
```

- Allows the runtime dispatch of the current request to a static resource, a JSP pages or a Java Servlet class in the same context as the current page
- Stops executing the current page
- Example:

```
<%! String whereTo = "/templates/" + someValue; %>
<jsp:forward page="<%= whereTo %>" />
```

REALDOLMEN

Comments

```
<%-- comment --%>
```

- Comments the JSP source
- Comments written within these tags are not included in the HTML output

Background: What are JavaBeans ?

- Java classes that follow certain conventions
  - Must have a zero-argument (empty) constructor
    - By explicitly defining such a constructor or by omitting all constructors
  - Should have no public instance variables (fields)
  - Persistent values should be accessed through methods called *getXxx* and *setXxx*
    - If a class has method getTitle that returns a String, class is said to have a String property named title
    - Boolean properties use *isXxx* instead of *getXxx*
  - Class should be public
  - Class needs some kind of persistence support
    - Has to implement java.io.Serializable or java.io.Externalizable
  - More info *http://java.sun.com/beans/doc*

Why Accessors and No Public Fields

- To create a bean, a class cannot have public fields
- So, you should replace

```
public double speed;
```

- with

```
private double speed;
public double getSpeed() {
    return (speed);
}
public void setSpeed(double newSpeed)
    speed = newSpeed;
}
```

- You should do this in all your Java classes ... Why ?

Why Accessors and No Public Fields

1) You can put constraints on values

```
public void setSpeed(double newSpeed) {
    if (newSpeed < 0) {
        sendErrorMessage(/* ... */);
        newSpeed = Math.abs(newSpeed);
    }
    speed = newSpeed;
}
```

- If users of your class access the fields directly, then they would be responsible for checking constraints themselves
- This can create problems if they forget about it!

Why Accessors and No Public Fields

2) You can change your internal representation without changing interface

```
// Now using metric units (kph, not mph)
public void setSpeed(double newSpeed) {
     speedInKPH = convert(newSpeed);
}
public void setSpeedInKPH(double newSpeed) {
     speedInKPH = newSpeed;
}
```

- If users of your class access the fields directly, they would be responsible for knowing about the conversion and they would be responsible to perform it before changing the field

Why Accessors and No Public Fields

3) You can perform arbitrary side effects

```
public void setSpeed(double newSpeed) {
     speed = newSpeed;
     updateSpeedometerDisplay();
}
```

- If users of your class access the fields directly, then they would be responsible for executing the side effects, which can easily be forgotten!

JSP Tags - Actions

- Basic tags:
  - <jsp:useBean ... >
  - <jsp:setProperty ... >
  - <jsp:getProperty ... >
- JSP Actions encompass the more advanced features of JSP and only use XML syntax

Basic Bean Use in JSP

```
<jsp:useBean id="id" scope="session" class="classname"/>
```

- Purpose
  - Allow instantiation of Java classes without explicit Java programming
- Simple interpretation

```
<jsp:useBean id="book" class="com.realdolmen.Book" />
```

  - can be thought of as equivalent to the scriptlet

```
<% com.realdolmen.Book book = new com.realdolmen.Book(); %>
```

  - But useBean has two additional advantages
    - It is easier to derive object values from request parameters
    - It is easier to share objects among pages or servlets

REALDOLMEN

Accessing Bean Properties

```
<jsp:getProperty name="name" property="property"/>
```

- Purpose
  - Allow access to bean properties (i.e., calls to getXxx methods) without explicit Java programming
  - Uses bean introspection to access the property
- Simple interpretation

```
<jsp:getProperty name="book" property="title" />
```

  - is equivalent to the following JSP expression

```
<%= book.getTitle() %>
```

Setting Bean Properties – Simple Case

```
<jsp:setProperty name="name" property="property"
    value="value"/>
```

- Purpose
  - Allow setting of bean properties (i.e., calls to setXxx methods) without explicit Java programming
  - Uses bean introspection to access the property
- Simple interpretation

```
<jsp:setProperty name="book" property="title" value="JSP Overview" />
```

  - is equivalent to the following scriptlet

```
<% book.setTitle("JSP Overview"); %>
```

StringBean Example

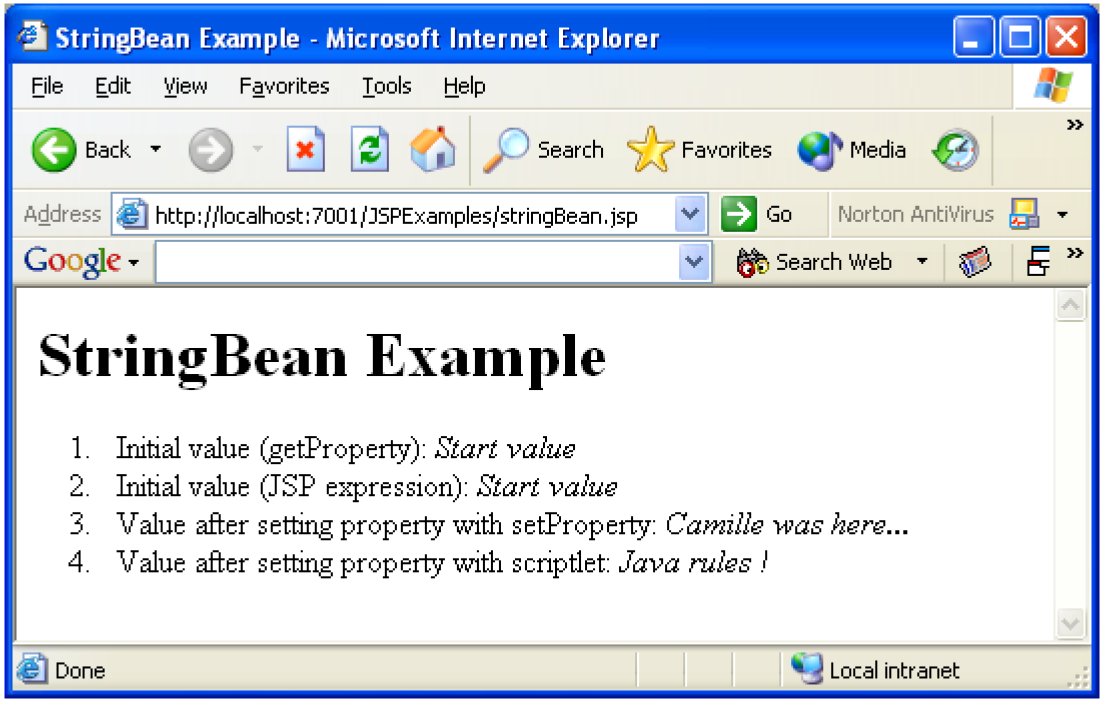
```
package jspexamples;

public class StringBean implements Serializable{
    private String sample = "Start value";

