Java SERVLET

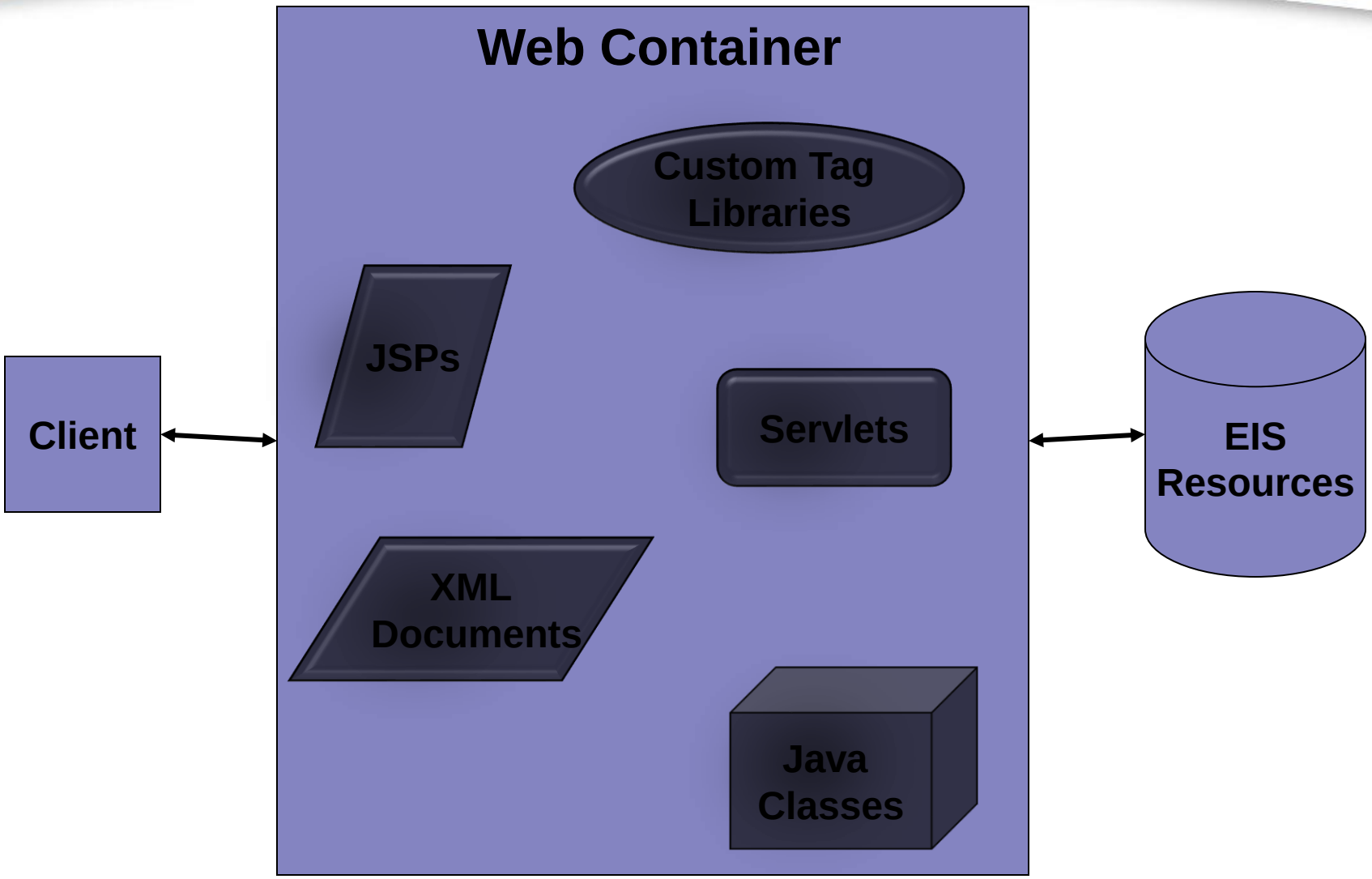
Unit I

Understanding Web Components

- When a web-based client such as a browser communicates with a J2EE application, it does so through server-side objects called web components.
- The web component should be those involved with receiving a request and creating a response to the client, whatever the client may be.
- For now we will focus on the components managed in web containers, starting with servlets then JSP.
- Servlets are Java programming language classes that dynamically process requests and construct responses.
- JSP pages are text-based documents that execute as servlets, but allow a more natural approach to creating static content.

Cont…

- So what is web containers?
  - Web components are supported by the services of a runtime platform called a web container.
  - A web container provides services such as request dispatching, security, concurrency, and life-cycle management.
  - It also gives web components access to APIs such as naming, transactions, and email.
  - Certain aspects of web application behavior can be configured when the application is installed, or deployed, to the web container.



