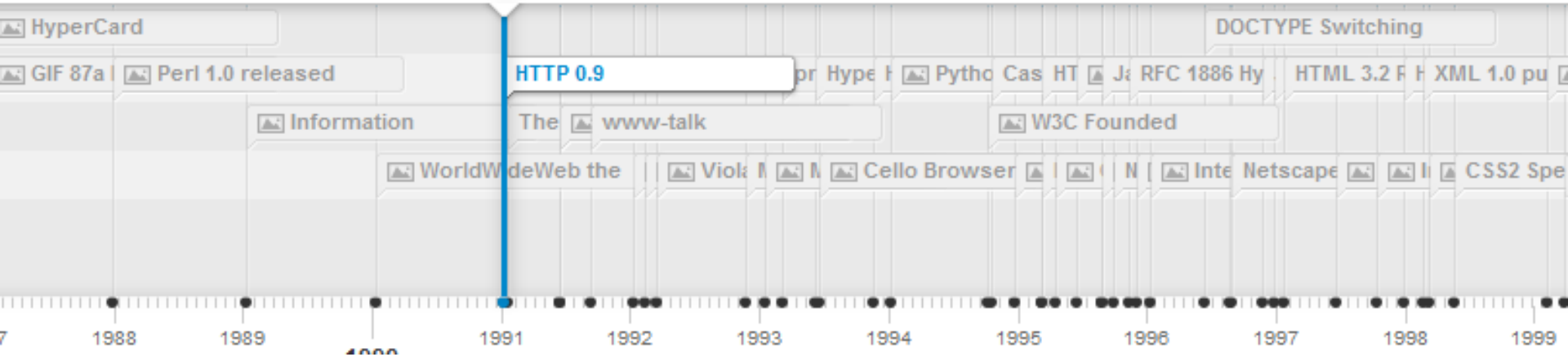
Introduction to Java Web Application Fundamentals