    //Access sample property
    public String getSample() {
        return sample;
    }

    //Access sample property
    public void setSample(String newValue) {
        if (newValue!=null) {
            sample = newValue;
        }
    }
}
```

REALDOLMEN

StringBean Example

```
<html>
    <head><title>StringBean Example</title></head>
    <jsp:useBean id="stringBean" scope="session"
        class="jspexamples.StringBean" />
    <body>
        <h1>StringBean Example</h1>
        <ol><li> Initial value (getProperty): <i>
        <jsp:getProperty name="stringBean" property="sample" /></i>
        <li> Initial value (JSP expression): <i>
        <%= stringBean.getSample() %></i>
        <li><jsp:setProperty name="stringBean" property="sample"
        value="Camille was here..." />
        Value after setting property with setProperty: <i>
        <jsp:getProperty name="stringBean" property="sample" /></i>
        <li><% stringBean.setSample("Java rules !");%>
        Value after setting property with scriptlet: <i>
        <%= stringBean.getSample() %></i></ol>
    </body>
</html>
```

StringBean Example


StringBean Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:7001/JSPExamples/stringBean.jsp
Go
Norton AntiVirus
Google
Search Web
Done
Local intranet

StringBean Example

1. Initial value (getProperty): *Start value*
2. Initial value (JSP expression): *Start value*
3. Value after setting property with setProperty: *Camille was here...*
4. Value after setting property with scriptlet: *Java rules !*

REALDOLMEN

Setting Bean Properties
Explicit Conversion & Assignment

```
<%
int numItemsOrdered = 1;
try {
    numItemsOrdered =
        Integer.parseInt(request.getParameter("numItems"));
} catch (NumberFormatException nfe) {
}
%>

<%-- getNumItems expects an int --%>
<jsp:setProperty name="entry" property="numItems"
    value="<%= numItemsOrdered %>" />
```

Setting Bean Properties
Associating Properties with param

- Use the param attribute of jsp:setProperty to indicate that
  - Value should come from specified request parameter
  - Simple automatic type conversion should be performed for properties that expect values of type boolean, Boolean, byte, Byte, char, Character, double, Double, int, Integer, float, Float, long or Long

REALDOLMEN

Setting Bean Properties
Associating Properties with param

```
<jsp:useBean id="entry" class="com.realdolmen.SaleEntry" />

<jsp:setProperty
    name="entry"
    property="itemID"
    param="itemID" />

<jsp:setProperty
    name="entry"
    property="numItems"
    param="numItems" />

<jsp:setProperty
    name="entry"
    property="discountCode"
    param="discountCode" />
```

Setting Bean Properties
Associating All Properties with param

- Use "*" for the value of the property attribute of jsp:setProperty to indicate that
  - Value should come from request parameter whose name matches property name
  - Simple automatic type conversion should be performed

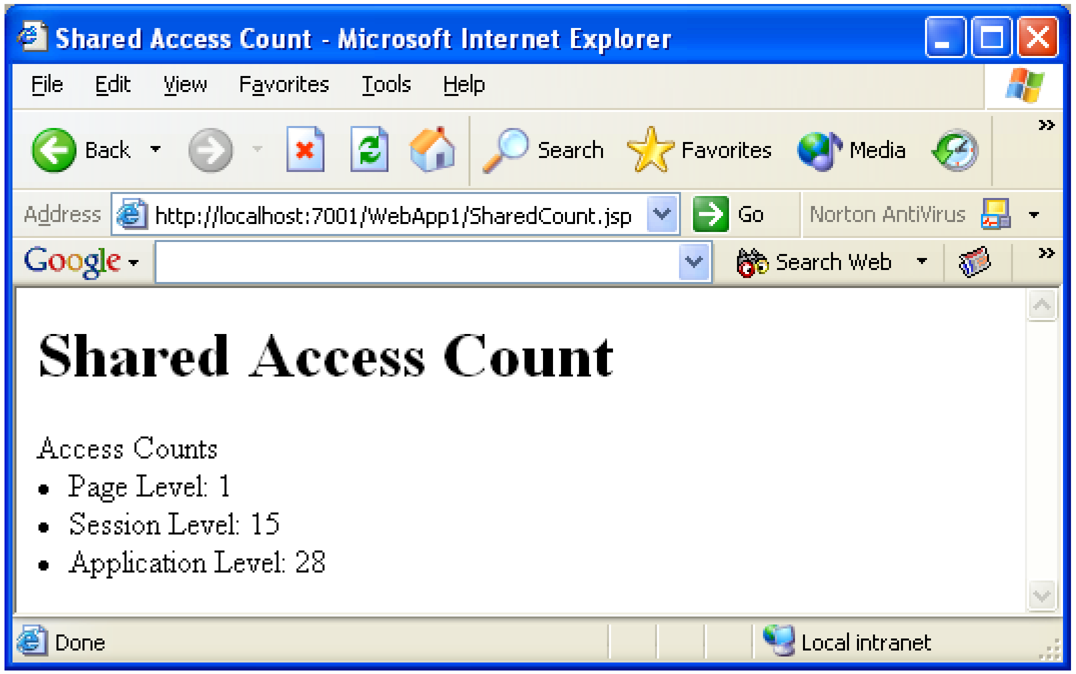
```
<jsp:useBean id="entry" class="com.realdolmen.SaleEntry" />
<jsp:setProperty name="entry" property="*" />
```

JSP Tags - Actions scope

- page
  - Default scope
  - Only available from the current invocation of this JSP page
- request
  - The object is stored in the current ServletRequest and is available to other included JSP pages that are passed the same request object
- session
  - Store the JavaBean object in the HTTP Session so that it may be tracked across several HTTP pages
  - The reference to the JavaBean is stored in the page's HttpSession object
- application
  - The JavaBean object is stored in the ServletContext
  - Is available to any other servlet or JSP page running in the server

Scope Example