Web Container Boundaries

Introduction of Servlet

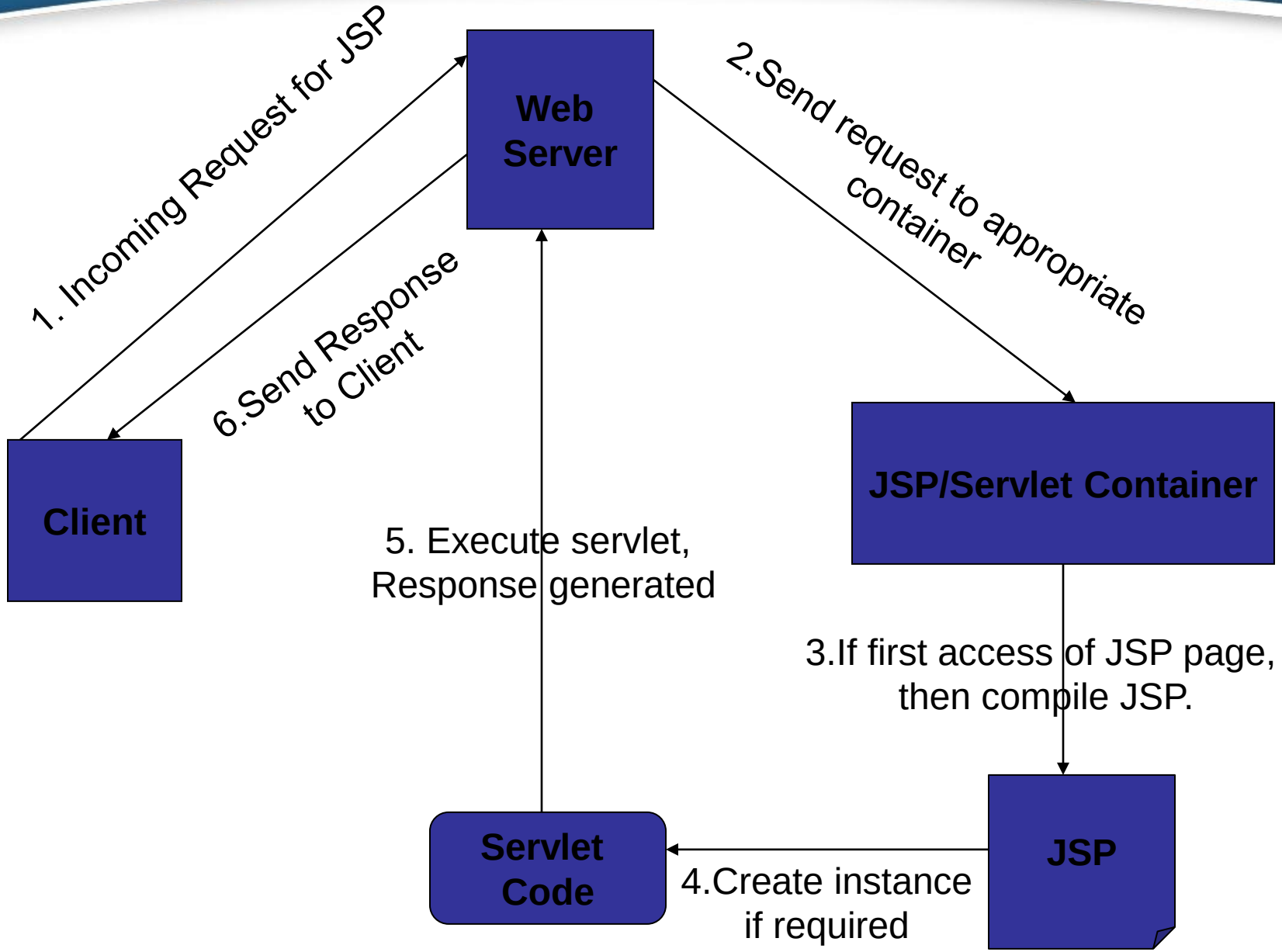
- They are programs that run on a Web server, acting as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Their job is to:
  - Read any data sent by the user
  - Look up any other information about the request that is embedded in the HTTP request
  - Generate the results
  - Format the results inside a document
  - Set the appropriate HTTP response parameters
  - Send the document back to the client

The Advantages of Servlets

- Servlet provides a programmatic way for creating dynamic and structured data.
- Servlets run inside a Java Virtual Machine (JVM) and are generic server extension that can be dynamically loaded when needed by the web server.
- Advantages of using Servlets:
- **Fast Performance:**
  - The performance of servlets is superior to CGI because there is no process creation for each client request. There is no process spawning with servlets so servlet invocation is very fast.
- **Scalable.**
  - Servlets are scalable as each request is handled by threads which gives better scalability.