Home of All the Laws of Nature

History of the Web

1991milestones HTTP; HTML

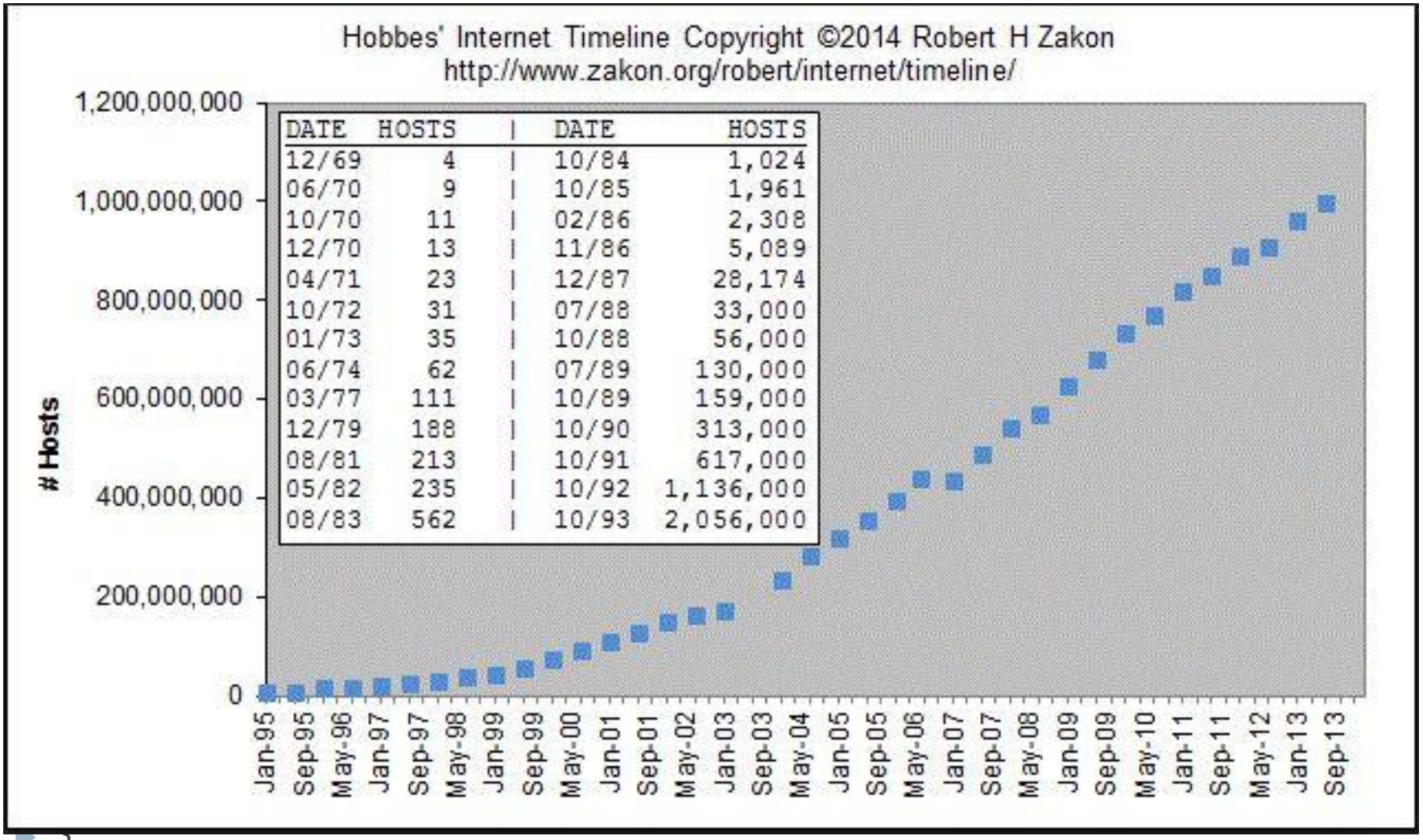


http://www.webdirections.org/history/

The oldest extant web page

The oldest web page continually served was last modified on Tuesday November 13 1990.

Growth of Internet Servers

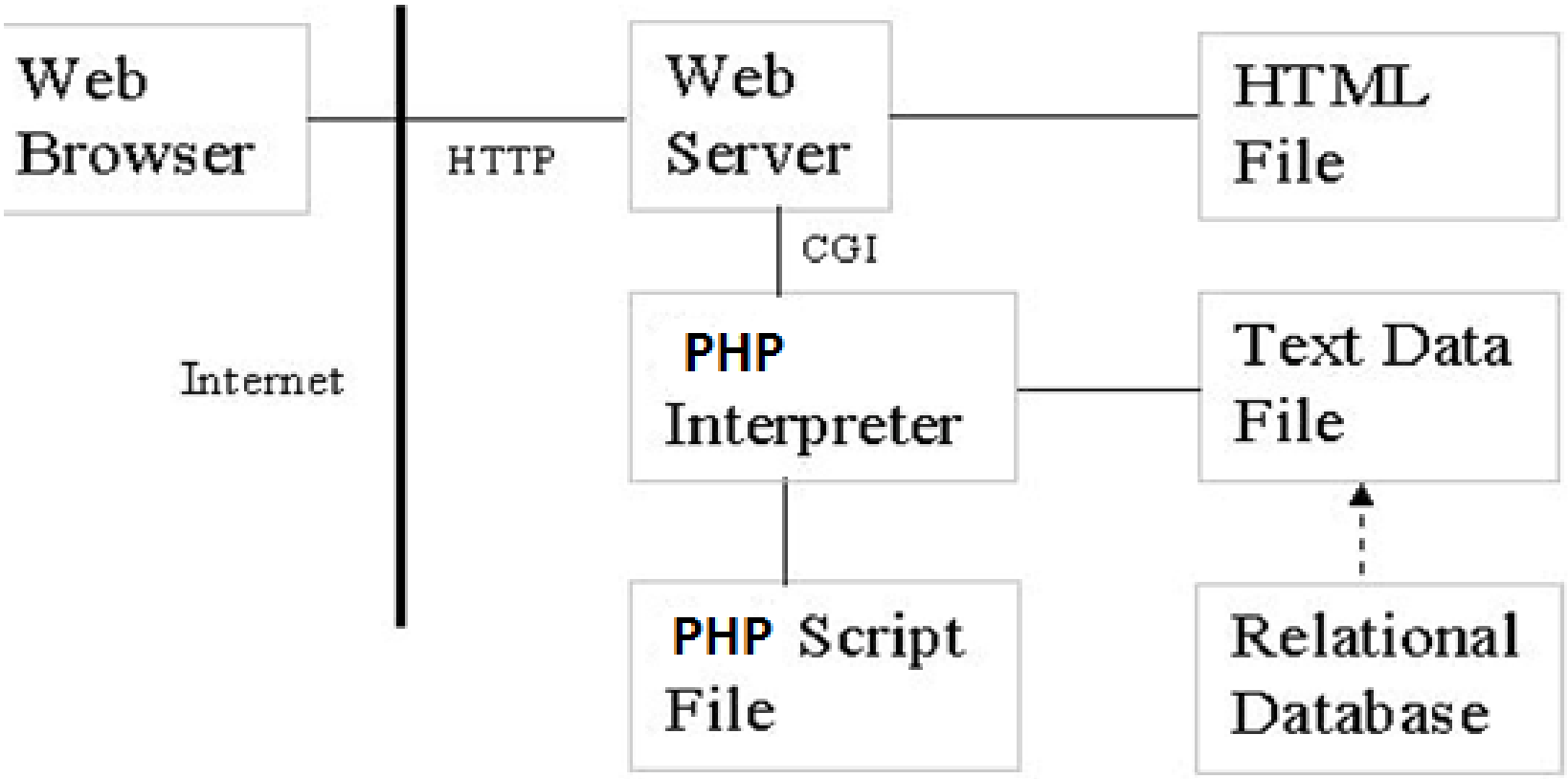


DATE	HOSTS	DATE	HOSTS
12/69	4	10/84	1,024
06/70	9	10/85	1,961
10/70	11	02/86	2,308
12/70	13	11/86	5,089
04/71	23	12/87	28,174
10/72	31	07/88	33,000
01/73	35	10/88	56,000
06/74	62	07/89	130,000
03/77	111	10/89	159,000
12/79	188	10/90	313,000
08/81	213	10/91	617,000
05/82	235	10/92	1,136,000
08/83	562	10/93	2,056,000

