Lecture Overview

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

Architecture

- A specification that identifies components and their associated functionality, describes connectivity of components, and describes the mapping of functionality onto components. Architectures can be of different types, e.g., hardware, software, or system, and can be domain-specific, e.g., networking.
- The manner in which hardware or software is structured. Architecture typically describes how the system or program is constructed, how it fits together, and the protocols and interfaces used for communication and cooperation among modules or components of the system.

Components of Data-Intensive Systems

Three separate types of functionality:

- Data management
- Application logic
- Presentation

- The system architecture determines whether these three components reside on a single system (“tier) or are distributed across several tiers

Single-Tier Architectures

All functionality combined into a single tier, usually on a mainframe

- User access through dumb terminals

Advantages:

- Easy maintenance and administration

Disadvantages:

- Today, users expect graphical user interfaces.
- Centralized computation of all of them is too much for a central system

Client-Server Architectures

Work division: Thin client

- Client implements only the graphical user interface
- Server implements business logic and data management

- Work division: Thick client
  - Client implements both the graphical user interface and the business logic
  - Server implements data management

Types of Clients

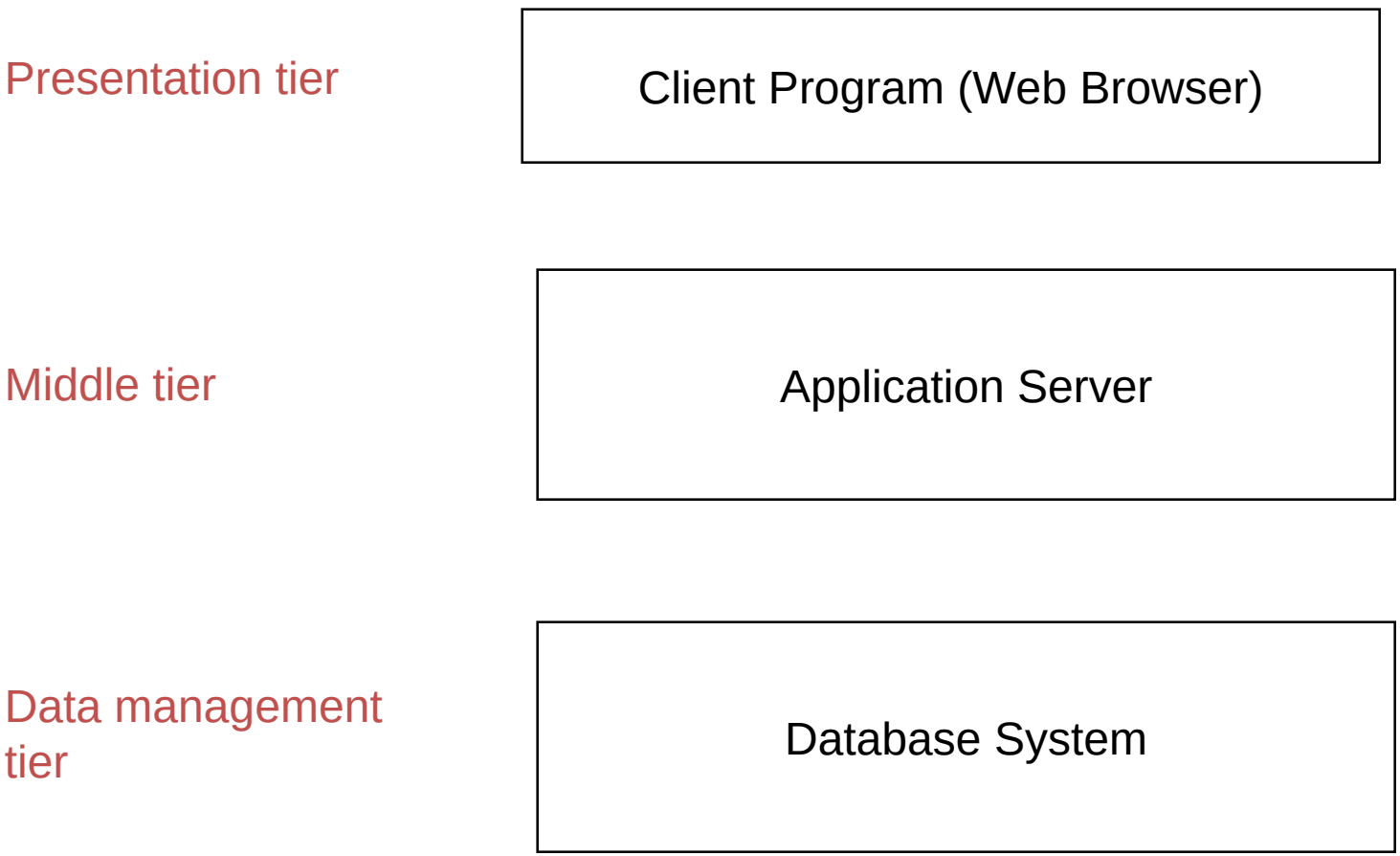
- Thin client
  - HTML, XML, Javascript
  - No processing
  - Best suited for load distribution to server
  - Client response slower
- Thick client
  - Application clients and applets
  - Better responsiveness
  - Difficult to maintain
  - Heavy weights
- Smart Clients
  - Provide best of both worlds
  - Swing or SWT Based
  - Requires Java web Start

Client-Server Architectures (Contd.)