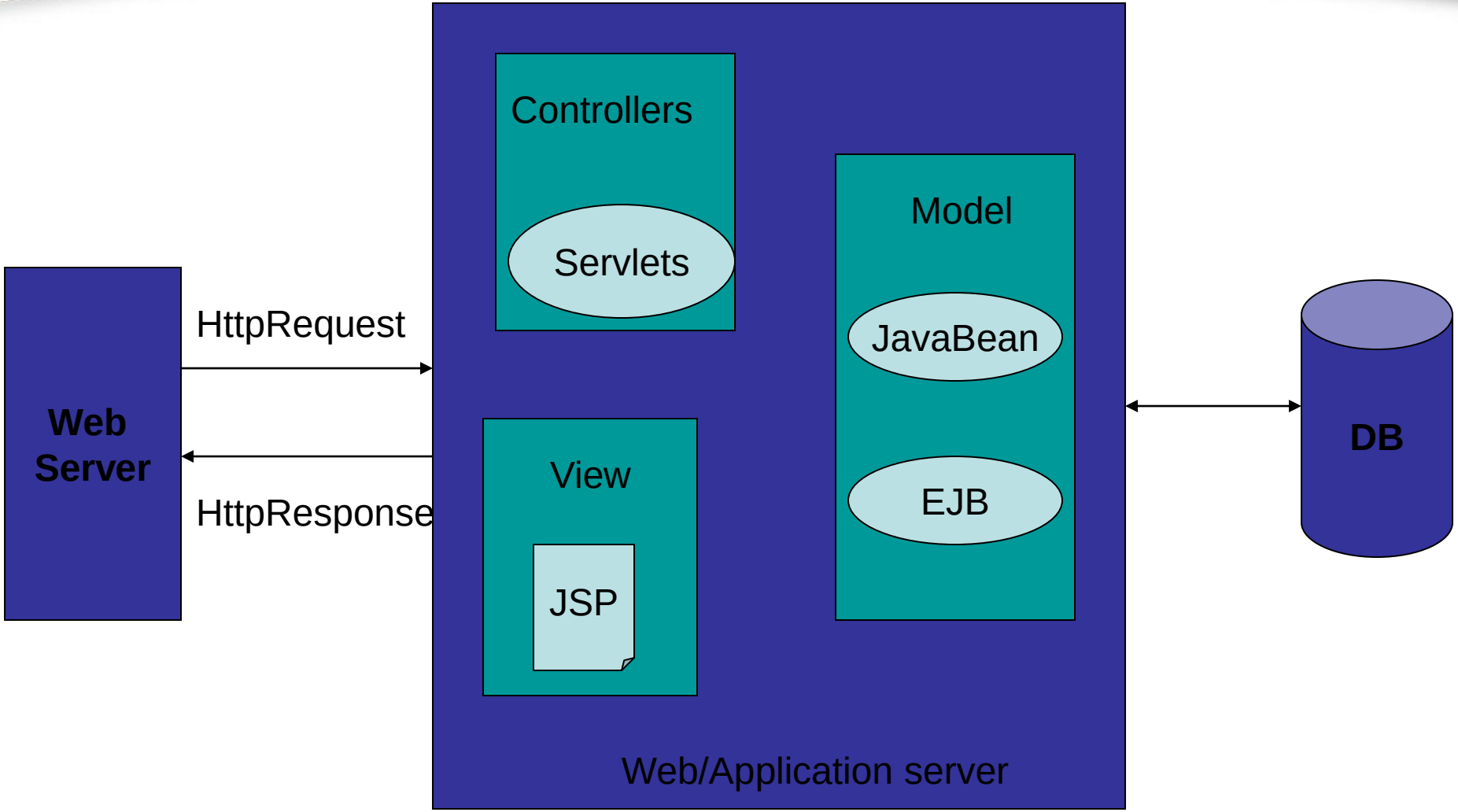
- **Portability.**
  - Being written in java servlets are portable between operating systems as well as Web Servers.
- **Rapid development cycle.**
  - Servlets technology is part of javax packages.
  - They are able to tale full advantages of all of the java API which helps speed up the development process.
- **Robustness.**
  - Servlets are managed by the Java Virtual Machine.
  - As such, you don't need to worry about memory leak or garbage collection, which helps you write robust applications.

- **Widespread acceptance.**
  - Java is a widely accepted technology. This means that numerous vendors work on Java-based technologies. One of the advantages of this widespread acceptance is that you can easily find and purchase components that suit your needs, which saves precious development time.
- Servlet will stay in memory one it’s loaded, which allows state management of other resources such as database connection.



Servlet Model

- A servlet is a Java class that can be loaded dynamically and run by a special web server.
- This servlet-aware web server is called a **servlet container**.
- Servlets interact with clients via a request-response model based on HTTP. Because servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client requests and server responses.
- In a JSP application, the servlet container is replaced by a JSP container. Both the servlet container and the JSP container often are referred to as the **web container or servlet/JSP container**, especially if a web application consists of both servlets and JSP pages.