Hypertext HTTP HTML

- Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.
  - You can go to any place on the Internet whenever you want by clicking on links — there is no set order to do things in.
- HTTP[Hyper Text Transfer Protocol] is the protocol to exchange or transfer hypertext.
- HTML[Hyper Text Markup Language] is a *Language*, as it has code-words and syntax like any other language.
  - Markup is what **HTML tags** do to the text inside them. They mark it as a certain type of text (*italicised* text, for example).

CGI Platform



Common Gateway Interface (CGI) is a standard method used to generate dynamic content on Web pages and Web applications.

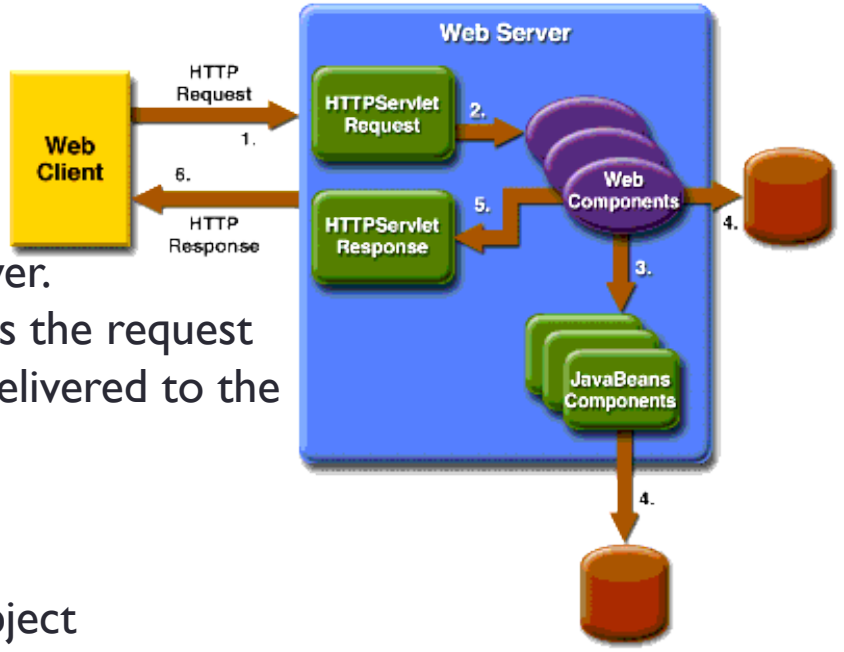
Servlets – An Efficient Technology

- A servlet is server-side java code that can handle http requests and return dynamic content.
- Servlets are managed by a *servlet engine* or *container*.
- Each request that comes in results in the spawning of a new thread that runs a servlet.

(eliminating the cost of creating a new process every time).

Java Web Applications

- In the Java 2 platform, web components provide dynamic extension capabilities for a web server. Servlets, JSPs and JSFs are web components.