```
public class AccessCountBean {
    private int accessCount = 1;
    public int getAccessCount() { return accessCount; }
    public void incrementAccessCount () { accessCount++; }
}
```


Shared Access Count - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:7001/WebApp1/SharedCount.jsp Go Norton AntiVirus
Google Search Web
Done Local intranet

Shared Access Count

Access Counts

- Page Level: 1
- Session Level: 15
- Application Level: 28

REALDOLMEN

Scope Example

```
<jsp:useBean id="pageCounter" class="beans.AccessCountBean"
    scope="page" />
<jsp:useBean id="sessionCounter" class="beans.AccessCountBean"
    scope="session" />
<jsp:useBean id="applicationCounter" class="beans.AccessCountBean"
    scope="application" />

<p>Access Counts

<li>Page Level:
<jsp:getProperty name="pageCounter" property="accessCount" />
<% pageCounter.incrementAccessCount(); %>

<li>Session Level:
<jsp:getProperty name="sessionCounter" property="accessCount" />
<% sessionCounter.incrementAccessCount(); %>

<li>Application Level:
<jsp:getProperty name="applicationCounter" property="accessCount" />
<% applicationCounter.incrementAccessCount(); %>
```

Exercise

- Write an Employee JavaBean with the following instance variables
  - firstName, lastName, salary, age, city
- Write a JSP form that allows data entry for the Employee Bean
- Print out the data from the Employee in the same page when the data has been filled in
- This exercise can later be done using the Model-View-Controller pattern

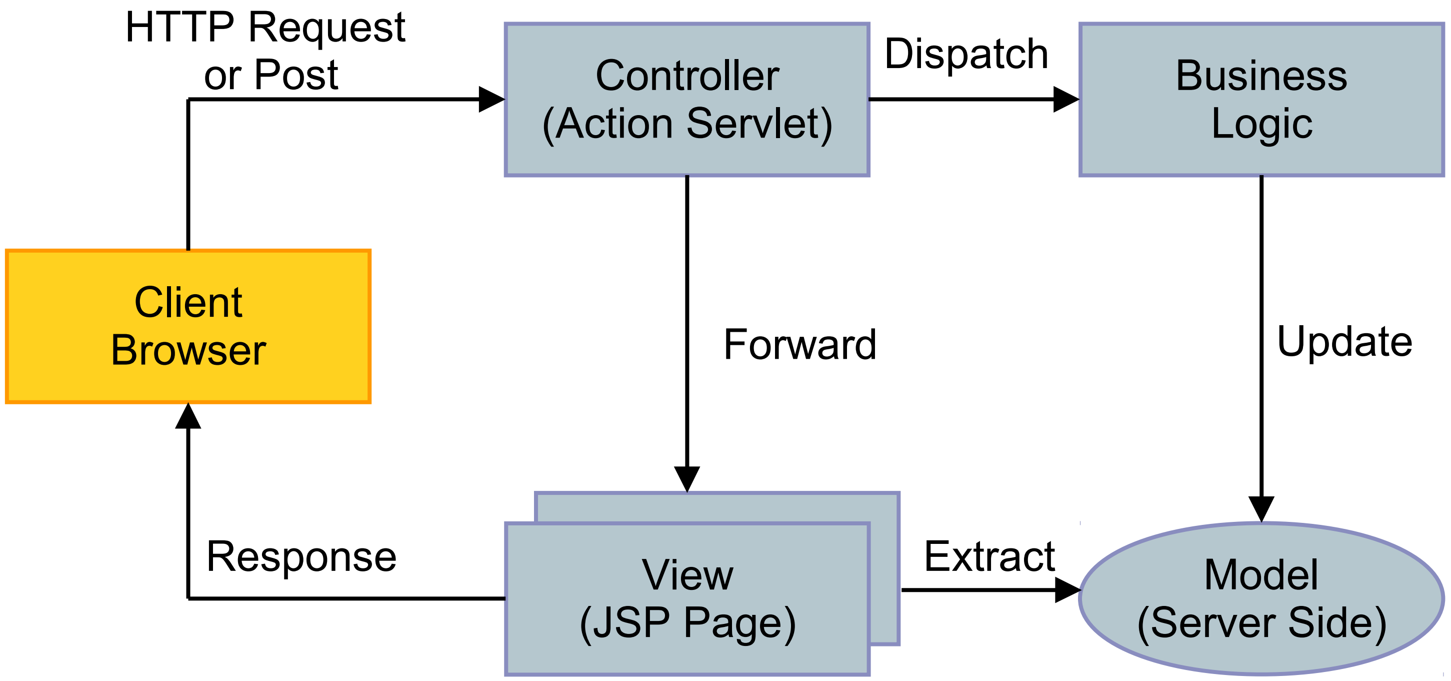
Why Combine Servlets & JSP?

- Typical picture: use JSP to make it easier to develop and maintain the HTML content
  - For simple dynamic code, call Servlet code from scripting element
  - For slightly more complex applications, use custom classes for scripting elements
  - For moderately complex applications, use beans and custom tags
- But that’s not enough
  - For complex processing, starting with JSP is awkward
  - Despite the ease of separating the real code into separate classes, beans and custom tags, the assumption behind JSP is that a single page give a single basic look

Approach

- Joint servlet / JSP process:
  - Original request is answered by a servlet
  - Servlet processes request data, does database lookup, business logic, etc.
  - Results are placed in beans
  - Request is forwarded to a JSP page to format result
  - Different JSP pages can be used to handle different types of presentation
- Often called the “MVC” (Model-View-Controller) or “Model 2” approach to JSP
- Formalized in Apache Struts Framework
  - *http://jakarta.apache.org/struts/*

Implementing MVC



REALDOLMEN

Implementing MVC

1. Define beans to represent the data
2. Use a servlet to handle requests
3. Populate the beans
4. Store the bean in the request, session or servlet context
5. Forward the request to a JSP page
   - Using RequestDispatcher.forward()
6. Extract the data from the beans
   - Using jsp:useBean & jsp:getProperty

Storing the Results

- Storing data that JSP page will use only in this request

```
Result value = new Result();
request.setAttribute("key", value);
// in JSP:
<jsp:useBean id="key" class="com.rd.Result" scope="request"/>
```

- Storing data that the JSP page will use in this request and in later requests from the same client