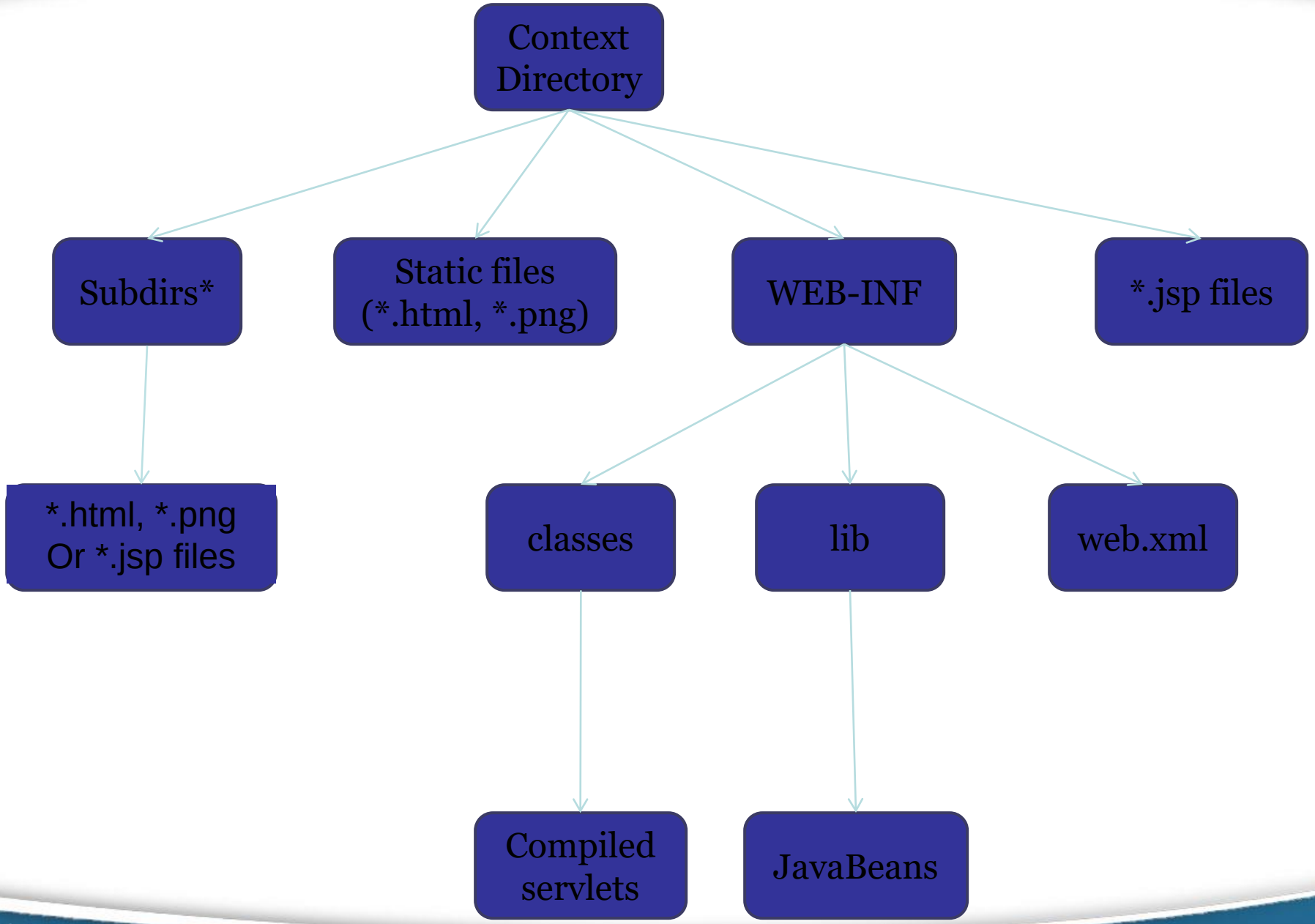


MVC Model

Web Application Structure

- Defined as a hierarchy of directories.
- The root of hierarchy is *document root.*
- The document root is a directory where the application context points.
- The **Context** element represents a *web application*, which is run within a particular virtual host.
- Anything located in document root becomes a public document accessible through the web application.
- For example, if index.html was in the document root of the j2ee context, it could be accessible through /j2ee/index.html.
- The context path for an application determines the URL for that application.

- There are rules applied for matching URL's to context paths, so you cant deploy two application with the same context.
- Otherwise, the web container would not be able to determine which application should actually receive the request.
- Usually web container will give error if you try to deploy an application that already has context defined or is a substring of an existing context.
- Since the document root is public you need some type of file protection, to restrict somebody to have access on your application's class file or other sensitive data.
- Figure shows the directory structure hierarchy.


Context Directory
Subdirs*
Static files (*.html, *.png)
WEB-INF
*.jsp files
*.html, *.png Or *.jsp files
classes
lib
web.xml
Compiled servlets
JavaBeans

Basic Servlet Structure

- To be a servlet, a class should extend HttpServlet and override doGet or doPost, depending on whether the data is being sent by GET or by POST.
- If you want the same servlet to handle both GET and POST and to take the same action for each, you can simply have doGet call doPost, or vice versa.
- Both of these methods take two arguments: an HttpServletRequest and an HttpServletResponse.

Listing 2.1 ServletTemplate.java

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

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

- The HttpServletRequest has methods by which you can find out about incoming information such as
  - form data,
  - HTTP request headers, and
  - the client's hostname.
- The HttpServletResponse lets you specify outgoing information such as
  - HTTP status codes (200, 404, etc.),
  - response headers (Content-Type, Set-Cookie, etc.), and
  - obtain a PrintWriter used to send the document content back to the client.

- Since doGet and doPost throw two exceptions, you are required to include them in the declaration.
- Finally, you have to import classes in java.io (for PrintWriter, etc.), javax.servlet (for HttpServlet, etc.), and javax.servlet.http (for HttpServletRequest and HttpServlet-Response).
- If your servlets generate HTML, set the HTTP Content-Type response header as response.setContentType("text/html");

  before actually returning any of the content via the Print-Writer.