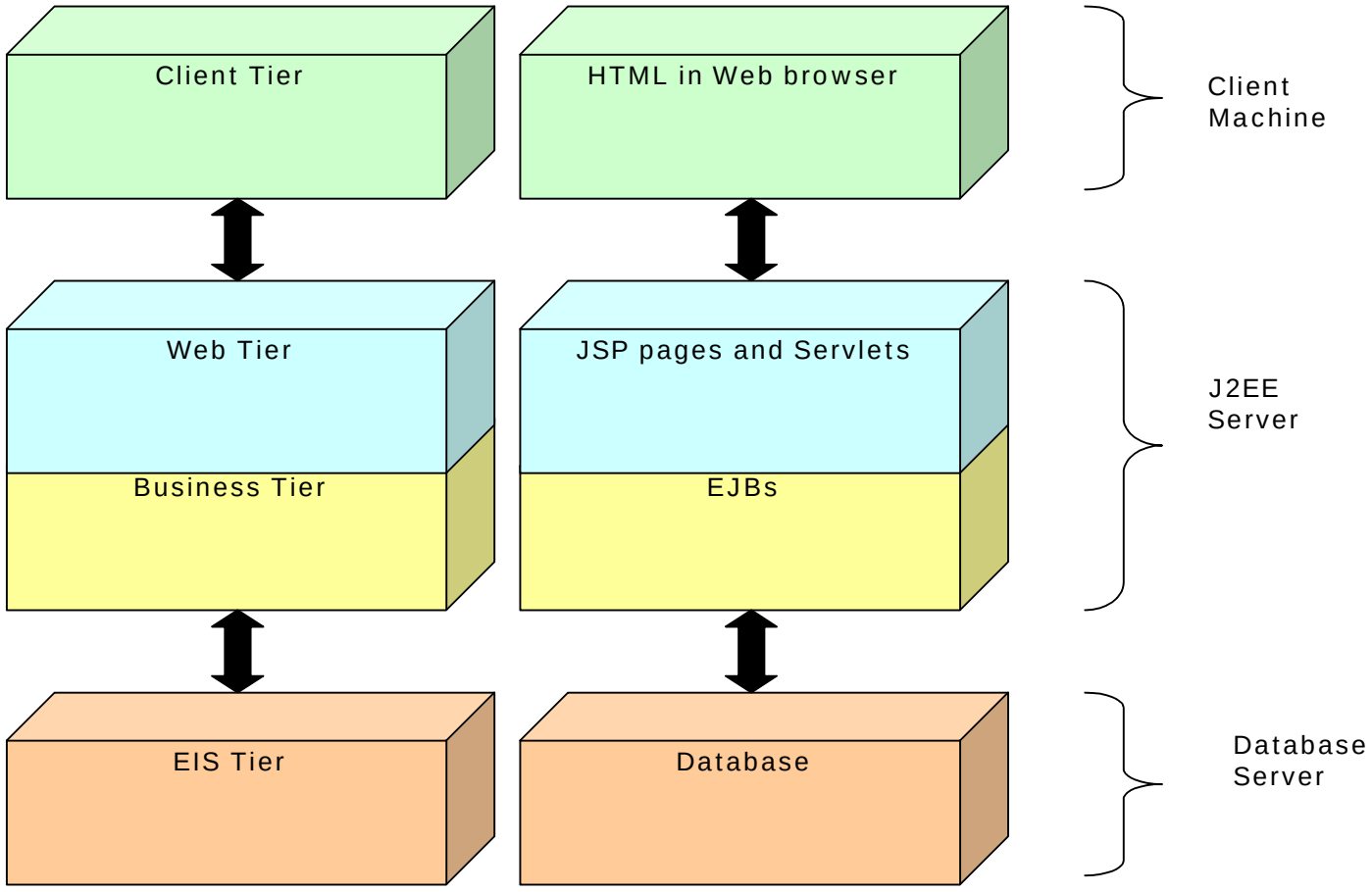
Disadvantages of thick clients

- No central place to update the business logic
- Security issues: Server needs to trust clients
  - Access control and authentication needs to be managed at the server
  - Clients need to leave server database in consistent state
  - One possibility: Encapsulate all database access into stored procedures
- Does not scale to more than several 100s of clients
  - Large data transfer between server and client
  - More than one server creates a problem: x clients, y servers: x*y connections