```
Result value = new Result();
HttpSession session = request.getSession();
session.setAttribute("key", value);
// in JSP:
<jsp:useBean id="key" class="com.rd.Result" scope="session" />
```

Storing the Results

- Storing data that the JSP page will use in this request and in later requests from any client

```
Result value = new Result();
getServletContext().setAttribute("key", value);
// in JSP:
<jsp:useBean id="key" class="com.rd.Result" scope="application"/>
```

Forwarding Requests to JSP Pages

- Forward request by calling forward() from *RequestDispatcher*

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
    String operation = request.getParameter("operation");
    if (operation == null)
        operation = "unknown";
    String address;
    if  (operation.equals("order")) {
        address = "/WEB-INF/Order.jsp";
    } else if  (operation.equals("cancel")) {
        address = "/WEB-INF/Cancel.jsp";
    } else {
        address = "/WEB-INF/UnknownOperation.jsp";
    }
    RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

Exercise

- Adapt your previous exercise to use the Model-View-Controller pattern
  - The user fills in the form from a JSP
  - A servlet reads the request parameters and saves them in an Employee Bean on the session
  - The servlet forwards to a result JSP when all the form parameters were filled in
    - If some parameters were not filled in, add an error message
    - Put this error message on the request scope
    - Forward to the same page
  - Forward to a final JSP page that shows the results
- Adapt the exercise further by keeping all filled in Employees on the application scope
  - Add JSP pages and actions in your Controller to show the list of Employees in a table

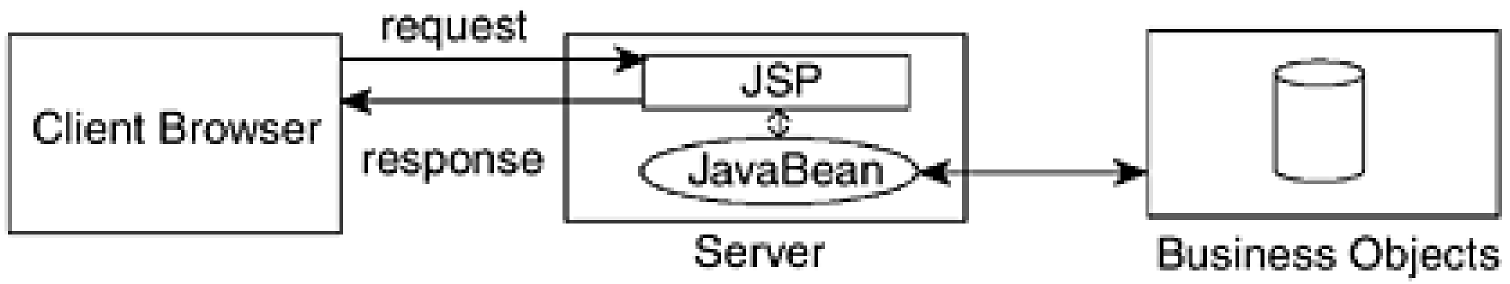
REALDOLMEN

Java Servlets and JSP Programming

Architectural Overview

Model 1 Architecture

- Application is page-centric
- Client browser navigates through a series of JSP pages in which any JSP page can employ a JavaBean that performs business operations
- Each JSP page processes its own input
- Applications normally have a series of JSP pages that user must traverse in sequence



REALDOLMEN

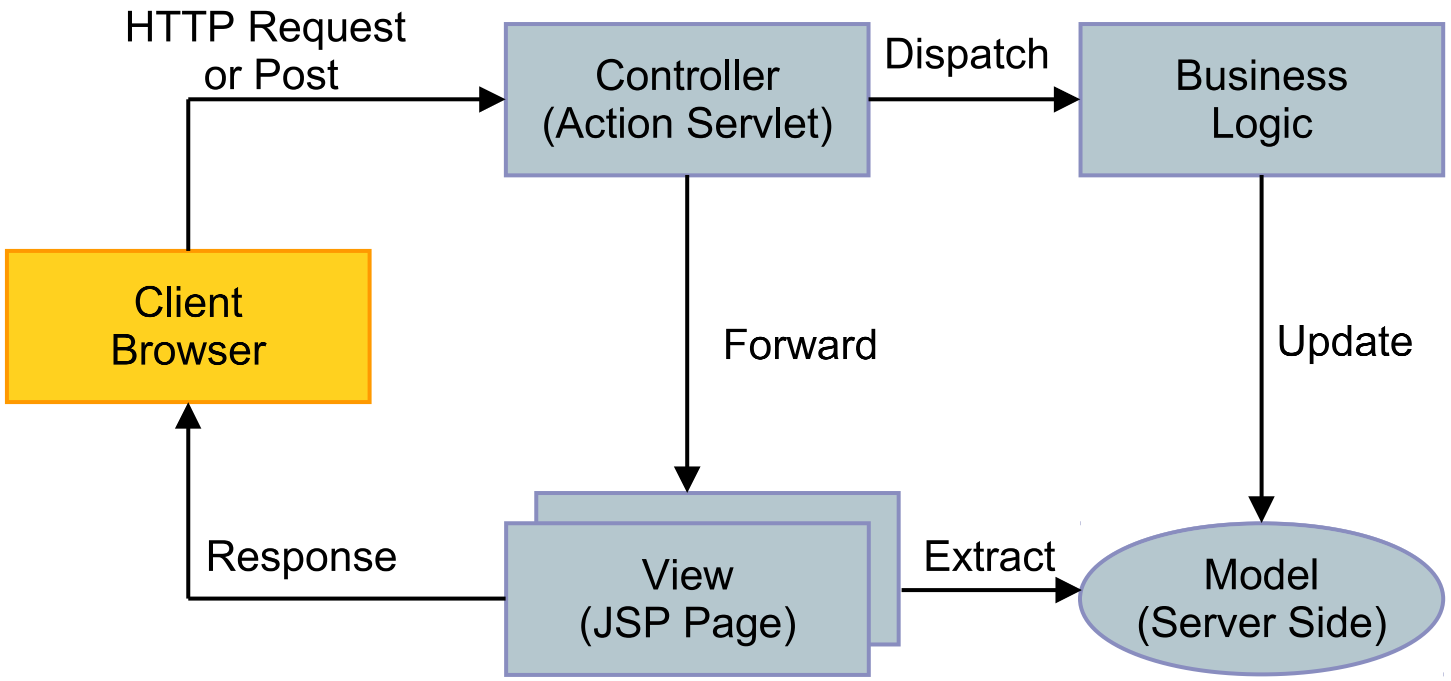
Model 1 Architecture

- Advantages
  - Ease of development
  - Suitable for small projects
- Disadvantages
  - Hard to achieve division of tasks between page designer and web developer
  - Hard to maintain
  - Not flexible

Model 2 Architecture

- Is a Model-View-Controller (MVC) architecture that separates content generation and content presentation
- A servlet acts as controller between client browser and JSP pages
- Controller servlet dispatches HTTP requests to corresponding presentation JSP pages, based on request URL, input parameters, application state
- Presentation pages are isolated
- Recommended for medium-sized to large applications
- Formalized (standardized) in Struts

Model 2 Architecture



Model 2 Architecture

- Advantages
  - Applications are more flexible, easier to maintain and to extend, because views do not reference each other directly
  - Controller servlet provides a single point of control for security and logging
  - Controller servlet encapsulates incoming data into a form usable by the back-end MVC model
- Disadvantages
  - Adds some initial complexity to application
  - Be careful for the Fat Controller anti-pattern (add Command Pattern, Business Delegates, etc)

REALDOLMEN

Java Servlets and JSP Programming

Expression Language (EL)

Expression Language (EL)

- Syntax:
  - ${expression}
- Capabilities
  - Concise access to stored objects
  - Shorthand notation for bean properties
  - Simple access to collection elements
  - Easy access to request parameters, cookies and other request data
  - Small but useful set of simple operators
  - Conditional output
  - Automatic type conversion
  - Empty values instead of error messages

EL Example