Packaging Servlets

- To better manage servlets in production environment, servlets are placed inside a package.
- Insert a package statement in the class file as
- package coreservlets;

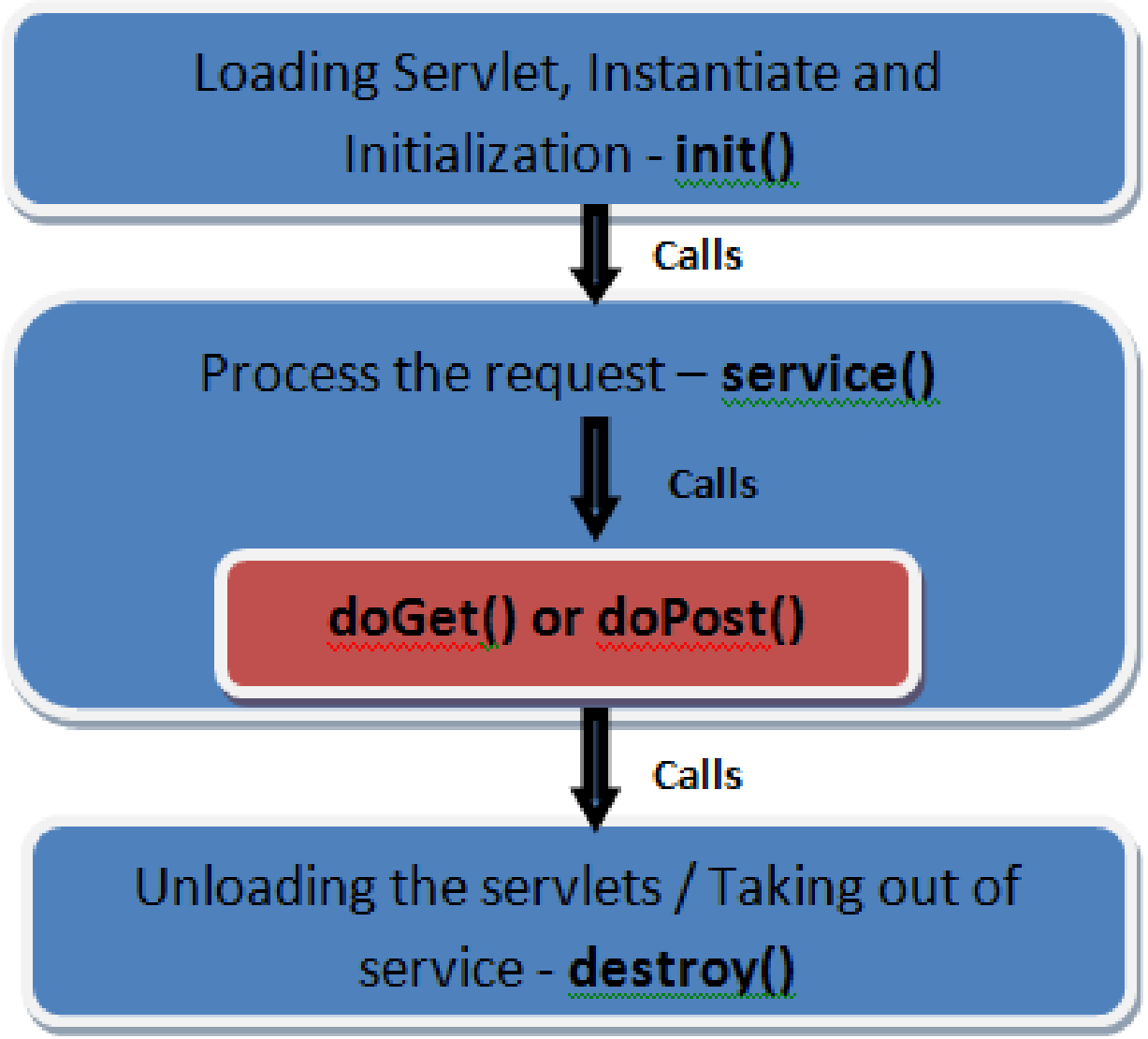
The Servlet Life Cycle

- When the servlet is first created, its init method is invoked, so that is where you put one-time setup code.
- After this, each user request results in a thread that calls the service method of the previously created instance.
- The service method then calls doGet, doPost, or another doXxx method, depending on the type of HTTP request it received.
- Finally, when the server decides to unload a servlet, it first calls the servlet's destroy method.

- ***The init Method:*** The init method is called **when the servlet is first created** and **is not called again for each user request**.
- The servlet can be created when a user first invokes a URL corresponding to the servlet or when the server is first started,
- There are two versions of init: one that takes no arguments and one that takes a ServletConfig object as an argument.

```
public void init() throws ServletException {
    // Initialization code...
}
public void init(ServletConfig config) throws ServletException {
      super.init(config);
      // Initialization code...
}
```

- ***The service Method:*** Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
- The service method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc., as appropriate.
- ***The destroy Method:*** The server may decide to remove a previously loaded servlet instance, perhaps because it is explicitly asked to do so by the server administrator, or perhaps because the servlet is idle for a long time. Before it does, however, it calls the servlet's destroy method.
- This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.