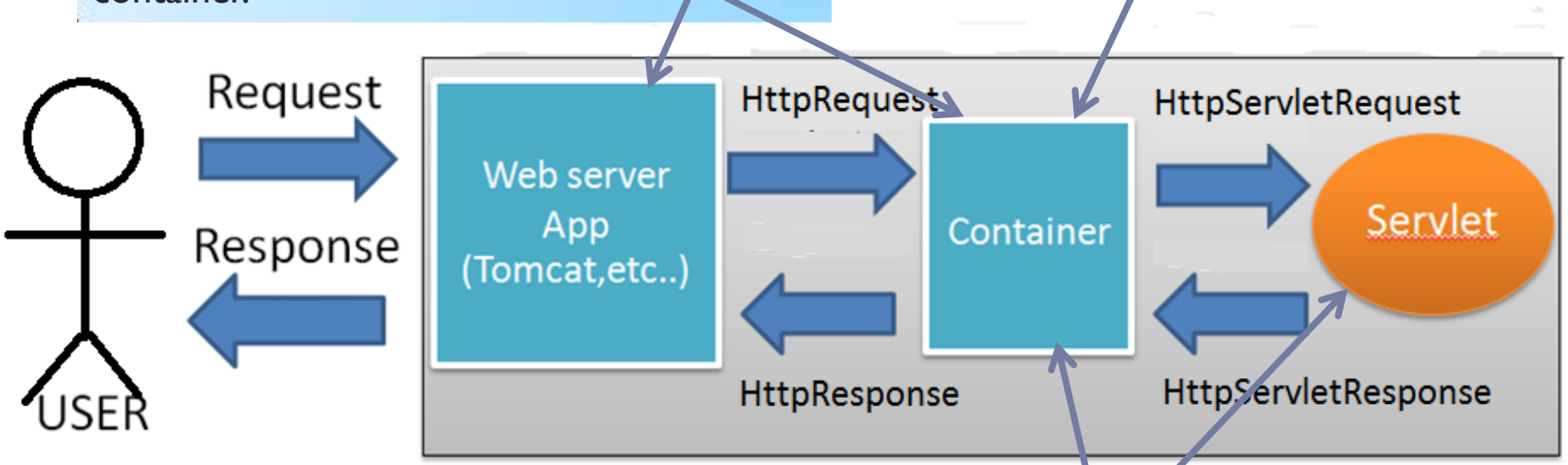


1. Client sends an HTTP request to the web server.
2. Web server [**via Servlet container**] converts the request into an HTTPServletRequest object which is delivered to the Servlet
3. Servlet interacts with JavaBeans components
4. or a database to generate dynamic content
5. Servlet generates an HTTPServletResponse object
6. Web server [**via Servlet container**] converts this object to an HTTP response and returns it to the client

Servlet Container

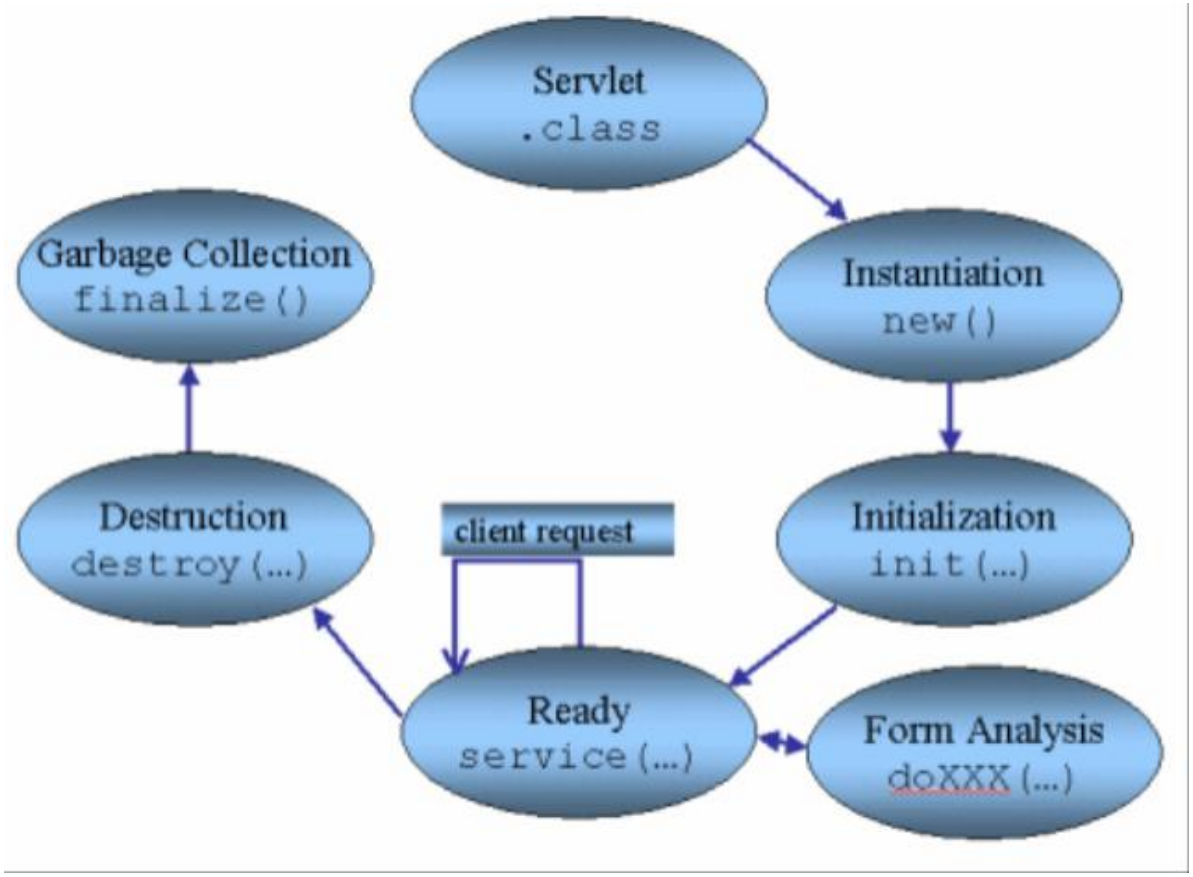
When a request comes to the web server, if the server sees the request is for a servlet, it passes the request data to the servlet container.

Servers that support servlets have as a helper app: *servlet container.*



The servlet container locates the servlet, creates **SERVLET** request and response objects and passes them to the servlet, and returns to the web server the response stream that the servlet produces.