```
<ul>
    <li>Name: ${expression1}
    <li>Address: ${expression2}
</ul>

<jsp:include page="${expression3}" />
```

REALDOLMEN

Accessing Scoped Variables

- Use scoped variable in expression language
  - Syntax: ${name}
- Search (in that order)
  - PageContext
  - HttpServletRequest
  - HttpSession
  - ServletContext
- If attribute is found, its toString() method is called
  - Else empty string (not null or error message!)

REALDOLMEN

Example ScopedVars

```
public class ScopedVars extends HttpServlet {
    public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
        request.setAttribute("attribute1", "First Value");
        HttpSession session = request.getSession();
        session.setAttribute("attribute2", "Second Value");
        ServletContext app = getServletContext();
        app.setAttribute("attribute3", "Third Value");
        request.setAttribute("repeated", "Request");
        session.setAttribute("repeated", "Session");
        app.setAttribute("repeated", "Application");
        RequestDispatcher dispatcher =
            request.getRequestDispatcher("/el/scoped-vars.jsp");
        dispatcher.forward(request, response);
    }
}
```

Example ScopedVars

```
<HTML>
<HEAD><TITLE>Accessing Scoped Variables</TITLE>
</HEAD>
<BODY>
<UL>
    <LI><B>attribute1: </B> ${attribute1}
    <LI><B>attribute2: </B> ${attribute2}
    <LI><B>attribute3: </B> ${attribute3}
    <LI><B>repeated attribute: </B> ${repeated}
</UL>
</BODY></HTML>

<!-- results
 attribute1: First Value
 attribute2: Second Value
 attribute3: Third Value
 repeated attribute: Request
-->
```

REALDOLMEN

Accessing Bean Properties

- JSP Expression Language uses a simple but powerful “dot notation” for accessing bean properties
- Based on reflection
- Examples:

```
${customer.firstName}
${customer.address.zipcode}
${employee.company.companyName}
```

Accessing Bean Properties

- Example:

```
${customer.firstName}
```

replaces

```
<%@ page import="testservlets.NameBean" %>
<%
    NameBean person = (NameBean) pageContext.findAttribute("customer");
%>
<%= person.getFirstName() %>
```

or

```
<jsp:useBean id="customer" class="testservlets.NameBean"
    scope="request" />
<jsp:getProperty name="customer" property="firstName"/>
```

Array Notation

JSP 2.0

```
${name.property} == ${name["property"]}
```

- With array notation, the value inside the brackets can be a variable (dot notation must be literal value) or an expression
- With array notation, the property names can be illegal
  - Useful when accessing collections or request headers
- Examples
  - ArrayList: ${customerName[0]}
  - HashMap: ${stateCapitals.maryland} == ${stateCapitals[“maryland”]}

Referencing Implicit Objects

- pageContext

```
${pageContext.session.id}
```

- param and paramValues

```
${param.custID}
```

- header and headerValues

```
${header.Accept} or ${header["Accept"]}
${header["User-Agent"]}
```

Referencing Implicit Objects

- cookie

```
${cookie.userCookie.value} or ${cookie["userCookie"].value}
```

- initParam

```
${initParam.defaultColor}
```

- pageScope, requestScope, sessionScope and applicationScope

```
${requestScope.name}
```

REALDOLMEN

Using EL Operators

- Arithmetic Operators
  - +, -, *, /, div, %, mod
  - Example syntax: ${(1 + 2) * 3} -> 9
- Relational Operators
  - ==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge
- Logical Operators
  - &&, ||, !, not
- Empty Operator
  - empty
- Conditional Expressions
  - ${ test ? expression1 : expression2 }

REALDOLMEN

Java Servlets and JSP Programming

JavaServer Pages Standard Tag Library (JSTL)

JSTL Overview

- JSTL is an extension library for JSP
- It contains a set of tags for common behaviour
  - Conditional output
  - Iteration
  - Internationalisation
  - ...

JSTL Tag libraries

- JSTL features four tag libraries
  - Core
    - Usual prefix: c
    - http://java.sun.com/jsp/jstl/core
  - Formatting
    - Usual prefix: fmt
    - http://java.sun.com/jsp/jstl/fmt
  - Xml
    - Usual prefix: xml
    - http://java.sun.com/jsp/jstl/xml
  - Sql
    - Usual prefix: sql
    - http://java.sun.com/jsp/jstl/sql
- Including these libraries can be done as follows

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

REALDOLMEN

JSTL Tag libraries

- Of these tag libraries only two are used frequently
  - Core
  - Formatting
- The other two are used less
  - Sql
    - Goes against the separation of layers principle
    - Accessing persistence layer functionality directly inside a view unit is considered “bad practice”
  - Xml
    - Used in specific xml parsing situations
- We will focus on the first two

Core library

- The JSTL core library offers some very useful tags
  - The “if” tag

```
<c:if test=”${account.amount < 0}”>
    We have a negative!
</c:if>
```

    - Useful for simple conditional markup generation
    - Does **not** have an “else” clause
  - The “choose” tag
    - Similar to a switch statement

```
<c:choose>
    <c:when test=”${mode == 'slow'}”>Tortoise</c:when>
    <c:when test=”${mode == 'normal'}”>Human</c:when>
    <c:when test=”${mode == 'fast'}”>Rabbit</c:when>
    <c:otherwise>Something else</c:otherwise
</c:choose>
```

Core library

- The forEach tag
  - Allows iteration of lists, arrays, ...