Loading Servlet, Instantiate and Initialization - init()
Calls
Process the request – service()
Calls
doGet() or doPost()
Calls
Unloading the servlets / Taking out of service - destroy()

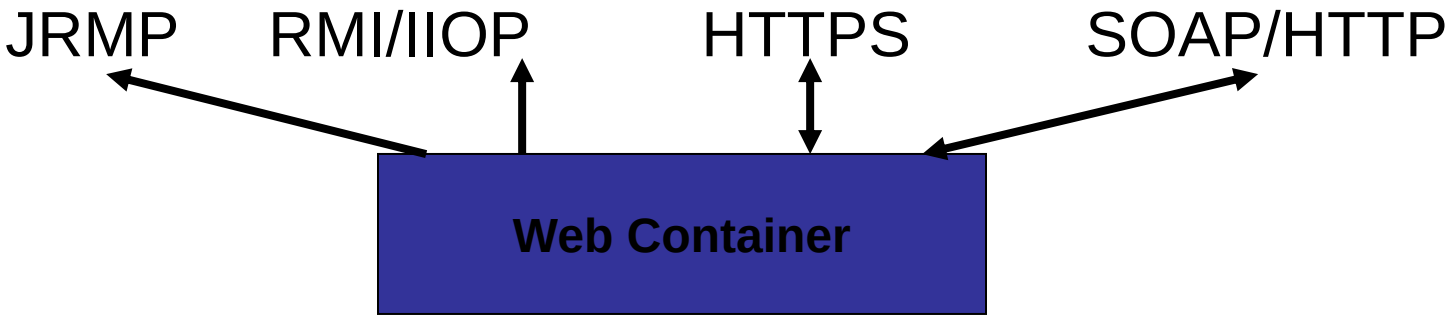
/WEB-INF/web.xml

- The web application deployment descriptor is the xml file describing the servlets, other components, and deployment information that make up the application.
- It describes the J2EE Web Application so that container knows how to manage application once it deployed.
- It is nothing more then a xml file which declaratively describes how to deploy various components that make up the web application.
- It allows you to change system settings without having to actually change the code.
- It is called as web.xml located at /WEB-INF directory of the web application. These are the type of information contained in the deployment descriptor.

- Servlet Context Initialization parameter
- Session Configuration
- Servlet Declaration
- Servlet Mapping
- Application lifecycle listener classs
- Filter definition and Filter mapping
- MIME type mapping
- Welcome file list
- Error Pages
- Security information
- Declaration of custom tag libraries
- Syntax for JNDI object.

Understanding Protocols

- Most standard web application uses browsers as their client.
- They use HTTP, HTTP Proxies, HTTPS (HTTP over SSL Secure Socket Layer).
- There are another protocols also which are supported by your J2EE application for different component.
- Some J2EE applications servers may provide support for tunneling, means one protocol is contained within another.



- **HTTP:** HTTP is standard protocol used on the web.
  - A client usually a web browser, sends a request to a web server.The web server receives the information and initiates specific processing on the server.
  - There are eight different request methods:
    - GET
    - POST
    - HEAD
    - OPTIONS
    - PUT
    - TRACE
    - DELETE
    - CONNECT
  - From these eight only GET and POST are commonly used.

- **GET Request Method :**
  - It is the most common request.
  - It's used to access resources such as documents, images or result sets.
  - It is used to retrieve dynamic information.
  - It can be retrieved by using query parameters that are encoded in the request URL.
  - When passing parameters in a URL, the resulting string referred as query string.
  - A request parameter is identified by using a ?, followed by a parameter name and its value.
  - Parameters are separated by the & symbol.
  - The sample URL can be:
    http://www.apress.com/j2ee?student=Liz&level=high

- The web server pare this string and get necessary information to generate response.
- Most browsers have limits on charactes in GET request string.
- Which is about 240 characters.
- GET request uses URL and query string so allows to bookmark the address in browser and allows to cut-paste the URL and use.
- If your application is of kind that it can create damage or wrong o/p by using same URL again, so do not use GET request.
- For example, if you are updating a database or processing a credit card order.

- **POST Request Method :**
  - We use POST requests for sending large amounts of data to the server.
  - Using the POST request it allows your data to be passed to server as a part of the HTTP request body, regardless of how much data exist.
  - A POST request doesn’t change URL at all.
  - The client web browser does not display anything different to the user or present a query string.
  - As a result, it can not be bookmarked or reloaded.
  - If you are using POST request, the information you are dealing with is supposed to be sent to the server only once.
  - A example of a POST request is uploading a file to a web server.

- **GET and POST in HTML Form Processing:**
  - In HTML form processing, it became common practice to use POST for request URLs that got to be too long.
  - The long URLs could not be handled by the limitations on the GET request size.
  - GET gives no protection against causing the change on a server.
  - With METHOD="GET", the form data is encoded into a URL.
  - This means that you can achieve a similar functionality of form submission by entering URL with appropriate parameters in the browser.
  - The browser divides the URL into the parts and recognize the host and then it sends to that host an HTTP GET request with the rest of the URL as an argument. Then server will process it.

- When using a method =“POST”, the form data is encoded using multipart/form data, and there for it’s not visible in a location line of a browser.
- It causes an HTTP POST request to be sent with the data encoded accordingly.