Servlet Lifecycle



A Simple Order Form

Servlet Code doGet Example

```
response.setContentType("text/html");
 PrintWriter writer = response.getWriter();
 writer.println("<html>");
 writer.println("<head>");
 writer.println("<title>" + TITLE + "</title></head>");
 writer.println("<body><h1>" + TITLE + "</h1>");
 writer.println("<form method='post'>");
 writer.println("<table>");
 writer.println("<tr>");
 writer.println("<td>Name:</td>");
 writer.println("<td><input name='name'/></td>");
 writer.println("</tr>");
 writer.println("<tr>");
 writer.println("<td>Address:</td>");
 writer.println("<td><textarea name='address' "
         + "cols='40' rows='5'></textarea></td>");
 writer.println("</tr>");
```

Continued

```
    writer.println("<tr>");
       writer.println("<td>Country:</td>");
       writer.println("<td><select name='country'>");
       writer.println("<option>United States</option>");
       writer.println("<option>Canada</option>");
       writer.println("</select></td>");
       writer.println("</tr>");
       writer.println("<tr>");
       writer.println("<td>Delivery Method:</td>");
       writer.println("<td><input type='radio' " +
               "name='deliveryMethod'"
               + " value='First Class'/>First Class");
       writer.println("<input type='radio' " +
               "name='deliveryMethod' "
               + "value='Second Class'/>Second Class</td>");
       writer.println("</tr>");
       writer.println("<tr>");
       writer.println("<td>Shipping Instructions:</td>");
writer.println("<td><textarea name='instruction' "
               + "cols='40' rows='5'></textarea></td>");
       writer.println("</tr>");
```

And More

```
writer.println("<tr>");
        writer.println("<td> </td>");
        writer.println("<td><textarea name='instruction' "
                + "cols='40' rows='5'></textarea></td>");
        writer.println("</tr>");
        writer.println("<tr>");
        writer.println("<td>Please send me the latest " +
                "product catalog:</td>");
        writer.println("<td><input type='checkbox' " +
                "name='catalogRequest'/></td>");
        writer.println("</tr>");
        writer.println("<tr>");
        writer.println("<td> </td>");
        writer.println("<td><input type='reset'/>" +
                "<input type='submit'/></td>");
        writer.println("</tr>");
        writer.println("</table>");
        writer.println("</form>");
        writer.println("</body>");
        writer.println("</html>");
```

Main Point

Servlets are the basis of dynamic web applications. Servlets process information from a request object and generate new information in a response object. *For every action in nature there is always a reaction.*

Web Application Deployment Descriptor (web.xml)

- Defines everything about your Java Web Application that a server needs to know:
  - Servlets
  - Initialization parameters
  - Listeners
  - Filters
  - Error pages
  - Welcome pages

Three Names of a Servlet [web.xml]

```
<servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>mum.Hello</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

“special” URL Patterns

Default Servlet Mapping:

<url-pattern>/</url-pattern> matches a request if no other pattern matches.

<url-pattern>/*</url-pattern> overrides all other servlets. Whatever request you fire, it will end up in that servlet.

Error-Page (web.xml)

```
<error-page>
  <error-code>404</error-code>
  <location>/page-not-found.html</location>
</error-page>
```

Welcome List [web.xml]

- <welcome-file-list>
- <welcome-file>index.html</welcome-file>
- </welcome-file-list>

ServletConfig & ServletContext

- ServletConfig is a configuration object PER servlet to pass initialization information to a servlet during servlet init().
- ServletContext holds parameters that are initialization information for the entire application(i.e, every servlet in the application). ServletContext is also used during the application for application state management.

```
<web-app>
        <servlet>
                <servlet-name>order</servlet-name>
                <servletclass>mum.edu.cs.Order</servlet-class>
                <init-param>
                        <param-name>servletName</param-name>
                        <param-value>MVC2</param-value>
                </init-param>
        </servlet>
        <context-param>
                <param-name>applicationName</param-name>
                <param-value>Order Form</param-value>
        </context-param>
         <listener>
                 <listener-class>mum.edu.listener.OrderContextListener </listener-class>
         </listener>
</web-app>
```

OrderFormStartup Demo

Override Servlet init() – to process the Servlet init-param FROM web.xml:

```
<init-param>
        <param-name>servletName</param-name>
        <param-value>MVC2</param-value>
</init-param>
```

```
public class Order extends HttpServlet {
  // Store servlet init-param here
  String servletName;
  @Override
  public void init( ) throws ServletException {
      servletName =
  getServletConfig().getInitParameter("servletName");
}
```

OrderFormStartup Demo [Cont.]

Implement Listener to process Application Context context-param [web.xml]:

```
<context-param>
        <param-name>applicationName</param-name>
        <param-value>Order Form</param-value>
</context-param>

<listener>
        <listener-class>mum.edu.listener.OrderContextListener </listener-class>
 </listener>
```