```
<table>
    <c:forEach items="${items}" var="item">
        <tr>
            <td>${item}</td>
        </tr>
    </c:forEach>
</table>
```

- The set tag
  - Calculates a temporary value

```
<c:set var="total" value="${price * vat}"/>
Total price: ${total}
```

REALDOLMEN

Core library

- The out tag
  - Outputs some text in a XML safe way

```
<c:out escapeXml="true" value="${description}"/>
```

- The url tag
  - Generates an URL relative to the application context root

```
<c:url var=”myurl” value=”myresource.html”/>
<a href=”${myurl}”>Relative link</a>
```

Formatting library

- The formatting library offers features for internationalization and formatting
  - The setLocale tag
    - Allows the locale to be set for later use of i18n features
  - The setBundle tag
    - Offers access to Java i18n properties files for translations
  - The message tag
    - Outputs a message based on the specified bundle and locale

```
<fmt:setBundle basename="com.realdolmen.Messages"/>
<fmt:setLocale value="nl_BE"/>
<fmt:message key="welcome.message"/>
```

Resolves the message
with name “welcome.message”
in “com.realdolmen.Messages_nl_BE.properties”

Formatting library

- The formatDate tag
  - Provides a wrapper over Java's DateFormat classes
  - It allows you to display a date using an arbitrary format

```
<jsp:useBean id="myDate" class="java.util.Date"/>
<fmt:formatDate value="${myDate}" pattern="yyyy-MM-dd"/>
```

  - For the pattern options, check http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html
- The formatNumber tag
  - Provides a wrapper over Java's NumberFormat classes
  - It allows you to display a number using an arbitrary format

```
<fmt:formatNumber value="${1507.83}" pattern="#,##0.00"/>
```

  - For the pattern options, check http://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

JSTL Examples

- An example showing the use of the “fmt” namespace to provide internationalisation

```
<%@ page language="java" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<fmt:setBundle basename="com.realdolmen.jstl.JstlExample"/>
<html>
    <head>
        <title><fmt:message key="jstl.example.title"/></title>
    </head>
    <body>
        <h1><fmt:message key="jstl.example.title"/></h1>
        <!-- ... -->
```

- The accompanying properties file
  - “com/realdolmen/jstl/JstlExample_en_UK.properties”

```
jstl.example.title=JSTL example page
jstl.example.body=Employee name: {0}.
jstl.example.body.empty.name=Employee name: No name given.
```

REALDOLMEN

JSTL Examples

- An example showing the “c” namespace for conditional markup

```
<!-- ... -->
<div>
    <c:choose>
        <c:when test="${empty employee}">
            <p>Employee not found</p>
        </c:when>
        <c:otherwise>
            Employee found: ${employee.name}
        </c:otherwise>
    </c:choose>
</div>
<!-- ... -->
```

Exercise

- Adapt your previous exercise by using JSTL and EL
  - Replace all Java code and <jsp:useBean/> in your JSP pages with EL and JSTL

Java Servlets and JSP Programming

Custom tags

Custom Tags

- Until now, we have seen the following tags in JSP
  - The “jsp:*” tags
    - <jsp:include/> <jsp:forward/>, <jsp:useBean/>, <jsp:scriptlet/>, ...
  - The JSTL tags
    - <c:forEach/>, <c:if/>, ...
- It is possible to create your own custom tags as well
  - Benefits of using tags
    - Encapsulation of Java code
    - Separation of code and presentation
    - Reusability of coding
    - Better maintenance
    - Role-based development

REALDOLMEN

Custom Tag Libraries

- In order to create a custom tag, you need the following
  - A Java class that implements the “Tag” interface
    - Or one of it's subtypes “TagSupport” or “SimpleTagSupport”
  - A tag library descriptor
    - This is an XML file that describes metadata about your tag
- It is then possible to use your custom tag just like any JSTL tag
  - First add it to your JSP page with a taglib directive

```
<%@ taglib uri="http://www.realdolmen.com/rd" prefix="rd"%>
```

  - Then start using it

```
<rd:action message="Hello World!"/>
```

REALDOLMEN

JSP Custom Tag Example

- The Java implementation class

```
public class MessageTag extends SimpleTagSupport {
    private String message = "";

    // Also add getter and setter for message!

    public void doTag() throws JspException {
        PageContext pageContext = (PageContext) getJspContext();
        JspWriter out = pageContext.getOut();

        try {
            out.println("<p>A message: " + message + "</p>");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JSP Custom Tag Example

- The TLD xml configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" 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-jsptaglibrary_2_1.xsd ">
    <description>RealDolmen example JSP tags</description>
    <tlib-version>1.0</tlib-version>
    <short-name>rd</short-name>
    <uri>http://www.realdolmen.com/rd</uri>
    <tag>
        <description>Tag to display a message</description>
        <name>action</name>
        <tag-class>com.realdolmen.tags.MessageTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>message</name>
            <required>true</required>
        </attribute>
    </tag>
</taglib>
```

JSP Custom Tag Example

- Using the custom tag

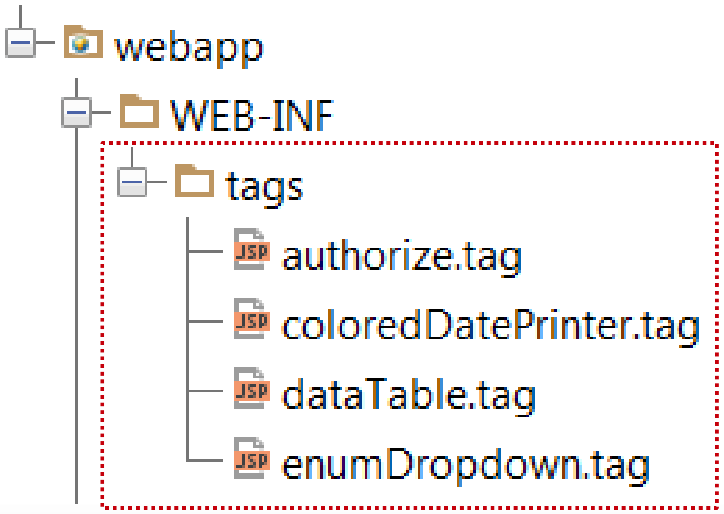
```
<%@ page language="java" %>
<%@ taglib uri="http://www.realdolmen.com/rd" prefix="rd" %>