- **Other Request Methods:**
  - The other request methods are not used frequently. They are:
  - **HEAD:**
    - It is sent from the client when it want to see only the headers of a response.
    - It can use to determine the size or a type of a document.
    - Its used as a READ-ONLY request. No document body is returned.
  - **OPTIONS:**
    - You can use the OPTIONS request to find out which HTTP request methods the server supports.
  - **PUT:**
    - It can be used to store a resources on a server under a given URL that may not already exist.
    - This differs from using POST in that when using POST, the target resource already exist.

- **TRACE:**
  - It allows client to see what's being received at the other end of the request chain and use that data for testing or diagnostic information.
- **DELETE:**
  - It is used for removing information corresponding to a given URL.
  - After a successful DELETE method, the URL becomes invalid for any future requests.
- **CONNECT:**
  - It is reserved for use with a proxy that can be dynamically switch to being a tunnel.

- **Http Response:**
  - HTTP is request/response protocol.
  - For each request there is a response.
  - In each response there is a status code and other information contained in the response headers.
  - Status codes are three digit numbers, where the first digit defines the general classification of response.
    - 1xx: Informational. Request received, continuing process
    - 2xx: Success.
    - 3xx: Redirection. Further action must be taken to complete the request.
    - 4xx: Client Error. The request contained bad syntax or can’t be fulfilled.
    - 5xx: Server Error. The server failed to fulfill the request.

Reading Form Data from Servlets

- Call the getParameter method of the Http-ServletRequest, supplying the case-sensitive parameter name as an argument.
- You use getParameter exactly the same way when the data is sent by GET as you do when it is sent by POST.
- An empty String is returned if the parameter exists but has no value, and null is returned if there was no such parameter.
- Parameter names are case sensitive

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

public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Listing 3.2 ThreeParamsForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>

<FORM ACTION="/servlet/coreservlets.ThreeParams">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY>
</HTML>
```

Reading All Parameters

- The servlet looks up all the parameter names by the getParameterNames method of HttpServletRequest.
- This method returns an Enumeration that contains the parameter names in an unspecified order.
- Next, the servlet loops down the Enumeration in the standard manner, using hasMoreElements to determine when to stop and using nextElement to get each entry.
- Since nextElement returns an Object, the servlet casts the result to a String and passes that to getParameterValues, yielding an array of strings.
- If that array is one entry long and contains only an empty string, then the parameter had no values and the servlet generates an italicized “No Value” entry.
- If the array is more than one entry long, then the parameter had multiple values and the values are displayed in a bulleted list.

```
Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements()) {
    String paramName = (String)paramNames.nextElement();
    out.print("<TR><TD>" + paramName + "\n<TD>");
    String[] paramValues =
                        request.getParameterValues(paramName);
}
```

Handling the Client Request: HTTP Request Headers

- Methods of these interface help to know about a request.
- The getHeader method of HttpServletRequest, returns a String if the specified header was supplied on this request, null otherwise.
- Header names are not case sensitive.
- Request header methods:
- **getCookies**
  - The getCookies method returns the contents of the Cookie header, parsed and stored in an array of Cookie objects. This method is discussed more in next chapters.
- **getAuthType and getRemoteUser**
  - The getAuthType and getRemoteUser methods break the Authorization header into its component pieces.

- **getContentLength**
  - The getContentLength method returns the value of the Content-Length header (as an int).
- **getContentType**
  - The getContentType method returns the value of the Content-Type header (as a String).
- **getDateHeader and getIntHeader**
  - The getDateHeader and getIntHeader methods read the specified header and then convert them to Date and int values, respectively.
- **getHeaderNames**
  - Rather than looking up one particular header, you can use the getHeaderNames method to get an Enumeration of all header names received on this particular request.

- **getMethod**
  - The getMethod method returns the main request method (normally GET or POST, but things like HEAD, PUT, and DELETE are possible).
- **getRequestURI**
  - The getRequestURI method returns the part of the URL that comes after the host and port but before the form data.
- **getProtocol**
  - Lastly, the getProtocol method returns the third part of the request line, which is generally HTTP/1.0 or HTTP/1.1.

ShowRequestHeaders.java