The Three-Tier Architecture



Multi-tiered Architecture

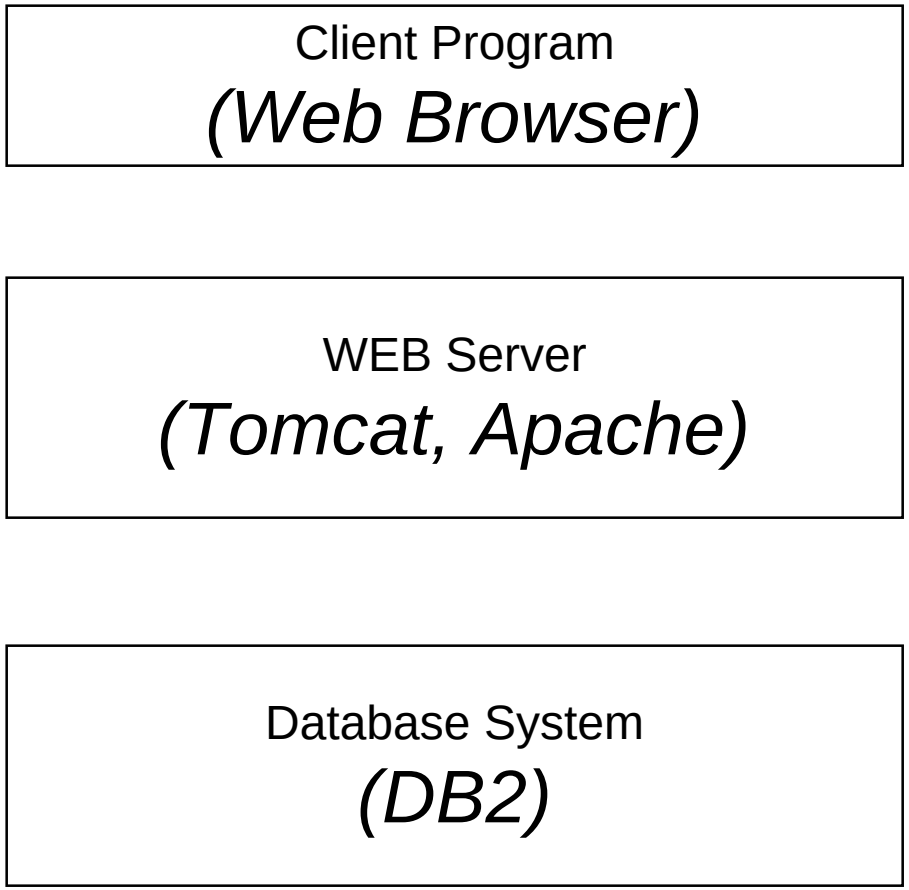


Multi-tier architecture VS **J2EE layers**

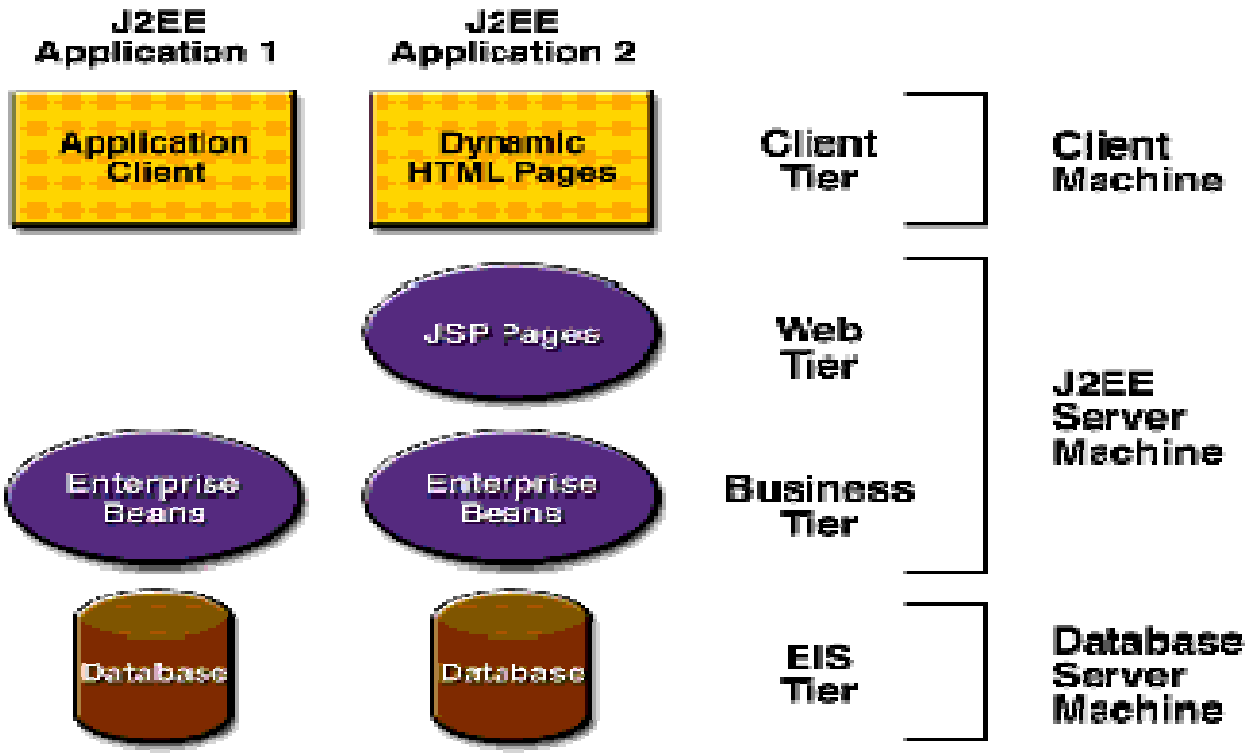
Example 2: Course Enrollment

- Build a system using which students can enroll in courses
- Database System
  - Student info, course info, instructor info, course availability, pre-requisites, etc.
- Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output

Technologies



Multitiered Applications



J2EE Components

- J2EE applications are made up of components.
- A *J2EE component* is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components.
- The J2EE specification defines the following J2EE components:
  - Application clients and applets are components that run on the client.
  - Java Servlet and JavaServer Pages (JSP) technology components are web components that run on the server.
  - Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server.

J2EE Components

- J2EE components are written in the Java programming language and are compiled in the same way as any program in the language.

- The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, are verified to be well formed and in compliance with the J2EE specification, and are deployed to production, where they are run and managed by the J2EE server.

Advantages of the Three-Tier Architecture

- Heterogeneous systems
  - Tiers can be independently maintained, modified, and replaced
- Thin clients
  - Only presentation layer at clients (web browsers)
- Integrated data access
  - Several database systems can be handled transparently at the middle tier
  - Central management of connections
- Scalability
  - Replication at middle tier permits scalability of business logic
- Software development
  - Code for business logic is centralized
  - Interaction between tiers through well-defined APIs: Can reuse standard components at each tier

Overview of the Presentation Tier

- Recall: Functionality of the presentation tier
  - Primary interface to the user
  - Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)
  - Simple functionality, such as field validity checking
- We will cover:
  - HTML Forms: How to pass data to the middle tier
  - JavaScript: Simple functionality at the presentation tier
  - Style sheets: Separating data from formatting

Inside HTML Forms

- INPUT tag
  - Attributes:
    - TYPE: text (text input field), password (text input field where input is, reset (resets all input fields)
    - NAME: symbolic name, used to identify field value at the middle tier
    - VALUE: default value
  - Example: <INPUT TYPE="text" Name="title">
- Example form:

```
<form method="POST" action="TableOfContents.jsp">
    <input type="text" name="userid">
    <input type="password" name="password">
    <input type="submit" value="Login" name="submit">
    <input type="reset" value="Clear">
</form>
```

Passing Arguments

Two methods: GET and POST

- GET
  - Form contents go into the submitted URI
  - Structure:
    action?name1=value1&name2=value2&name3=value3
    - Action: name of the URI specified in the form
    - (name,value)-pairs come from INPUT fields in the form; empty fields have empty values (“name=“)
  - Example from previous password form:
    TableOfContents.jsp?userid=john&password=johnpw
  - Note that the page named action needs to be a program, script, or page that will process the user input

HTML Forms: A Complete Example

```
<form method="POST" action="TableOfContents.jsp">
    <table align = "center" border="0" width="300">
    <tr>
        <td>Userid</td>
        <td><input type="text" name="userid" size="20"></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password" name="password" size="20"></td>
    </tr>
    <tr>
        <td align = "center"><input type="submit" value="Login“
                name="submit"></td>
    </tr>
    </table>
</form>
```

HTML Forms: A Complete Example

```
<form method="POST" action="TableOfContents.jsp">
    <table align = "center" border="0" width="300">
    <tr>
        <td>Userid</td>
        <td><input type="text" name="userid" size="20"></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password" name="password" size="20"></td>
    </tr>
    <tr>
        <td align = "center"><input type="submit" value="Login“
                name="submit"></td>
    </tr>
    </table>
</form>
```

JavaScript

- HTML Revisited
- Need for Client Side Scripting Languages
- Overview of JavaScript
- JavaScript v/s Java
- Embedding JavaScript in HTML
- Server Side JavaScript v/s Client Side JavaScript

JavaScript v/s Java

- JavaScript is not Java
- JavaScript is a scripting language that is parsed and executed by the parser
- Java is programming language developed by Sun Microsystems and JavaScript is by Netscape
- But both of them are based on OOP

JavaScript v/s Java

- Java programming can create fully stand-alone events
- Java cannot run as text. It must be compiled into machine language before it can be run
- JavaScript cannot stand alone
- JavaScript Must be running inside of a web page.
- Browser which is showing web page must understand JavaScript

Embedding JavaScript in HTML

- Can be anywhere inside your HTML
- Enclosed within the <SCRIPT language="javascript"> ....</script>
- Should be in the <HEAD> of your HTML document

Server Side JavaScript v/s Client Side JavaScript

- Client Side Scripting
  - A script that is executed by the browser on a user's computer
  - Part of the Page. Sent to the browser when user requests the page
  - Mostly run in response to an event
    - e.g. Click events, validations
  - Direct interaction with client

JavaScript

- Goal: Add functionality to the presentation tier.
- Sample applications:
  - Detect browser type and load browser-specific page
  - Form validation: Validate form input fields
  - Browser control: Open new windows, close existing windows (example: pop-up ads)
- Usually embedded directly inside the HTML with the <SCRIPT> ... </SCRIPT> tag.
- <SCRIPT> tag has several attributes:
  - LANGUAGE: specifies language of the script (such as javascript)
  - SRC: external file with script code
  - Example:
    <SCRIPT LANGUAGE="JavaScript" SRC="validate.js>
    </SCRIPT>

JavaScript (Contd.)