<!DOCTYPE html>

<html>
  <head>
    <title>Tag test</title>
  </head>
  <body>
    <h1>Custom Tag test</h1>
    <p><rd:action message="Hello World!"/></p>
  </body>
</html>
```

JSP Tag fragments

- Custom JSP tags written in Java are very powerful
  - They allow access to the full capabilities of Java
  - They are extensible
  - They are reusable
  - They still provide a separation between UI logic and backend logic
- They are however, also relatively complex to create
  - You need a Java class
    - Which often requires redeploying of your application
  - You need a TLD
    - Which requires lots of XML configuration

JSP Tag fragments

- The JSP specification also allows you to create custom tags, while staying in JSP
  - These are called “Tag fragments”
- You can create a library of Tag fragments as follows
  - Create a folder in your web application
    - This folder is usually created under “WEB-INF”, so that it's contents are not publicly visible
  - Inside of this folder, you can add any number of “.tag” files



```
webapp
  WEB-INF
    tags
      authorize.tag
      coloredDatePrinter.tag
      dataTable.tag
      enumDropdown.tag
```

Tag fragment files

- A Tag fragment is a special kind of JSP page
  - Instead of containing an entire page, they only contain a “fragment” of a JSP page
  - Unlike a “jsp:include”, these fragments can be parametrized
    - This allows them to be reused in varying situations

```
<%@attribute name="color" required="true" type="java.lang.String"%>
<%@attribute name="pattern" required="true"%>
<%@attribute name="value" required="true"%>

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

<span style="color: ${color};">
    <fmt:parseDate var="date" pattern="${pattern}" value="${value}"/>
    <fmt:formatDate value="${date}" pattern="${pattern}"/>
</span>
```

Using Tag fragment libraries

- A Tag fragment library can be included to a JSP page with a taglib directive
  - The name of the tag will be the same as the name of the tag file, with the .tag suffix omitted

```
<%@ taglib prefix="rd" tagdir="/WEB-INF/tags" %>

<html>
    <head><title>My cool page</title></head>
    <body>
        <h1>My cool page</h1>
        <rd:coloredDatePrinter color="red" pattern="yyyy-MM-dd"
                value="1983-07-15"/>
    </body>
</html>
```

Assumes /WEB-INF/tags/coloredDatePrinter.tag

Custom Java Tags versus JSP Tag fragments

- Both approaches have advantages and disadvantages

- Custom Java Tags
  - (+) More powerful
    - They allow full access to all Java capabilities
  - (-) More difficult to create
    - Requires more clumsy configuration
- JSP Tag fragments
  - (+) Easier to create
    - You just need to add a .tag file
  - (-) Less powerful
    - You can only access the features of JSPs

When to use what?

- Custom Java Tags are usually better if you need a lot of dynamic behaviour
  - Java code is better for this
- JSP Tag fragments are usually better if you need a lot of static markup
  - Templating systems like JSP are better for this
- Rule of thumb
  - Try to see if you can get it done with a Tag fragment first
  - If not, consider the full custom Java Tag approach

REALDOLMEN

REALDOLMEN

Java Servlets and JSP Programming

JSP Documents: JSPX

JSP Hybrid

- JSP is a hybrid technology
  - It looks a lot like XML but it's not!
  - JSP uses syntax that is not valid according to XML

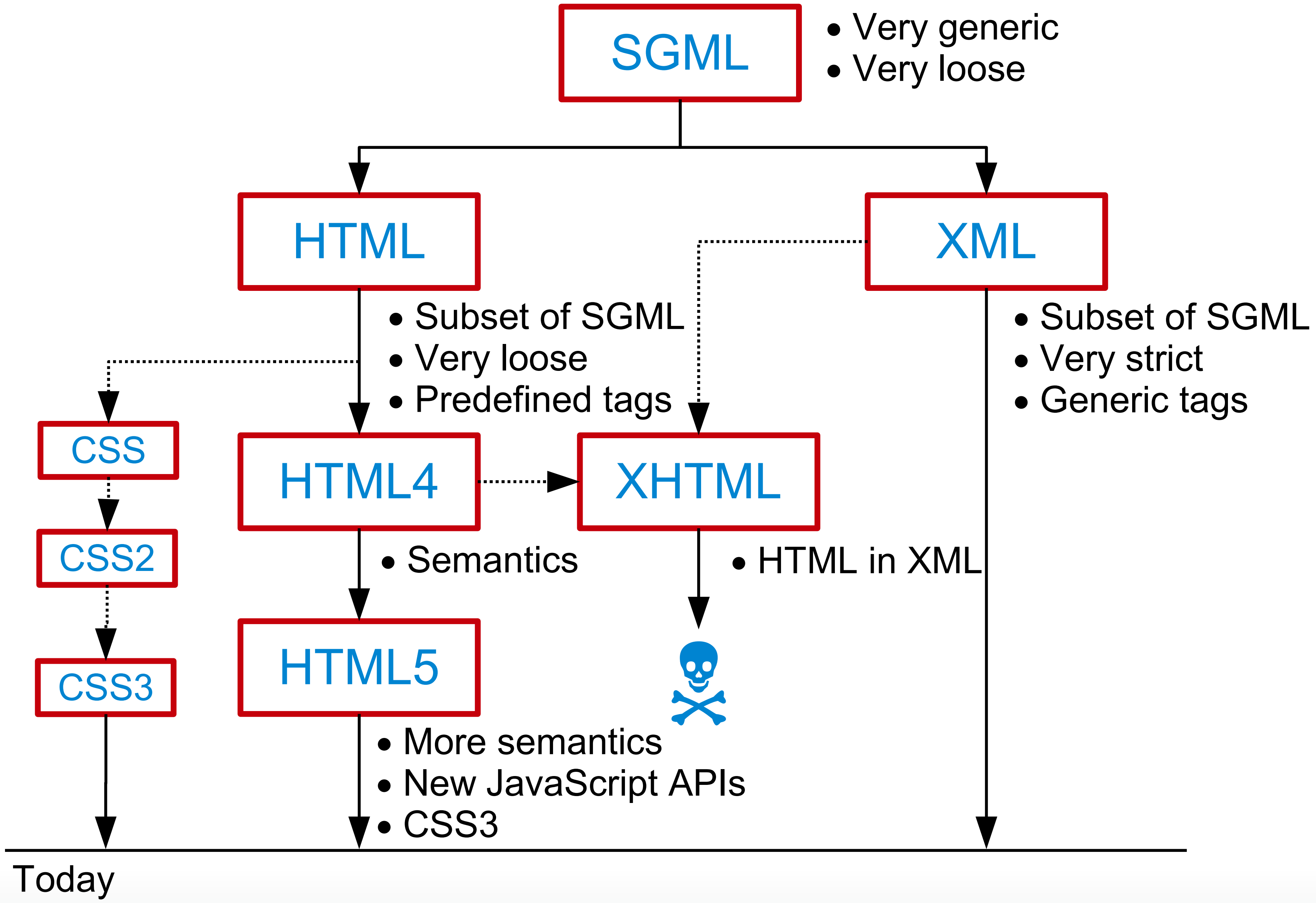
```
<%@ %> <%= %> <% %>
```

- This means JSP pages can not be authored or parsed by general purpose XML parsing technology
  - You need a specially adapted JSP parser
- This can be considered as a disadvantage

Analogy to JSP, HTML and XML

- In the past, attempts have been made to unify XML and HTML: XHTML
  - This allows HTML documents to be authored using strict XML syntax
  - Any XML document can be parsed and authored with any XML parser technology
- In the same light, there have been attempts to unify XML and JSP: JSPX
  - Writing JSP documents as XHTML with additional XML namespace declarations allows JSP documents to be just another XML vocabulary

Evolution of HTML and XML



REALDOLMEN

The end of XHTML

- Unfortunately, XHTML has died out
  - The new HTML 5 specification builds on top of HTML 4, not XHTML
  - The W3C officially marked XHTML 2.0 as “deprecated”
- This means JSPX is now in an uncomfortable position
  - They are essentially a variant of XHTML documents
  - Java's enterprise MVC framework, “JavaServer Faces” faces the same problem
    - It uses XHTML as it's template mechanism (these are called “facelets”)

JSPX: Pros and cons

- Regardless of the position of JSPX, it has some advantages and disadvantages to JSP
  - (+) JSPX documents is a strict XML vocabulary
    - All existing XML infrastructure will “work”
  - (-) Because of this, it enforces strict XML synax, which is clumsy for JavaScript