```
public class OrderContextListener implements ServletContextListener{
   public void contextInitialized(ServletContextEvent event){
     ServletContext servletContext = event.getServletContext();
        String applicationName =
                        servletContext.getInitParameter("applicationName");
      servletContext.setAttribute("applicationName", applicationName);
   }
```

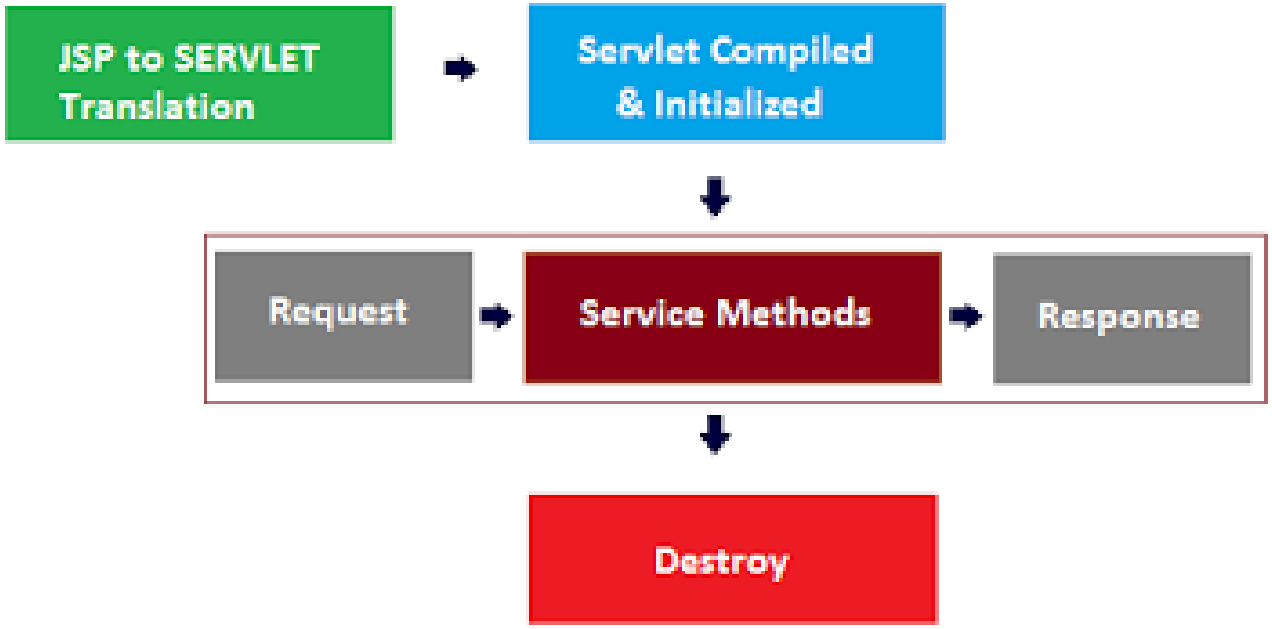
Main Point

- Web containers manage the life cycle of servlets. *The unified field manages the entire universe.*

Java Server Pages

- Architecturally, a Java Server Page is a high-level abstraction of Java servlets.
- It is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application.
- Developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

JSP Lifecycle



A Simple Order Form “DEMO”

```
public void _jspService(final javax.servlet.http.HttpServletRequest request, final
   javax.servlet.http.HttpServletResponse response)
 try {
   response.setContentType("text/html");
   pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
   _jspx_page_context = pageContext;
   application = pageContext.getServletContext();
   config = pageContext.getServletConfig();
   session = pageContext.getSession();
   out = pageContext.getOut();
   _jspx_out = out;
   out.write("<!DOCTYPE html>\r\n");
   out.write("<html>\r\n");
   out.write("<head>\r\n");
   out.write("<link rel=\"stylesheet\" type=\"text/css\" href=\"resources/mystyle.css\">\r\n");
   out.write("<meta charset=\"ISO-8859-1\">\r\n");
   out.write("<h4>Order</h4>\r\n");
   out.write("</head>\r\n");
   out.write("<body>\r\n");
   out.write("\t<form  method=\"post\">\r\n");
  …
```

JSP Intro Demo

SCRIPLETS [UGH!]

BETTER THAN SERVLETS!!!

Main Point

The web container generates a servlet from a JSP file the first time the JSP is requested from a web application. Since a JSP is essentially a servlet, one should understand servlets to effectively deal with JSPs. *Actions in accord with fundamental levels of knowledge insure success in dealing with more expressed values.*

Expression Language versus Scriptlet

- Example: We have a person object stored as a request attribute. A person has an address and an address has a zip. How to retrieve zip?

- Scripting approach:

```
<%((pkg.Person)request.getAttribute("person")).
getAddress().getZip() %>
```

- EL approach:

```
${person.address.zip}
```

"Scripting considered harmful"

- JSP scripting originated in early days of web apps
- JSPs try to avoid scripting
- Might see in legacy code
- Might see something similar in other frameworks
  - PHP
  - ASP.net
  - ASPMVC.net
  - …?
- Scripting is part of "Model 1 JSP architecture"

Scriptless JSPs

- The JSP Expression Language (EL)
  - Syntax: ${person.name}
  - Left value is either an attribute or an implicit object (like request), right side is a property
- Tag Libraries
  - There is a standard set of additional tags for JSP pages that encapsulate functionality of scriptlets (like for loops), thereby eliminating embedded code

JSP Expression Language [EL]

- An expression ${expr} prints expr to the page
- An expression ${person.name} evaluates (and prints to the page) something like: person.getName(), where person is an instance of a Person bean having a *name* variable.
- Has arithmetic and logical operators
- To test whether two objects are equal, EL uses **eq[or ==]** for equals and **ne[or !=]** for not equals
- Can use integers, floating point numbers, strings, the built-in constants true and false for boolean values, and null.

Main Point

An EL expression is a compact expression of a systematic evaluation of the page, request, session and application scopes. *The laws of nature are compact expressions that control the infinite diversity of the manifest creation*

JSTL – JSP Standard Tag Library

- Provides new tags for JSPs that reduce scripting
- They use the custom tag API, but have become a standard library, essentially a part of JSP language

Using JSTL

The JSTL library provides 5 Sets of tags, each having a different (standard) prefix. You "import" a library by placing a "taglib" directive at the top of your jsp page. Here are the choices:

- **Core tags: (can do if, if/else, loops…)**

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