- If <SCRIPT> tag does not have a SRC attribute, then the JavaScript is directly in the HTML file.
- Example:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- alert("Welcome to our bookstore")
//-->
</SCRIPT>
```

- Two different commenting styles
  - <!-- comment for HTML, since the following JavaScript code should be ignored by the HTML processor
  - // comment for JavaScript in order to end the HTML comment

JavaScript (Contd.)

- JavaScript is a complete scripting language
  - Variables
  - Assignments (=, +=, ...)
  - Comparison operators (<,>,...), boolean operators (&&, ||, !)
  - Statements
    - if (condition) {statements;} else {statements;}
    - for loops, do-while loops, and while-loops
  - Functions with return values
    - Create functions using the function keyword
    - f(arg1, ..., argk) {statements;}

JavaScript: A Complete Example

HTML Form:

```
<form method="POST“
  action="TableOfContents.jsp">
  <input type="text"
    name="userid">
  <input type="password"
    name="password">
  <input type="submit"
    value="Login“ name="submit">
  <input type=“reset”
    value=“Clear”>
</form>
```

Associated JavaScript:

```
<script language="javascript">
function testLoginEmpty()
{
  loginForm = document.LoginForm
  if ((loginForm.userid.value == "") ||
     (loginForm.password.value == ""))
  {
    alert('Please enter values for userid and
      password.');
    return false;
  }
  else return true;
}
</script>
```

Stylesheets

- Idea: Separate display from contents, and adapt display to different presentation formats
- Two aspects:
  - Document transformations to decide what parts of the document to display in what order
  - Document rending to decide how each part of the document is displayed
- Why use stylesheets?
  - Reuse of the same document for different displays
  - Tailor display to user's preferences
  - Reuse of the same document in different contexts
- Two stylesheet languages
  - Cascading style sheets (CSS): For HTML documents
  - Extensible stylesheet language (XSL): For XML documents

CSS: Cascading Style Sheets

- Defines how to display HTML documents
- Many HTML documents can refer to the same CSS
  - Can change format of a website by changing a single style sheet
  - Example:
    <LINK REL="style sheet" TYPE="text/css" HREF="books.css"/>

Each line consists of three parts:

selector {property: value}

- Selector: Tag whose format is defined
- Property: Tag's attribute whose value is set
- Value: value of the attribute

CSS: Cascading Style Sheets

Example style sheet:

```
body {background-color: yellow}
h1 {font-size: 36pt}
h3 {color: blue}
p {margin-left: 50px; color: red}
```

The first line has the same effect as:

```
<body background-color=“yellow>
```

XSL

- Language for expressing style sheets
  - More at: http://www.w3.org/Style/XSL/
- Three components
  - XSLT: XSL Transformation language
    - Can transform one document to another
    - More at http://www.w3.org/TR/xslt
  - XPath: XML Path Language
    - Selects parts of an XML document
    - More at http://www.w3.org/TR/xpath
  - XSL Formatting Objects
    - Formats the output of an XSL transformation
    - More at http://www.w3.org/TR/xsl/

Lecture Overview

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

Overview of the Middle Tier

- Recall: Functionality of the middle tier
  - Encodes business logic
  - Connects to database system(s)
  - Accepts form input from the presentation tier
  - Generates output for the presentation tier
- We will cover
  - CGI: Protocol for passing arguments to programs running at the middle tier
  - Application servers: Runtime environment at the middle tier
  - Servlets: Java programs at the middle tier
  - JavaServerPages: Java scripts at the middle tier
  - Maintaining state: How to maintain state at the middle tier. Main focus: Cookies.

CGI: Common Gateway Interface

- Goal: Transmit arguments from HTML forms to application programs running at the middle tier
- Details of the actual CGI protocol unimportant → libraries implement high-level interfaces

- Disadvantages:
  - The application program is invoked in a new process at every invocation (remedy: FastCGI)
  - No resource sharing between application programs (e.g., database connections)
  - Remedy: Application servers

CGI: Example

- HTML form:

```
<form action="findbooks.cgi" method=POST>
Type an author name:
<input type="text" name="authorName">
<input type="submit" value="Send it">
<input type="reset" value="Clear form">
</form>
```

- Perl code:

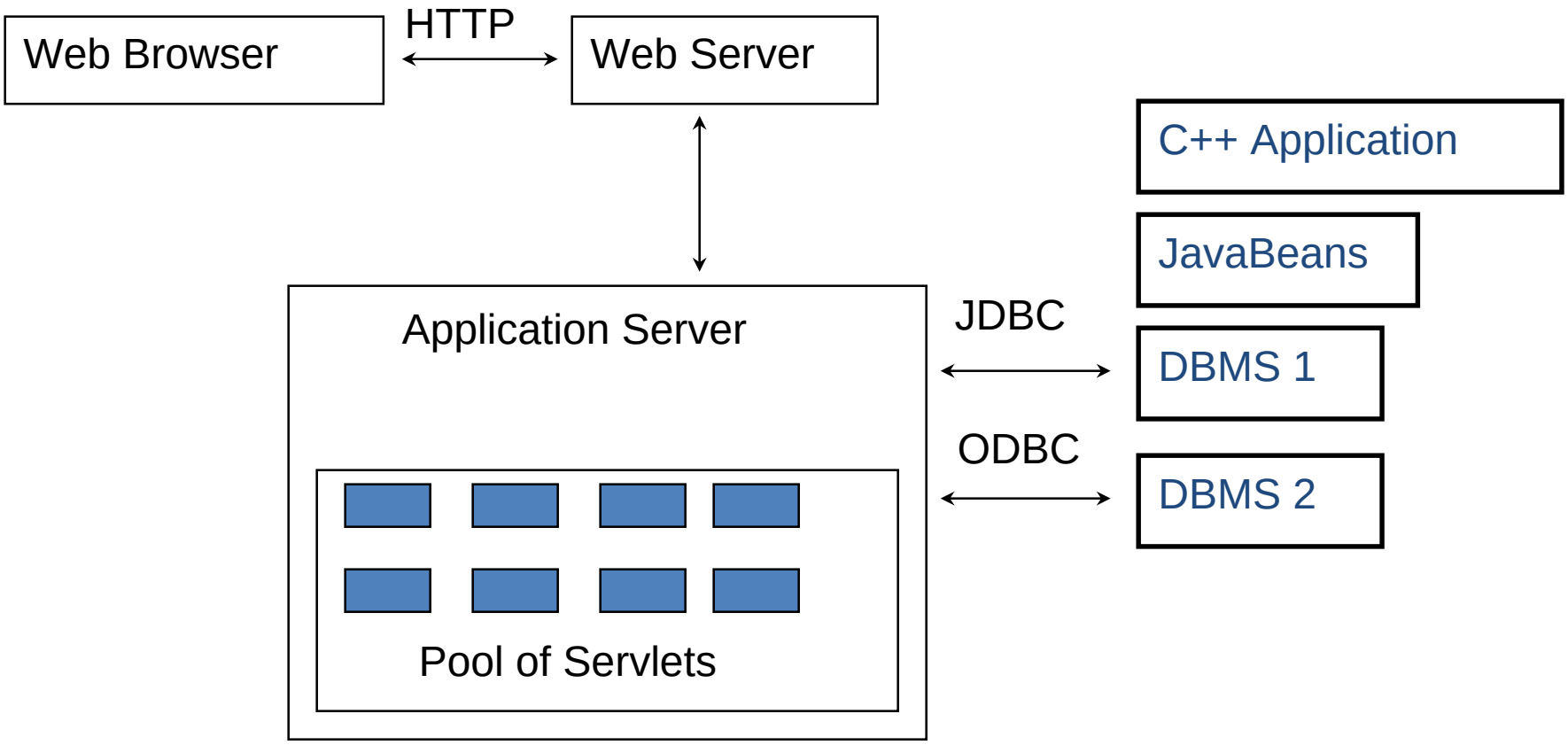
```
use CGI;
$dataIn=new CGI;
$dataIn->header();
$authorName=$dataIn->param('authorName');
print("<HTML><TITLE>Argument passing test</TITLE>");
print("The author name is " + $authorName);
print("</HTML>");
exit;
```

Application Servers

- Idea: Avoid the overhead of CGI
  - Main pool of threads of processes
  - Manage connections
  - Enable access to heterogeneous data sources
  - Other functionality such as APIs for session management

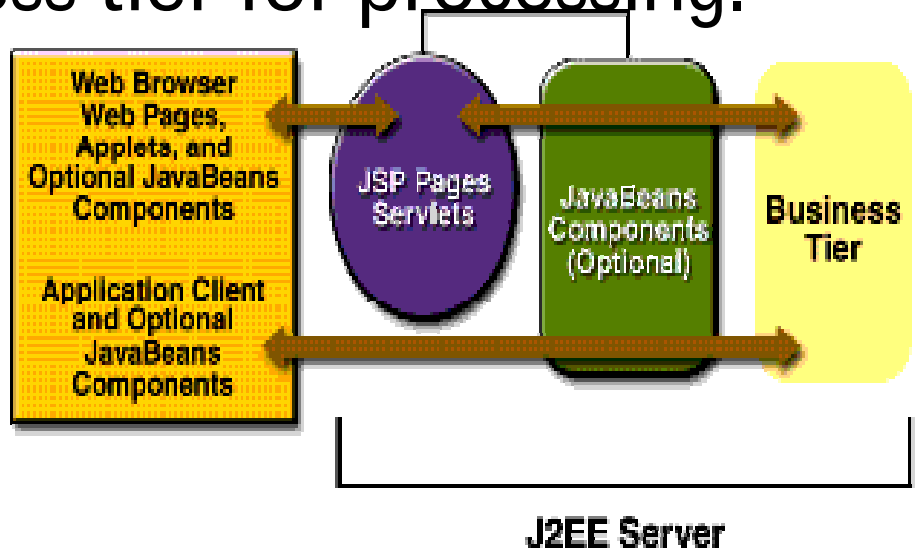
Application Server: Process Structure

Process Structure

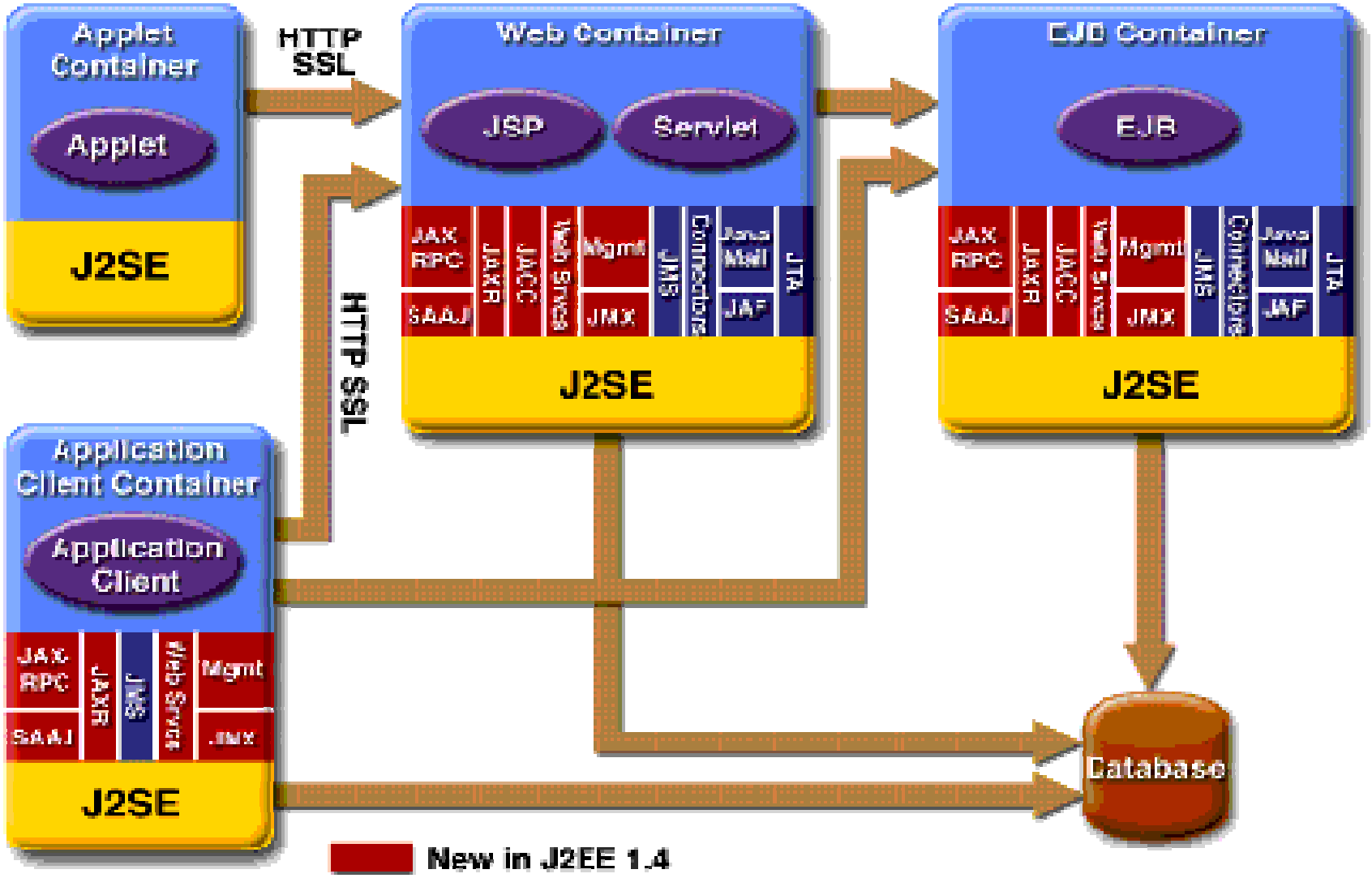


Web Components

- As shown in Figure, the web tier, like the client tier, might include a JavaBeans component to manage the user input and send that input to enterprise beans running in the business tier for processing.



The J2EE Platform



Servlets : Overview

- Introduction:

  Servlets are objects used to process server-side requests and generate responses for clients. Servlets typically execute within the context of an HTTP Web server. Inside the Web server, servlets process requests from HTTP clients, such as Web browsers.

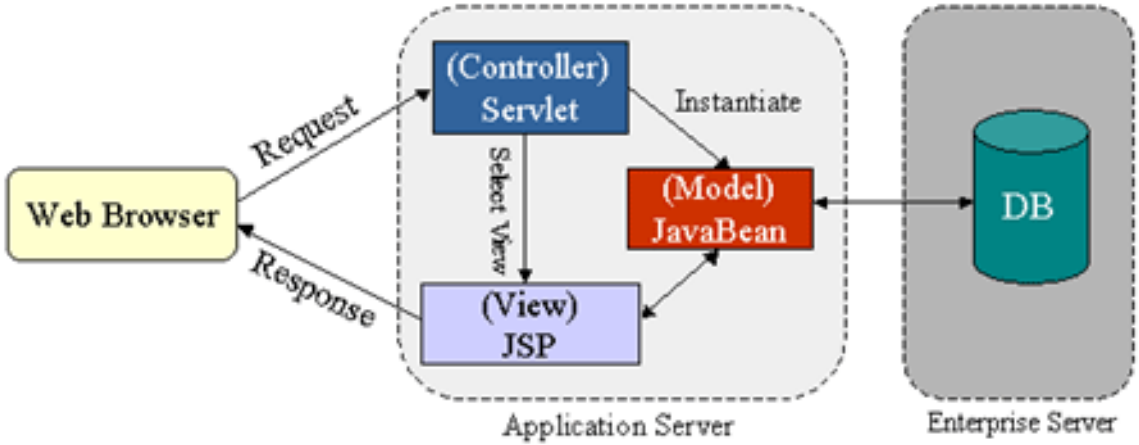
- What Does a Servlets Do?

  - Unlike JavaScript applets, which run in the context of a Web browser and are a client-side technology, Servlets run in the context of a Web server and are a server-side technology.
  - Allow the exchange of data from the Web server to the Web browser.
  - Servlets load only once into the server's memory, and hence are more efficient than other technologies.

Servlet Advantages over older technologies

- **Efficient.** With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.
- **Convenient.** Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such tasks.
- **Powerful.** Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.
- **Portable.** Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.
- **Inexpensive.** There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites.

Request – Request Paradigm



This model promotes the use of the Model View Controller (MVC) architectural style design and pattern.

- JSP pages are used for the presentation layer and servlets for processing tasks.
- The servlet acts as a *controller* responsible for processing requests and creating any beans needed by the JSP page.
- The controller is also responsible for deciding to which JSP page to forward the request.
- The JSP page retrieves objects created by the servlet and extracts dynamic content for insertion within a template

Creating a simple Servlet

The servlet API provides the *javax.servlet.http.HttpServlet* class - as a starting point.

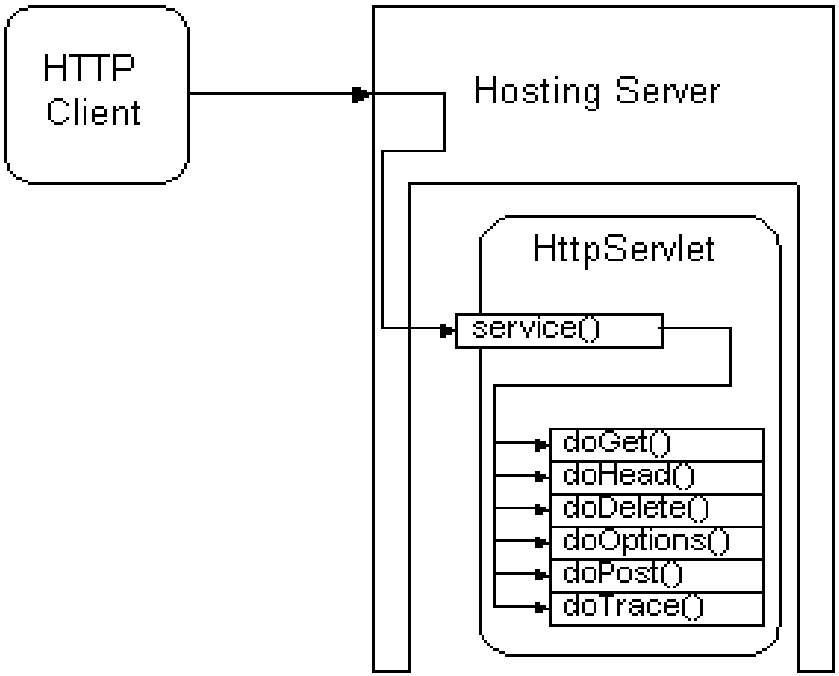
This module introduces you to the HttpServlet class and should provide a basic understanding of how to use the HttpServlet class to generate Web pages programmatically for HTTP clients.

Methods of the HTTP Servlet Class

Important methods of HTTP servlet classes are:

- doGet: to be used if the servlet supports HTTP GET requests
- doPost: to be used for HTTP POST requests
- init and destroy: to be used to manage resources that are held for the life of the servlet
- getServletInfo: the servlet uses this method to provide information about itself, usually via the Servlet container

Creating a simple Servlet



Servlets: A Complete Example