```
<script>
    if(a < b) {
    }
</script>
```

Invalid in JSPX: must be replaced by &lt;
Same problem for:
 & → &amp;
 > → &gt;
 ...

- With all things considered, the choice is yours!
  - Some projects use JSPX, others use JSP

Creating a JSPX document

- JSPX documents have the “.JSPX” file extension

```
<?xml version="1.0" encoding="utf-8"?>

<jsp:root xmlns="http://www.w3.org/1999/xhtml"
        xmlns:jsp="http://java.sun.com/JSP/Page"
        xmlns:c="http://java.sun.com/jsp/jstl/core"
        xmlns:rd="urn:jsptagdir:/WEB-INF/tags"
        version="2.0">

  <jsp:directive.page contentType="text/html" pageEncoding="UTF-8" />

  <html>
    <body>
      Today is: <jsp:expression>new java.util.Date()</jsp:expression>
    </body>
  </html>
</jsp:root>
```

Porting JSP to JSPX

- Apart from JSPX being a strict XML document, the following table shows how to migrate from/to JSP

Syntax Elements	Standard Syntax	XML Syntax
Comments	`<%-- ... --%>`	`<!-- ... -->`
Declarations	`<%! ... %>`	`<jsp:declaration> ... </jsp:declaration>`
Directives	`<%@ include ... %>`	`<jsp:directive.include ... />`
	`<%@ page ... %>`	`<jsp:directive.page ... />`
	`<%@ taglib ... %>`	`xmlns:prefix="tag library URL"`
Expressions	`<%= ... %>`	`<jsp:expression>...</jsp:expression>`
Scriptlets	`<% ... %>`	`<jsp:scriptlet>...</jsp:scriptlet>`

REALDOLMEN

Exercise

- Open, configure and run the CourseList project
  - Extend the project so that a press on the ‘Reset’ button in the CourseList.jsp page will contact the server
    - The server should then reset the number of participants for all the inscriptions to 0
  - Modify the Invoice.jsp page so that is shows not only the total sum of the invoice, but also a list of all the ordered courses with their full information (title of the course, number of participants inscribed, unit price, subtotal for that course)
    - Do not show any information if there are no participants for the course
  - Insert a confirmation page in the flow
    - This page should be shown in between the EnterContactPerson.jsp page and the Invoice.jsp page
    - It should list the contactperson information together with all the information for the invoice
    - There should also be a ‘Next’ and ‘Previous’ button on this screen
    - This page is used to do a final verification before submitting the order

Exercise

- Add some input validation to the CourseList.jsp page
  - E.g., the number should be a valid number, greater than or equal to 0 and less than 12
  - Show the error on the CourseList.jsp page
  - Note you have to check on both the recalculate and next buttons
- When done, try to improve the exercise: make use of JSTL and EL
  - You could also add the Command Design Pattern to reduce the code in the CustomerServlet

REALDOLMEN

Java Servlets and JSP Programming

Summary

Summary

- In J2EE a component / container model is used to create web applications
- J2EE contains two complementary technologies, Servlets and JSP that output HTML or other formats over HTTP
- Servlets are Java centric
- JSP are HTML centric
- Use the Model-View-Controller pattern to create a web application architecture that is flexible, and easier to maintain
- JSTL provides in an additional mechanism to keep Java code out of your JSP pages