```
NOTE: the taglib directive has no XML syntax analog
```

Additional Tags

- Function tags: (standard Java string manipulation)

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

- Format Tags (format numbers, dates..)

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

- SQL Tags (set data source, perform queries)

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

- XML Tags (for navigating/parsing/working with XML files)

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

Head First, p. 475 lists all the available JSTL library tags

C:OUT
Preventing XSS (Cross Site Scripting)

Encode "untrusted data"

- Replace HTML markup with alternate representations
- For example, <script> gets converted to <script>
- Sample Character representations:

Char	Description	Entity name	Entity number
	Space		
<	Less than	<	<
>	Greater than	>	>
&	Ampersand	&	&

- Open Web Application Security Project (OWASP)

C:URL provides for URL Encoding

- URLs can only be sent over the Internet using the ASCII character-set.
- Since URLs often contain characters outside the ASCII set [UTF-8/Unicode], the URL has to be converted into a valid ASCII format.
- URL encoding replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits.

- For example, URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign or with %20.
- Percent Encoding
- Unsafe Characters: rfc1738

JSTL DEMO

- DEMO: show how to use some of them…
  - c:set (set value of a variable)
  - c:out --- XSS character escaping
  - c:if
  - c:choose
  - c:forEach
  - c:url --- with URL encoding

- Many of these make simple use of the EL.

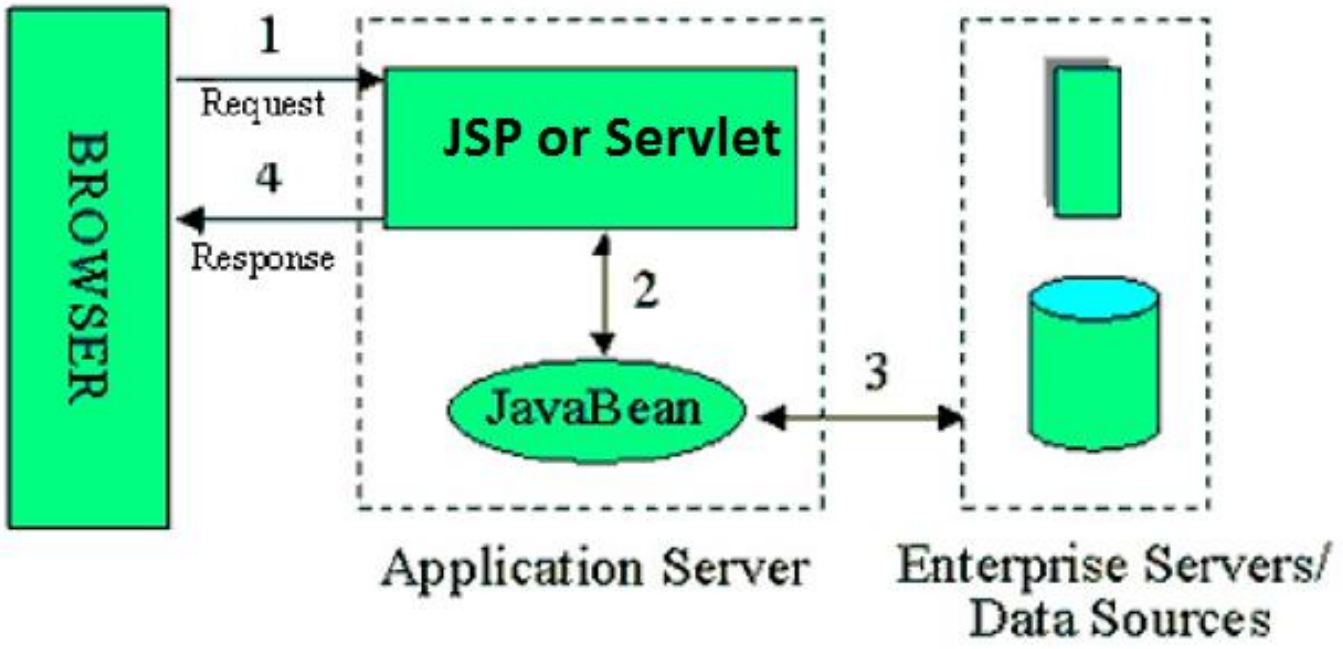
Main Point

- The JSP Standard Tag Library provides convenient action tags for many common operations on a JSP page. JSTL combined with the EL could satisfy JSP needs. *Using a JSP action is analogous to the efficient use of some law of nature.*

MVC: Model-View-Controller

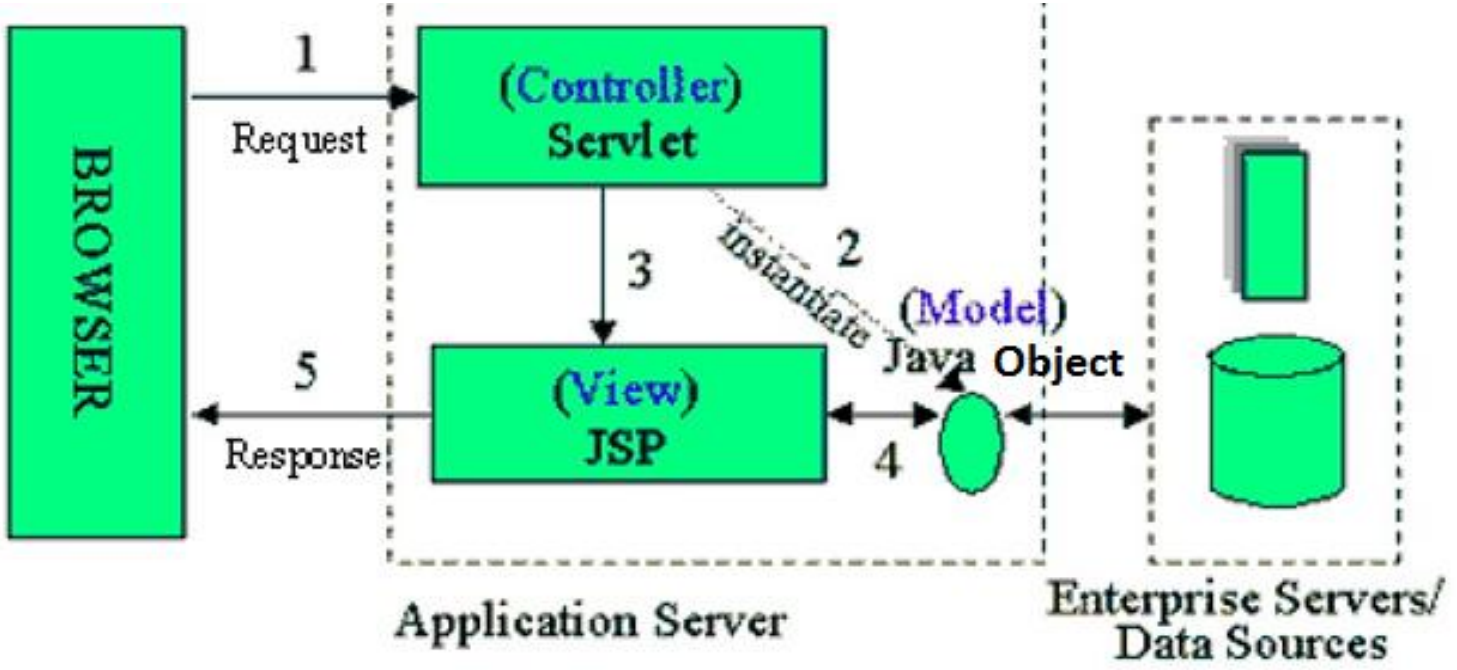
- **Separation of Concerns**
  - Separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes
- **Model**
  - Maintains the data[i.e. state] of the application domain
- **View**
  - Manages the display of information.
- **Controller**
  - Manages user input triggering the model and/or the view to change as appropriate.

Model 1 Architecture



Model 1 mixes view and business logic inside each handling servlet (or JSP). This makes it more difficult to change the view independently of that logic, and difficult to change business logic without changing the view.

JSP Model 2 Architecture



Model II cleanly separates business data and logic from the view, and the two are connected by way of a controller servlet. The model allows for multiple controllers/servlets, [e.g., one per GET/POST pair]. Typical MVC Framework implementations have just **one controller servlet** which centralizes common tasks.

DEMO

JSP Intro Demo :
MVC1 – MVC2 examples

Main Point

When you use JSP pages according to a Model 2 architecture, there is a servlet that acts as a controller (**process of knowing**) that sets attribute values based on computations and results from a business model (**knower**), then dispatches the request to the servlet generated by the JSP page (**known**). The JSP servlet then retrieves the attribute values and inserts them into the designated places in the HTML being sent to the browser.