```
public class ReadUserName extends HttpServlet {
    public void doGet(              HttpServletRequest request,
                                    HttpSevletResponse response)
            throws ServletException, IOException {
            reponse.setContentType("text/html");
            PrintWriter out=response.getWriter();
            out.println("<HTML><BODY>\n <UL> \n" +
                        "<LI>" + request.getParameter("userid") + "\n" +
                        "<LI>" + request.getParameter("password") + "\n" +
                        "<UL>\n<BODY></HTML>");
    }
    public void doPost(             HttpServletRequest request,
                                    HttpSevletResponse response)
            throws ServletException, IOException {
            doGet(request,response);
    }
}
```

Creating a simple Servlet (contd)

- Any HTTP Url takes the following form
  - http://[host][:port]/[request path]?[query String]

- Request path contains
  - Context path
  - Servlet path
  - Path info
  - Query String containing the parameters and values.

Deploying a Servlet

- A Servlet class file goes in the /WEB-INF/classes
- For a client to access a Servlet, a unique URL, or set of URLs, needs to be declared in the Web Application Deployment Descriptor
- Open up the /WEB-INF/web.xml and edit it

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4">
        <servlet>
                <servlet-name>HelloWorld</servlet-name>
                <servlet-class>com.myexample.HelloWorld</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>HelloWorld</servlet-name>
                <url-pattern>/HelloWorld</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>welcome.html</welcome-file>
        </welcome-file-list>
</web-app>
```

Structure – What goes where?

- \jakarta-tomcat-5\webapps
  - **\AdvancedJava**
    - **Welcome.html**
    - **MyFirstHTML.html**
    - **\WEB-INF**
      - **Web.xml**
      - **\classes**
        - » **\com**
        - » **\myexample**
        - » **HelloWorld.class**

Form Servlet Integration

Example source HTML for a form:

Maps to mapping in web.xml refer

```
< FORM METHOD=POST ACTION= “test.view" >
    Enter your name:< INPUT <INPUT TYPE=text NAME=your_name>
    < INPUT TYPE=submit VALUE="Test this form" >
< /FORM >
```

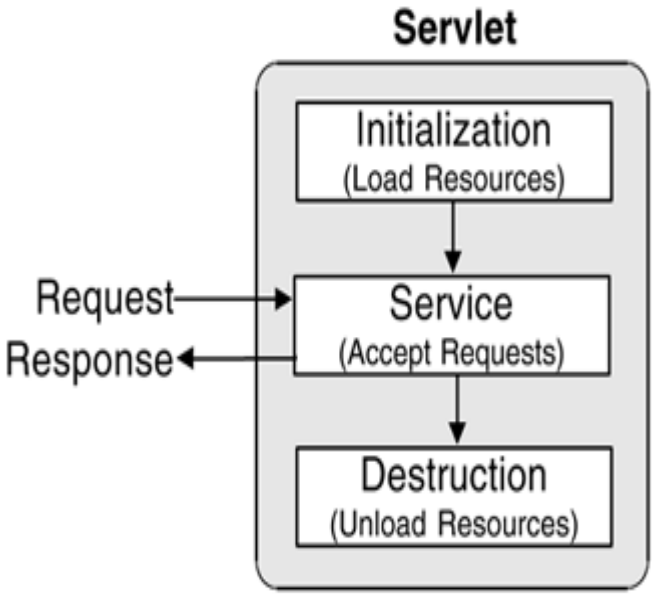
When the user clicks the submit button, the browser collects the values of each of the input fields (text typed by the user, etc.) and sends them to the web server identified in the ACTION keyword of the FORM open tag (and that web server passes that data to the program identified in the ACTION, using the METHOD specified).

Action is mapped in web.xml file to call the respective servlet to handle the user request. Otherwise Form “Action” can directly be mapped to open another html page.

Note: Each form must have exactly one field of type submit

The Servlets Life Cycle

- When a request is mapped to the servlet the container
  - Looks for the instance of the servlet class
  - If no exists loads the servlet class
  - Instantiates it
  - Initializes it by calling the init method.
  - Invokes the service method
  - When the servlet needs to be removed the container
  - Finalizes the servlet by calling the destroy method.



Java Server Pages

- Servlets
  - Generate HTML by writing it to the “PrintWriter” object
  - Code first, webpage second
- JavaServerPages
  - Written in HTML, Servlet-like code embedded in the HTML
  - Webpage first, code second
  - They are usually compiled into a Servlet

Advantages of JSP

- Separating content generation from presentation
- Emphasizing reusable components
- Simplifying page development with tags

Advantages over other technologies

Vs asp

it is portable to other operating systems and non-Microsoft Web servers

Vs servlet

- your Web page design experts can build the HTML, leaving places for your servlet programmers to insert the dynamic content

JavaServerPages: Example

```
<html>
<head><title>Welcome to B&N</title></head>
<body>
    <h1>Welcome back!</h1>
    <% String name="NewUser";
            if (request.getParameter("username") != null) {
                    name=request.getParameter("username");
            }
    %>
    You are logged on as user <%=name%>
    <p>
</body>
</html>
```

JSP Environment

- Set your CLASSPATH
- Compile your code
- Use packages to avoid name conflicts

- Put JSP page in special directory
  - *install_dir*\webapps\ROOT\ (HTML *and* JSP -- Tomcat)
  - *install_dir*\servers\default\default-app (JRun)

- Use special URLs to invoke JSP page
  - Use same URLs as for HTML pages (except for file extensions)

- Caveats
  - Previous rules about CLASSPATH, install dirs, etc., still apply to regular Java classes used by a JSP page

Maintaining State

HTTP is stateless.

- Advantages
  - Easy to use: don't need anything
  - Great for static-information applications
  - Requires no extra memory space
- Disadvantages
  - No record of previous requests means
    - No shopping baskets
    - No user logins
    - No custom or dynamic content
    - Security is more difficult to implement

Application State

- Server-side state
  - Information is stored in a database, or in the application layer's local memory
- Client-side state
  - Information is stored on the client's computer in the form of a cookie
- Hidden state
  - Information is hidden within dynamically created web pages

Application State

Server-Side State

- Many types of Server side state:
- 1. Store information in a database
  - Data will be safe in the database
  - BUT: requires a database access to query or update the information
- 2. Use application layer's local memory
  - Can map the user's IP address to some state
  - BUT: this information is volatile and takes up lots of server main memory

5 million IPs = 20 MB

Server-Side State

- Should use Server-side state maintenance for information that needs to persist
  - Old customer orders
  - “Click trails” of a user’s movement through a site
  - Permanent choices a user makes

Jsp is a servlet

```
<html><body>

<%@ page language="java" %>
<%! int count = 0;          %>
<%  count++;                %>

Welcome! You are visitor number <%= count %>

</body></html>
```

File counter.jsp

Translation Time

```
// In Generated Servlet

int count = 0;

// in _jspService()

out.write("<html><body>");

count++;

out.write("
Welcome! You are visitor number
");

out.print(count);

out.write("
   </body></html>
");
```

Generated servlet for counter.jsp

Request Time

```
<html><body>

Welcome! You are visitor number
1

</body></html>
```

Output HTML

Client-side State: Cookies

- Storing text on the client which will be passed to the application with every HTTP request.
  - Can be disabled by the client.
  - Are wrongfully perceived as "dangerous", and therefore will scare away potential site visitors if asked to enable cookies[1]
- Are a collection of (Name, Value) pairs

[1]http://www.webdevelopersjournal.com/columns/stateful.html

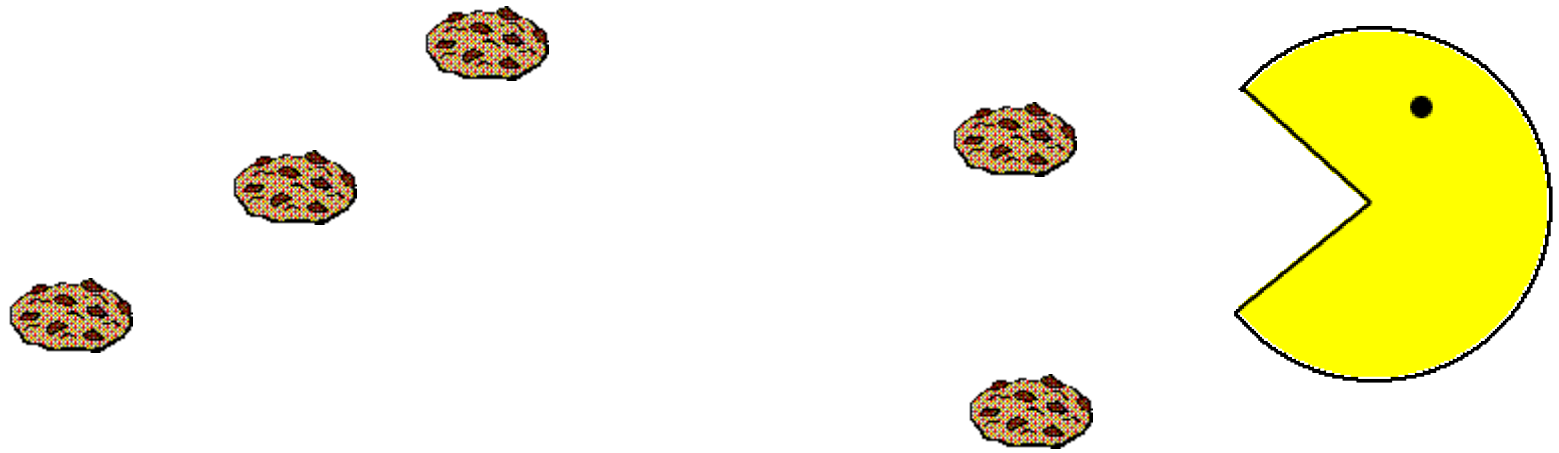
Client State: Cookies

- Advantages
  - Easy to use in Java Servlets / JSP
  - Provide a simple way to persist non-essential data on the client even when the browser has closed
- Disadvantages
  - Limit of 4 kilobytes of information
  - Users can (and often will) disable them
- Should use cookies to store interactive state
  - The current user's login information
  - The current shopping basket
  - Any non-permanent choices the user has made

Creating A Cookie

```
Cookie myCookie =
  new Cookie(“username", “jeffd");
response.addCookie(userCookie);
```

- You can create a cookie at any time



Accessing A Cookie

```
Cookie[] cookies = request.getCookies();
String theUser;
for(int i=0; i<cookies.length; i++) {
  Cookie cookie = cookies[i];
  if(cookie.getName().equals(“username”))
      theUser = cookie.getValue();
}
// at this point theUser == “username”
```

- Cookies need to be accessed BEFORE you set your response header:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

Cookie Features

- Cookies can have
  - A duration (expire right away or persist even after the browser has closed)
  - Filters for which domains/directory paths the cookie is sent to
- See the Java Servlet API and Servlet Tutorials for more information

Hidden State

- Often users will disable cookies
- You can “hide” data in two places:
  - Hidden fields within a form
  - Using the path information
- Requires no “storage” of information because the state information is passed inside of each web page

Hidden State: Hidden Fields

- Declare hidden fields within a form:
  - <input type='hidden' name='user' value='username'/>
- Users will not see this information (unless they view the HTML source)
- If used prolifically, it's a killer for performance since EVERY page must be contained within a form.

What is Session?

- **Session object**
  - Represented by HttpSession object
  - Supports applications that needs to maintain the client state.
  - Accessed via the getSession() method of the request.
  - Object valued atttributes can be associated with session by name.
  - Session.setAttribute(key, Object)
  - An object set as an attribute at the session level, will be available for that user through out his interaction with the server.

Expiring the session

```
<web-app>….
   <servlet>
   …
   </servlet>
   <session-config>
        <session-timeout>15</session-
   timeout>
   </session-config>
   </web-app>
```

Hidden State: Path Information

- Path information is stored in the URL request:

  `http://server.com/index.htm?user=jeffd`

- Can separate ‘fields’ with an & character:

  `index.htm?user=jeffd&preference=pepsi`

- There are mechanisms to parse this field in Java. Check out the `javax.servlet.http.HttpUtils parserQueryString()` method.

Multiple state methods

- Typically all methods of state maintenance are used:
  - User logs in and this information is stored in a cookie
  - User issues a query which is stored in the path information
  - User places an item in a shopping basket cookie
  - User purchases items and credit-card information is stored/retrieved from a database
  - User leaves a click-stream which is kept in a log on the web server (which can later be analyzed)