```
Out.println(“<B>Request Method: </B>" + request.getMethod() );
Out.println(“"<B>Request URI: </B>" +request.getRequestURI();
Out.println(“ "<B>Request Protocol: </B>" + request.getProtocol());
Enumeration headerNames = request.getHeaderNames();
while(headerNames.hasMoreElements()) {
    String headerName = (String)headerNames.nextElement();
    out.println( headerName);
    out.println(request.getHeader(headerName));
}
```

Server Response: HTTP Status Codes

- When a Web server responds to a request from a browser or other Web client, the response typically consists of a status line, some response headers, a blank line, and the document. Here is a minimal example: HTTP/1.1 200 OK

  Content-Type: text/plain

  Hello World
- The status line consists of
  - the HTTP version (HTTP/1.1 in the example above),
  - a status code (an integer; 200 in the above example), and
  - a very short message corresponding to the status code (OK in the example).

- Servlets can perform a variety of important tasks by manipulating the status line and the response headers.
- For example, they can forward the user to other sites; indicate that the attached document is an image, Adobe Acrobat file, or HTML file; tell the user that a password is required to access the document; and so forth.
- Since the message is directly associated with the status code and the HTTP version is determined by the server, all a servlet needs to do is to set the status code.
- The way to do this is by the setStatus method of HttpServletResponse.

- If your response includes a special status code and a document, be sure to call setStatus before actually returning any of the content via the PrintWriter.
- That's because an HTTP response consists of the status line, one or more headers, a blank line, and the actual document, in that order.
- The setStatus method takes an int (the status code) as an argument, but instead of using explicit numbers, it is clearer and more reliable to use the constants defined in HttpServletResponse.
- The name of each constant is derived from the standard HTTP 1.1 message for each constant, all uppercase with a prefix of SC (for Status Code) and spaces changed to underscores. Thus, since the message for 404 is "Not Found," the equivalent constant in HttpServletResponse is SC_NOT_FOUND.

- There are two common cases where a shortcut method in HttpServletResponse is provided.
- public void sendError(int code, String message)
  - The sendError method sends a status code (usually 404) along with a short message that is automatically formatted inside an HTML document and sent to the client.
- public void sendRedirect(String url)
  - The sendRedirect method generates a 302 response along with a Location header giving the URL of the new document

Status Codes

- 100-199 Codes in the 100s are informational, indicating that the client should respond with some other action.
- 200-299 Values in the 200s signify that the request was successful.
- 300-399 Values in the 300s are used for files that have moved and usually include a Location header indicating the new address.
- 400-499 Values in the 400s indicate an error by the client.
- 500-599 Codes in the 500s signify an error by the server.

Response Headers

- A response from a Web server normally consists of a status line, one or more response headers, a blank line, and the document.
- To get the most out of your servlets, you need to know how to use the status line and response headers effectively, not just how to generate the document.
- Setting the HTTP response headers often goes hand in hand with setting the status codes in the status line.
- For example, all the “document moved” status codes (300 through 307) have an accompanying Location header, and a 401 (Unauthorized) code always includes an accompanying WWW-Authenticate header.

- Response headers can be used to
  - specify cookies,
  - to supply the page modification date (for client-side caching),
  - to instruct the browser to reload the page after a designated interval,
  - to give the file size so that persistent HTTP connections can be used,
  - to designate the type of document being generated, and to perform many other tasks

Setting Response Headers from Servlets

- The most general way to specify headers is to use the setHeader method of HttpServletResponse.
- This method takes two strings:
  - the header name and
  - the header value.
- As with setting status codes, you must specify headers before returning the actual document.
- HttpServletResponse also has two specialized methods to set headers that contain dates and integers:

- setDateHeader(String header, long milliseconds) This method saves you the trouble of translating a Java date in milliseconds since 1970 (as returned by System.currentTimeMillis, Date.getTime, or Calendar.getTimeInMillis) into a GMT time string.
- setIntHeader(String header, int headerValue) This method spares you the minor inconvenience of converting an int to a String before inserting it into a header.
- With servlets version 2.1, setHeader, setDateHeader and setIntHeader always add new headers, so there is no way to “unset” headers that were set earlier
- With servlets version 2.2, setHeader, setDateHeader, and setIntHeader replace any existing headers of the same name, whereas addHeader, addDateHeader, and addIntHeader add a header regardless of whether a header of that name already exists.

- HttpServletResponse also supplies a number of convenience methods for specifying common headers. These methods are summarized as follows.
- setContentType
  - This method sets the Content-Type header and is used by the majority of servlets
- setContentLength
  - This method sets the Content-Length header, which is useful if the browser supports persistent (keep-alive) HTTP connections

- addCookie
  - This method inserts a cookie into the Set-Cookie header. There is no corresponding setCookie method, since it is normal to have multiple Set-Cookie lines.
- sendRedirect
  - the sendRedirect method sets the Location header as well as setting the status code to 302.

Content-Type

- The Content-Type header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
- Setting this header is so common that there is a special method in HttpServletResponse for it: setContentType.
- MIME types are of the form maintype/subtype for officially registered types, and of the form maintype/x-subtype for unregistered types.
- The default MIME type for servlets is text/plain, but servlets usually explicitly specify text/html.
- Common MIME Types:

Cookies Handling

- Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.
- There are three steps involved in identifying returning users:
  - Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
  - Browser stores this information on local machine for future use.
  - When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

- Cookies are usually set in an HTTP header
- Servlet Cookies Methods:
- **public void setMaxAge(int expiry)**
  - This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
- **public int getMaxAge()**
  - This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
- **public String getName()**
  - This method returns the name of the cookie. The name cannot be changed after creation.

- **public void setValue(String newValue)**
  - This method sets the value associated with the cookie.
- **public String getValue()**
  - This method gets the value associated with the cookie.
- **public void setComment(String purpose)**
  - This method specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user.
- **public String getComment()**
  - This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

- Setting Cookies with Servlet:

(1) Creating a Cookie object: You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

- Cookie cookie = new Cookie("key","value");
- Keep in mind, neither the name nor the value should contain white space or any of the following characters:
- [ ] ( ) = , " / ? @ : ;

(2) Setting the maximum age: You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

- cookie.setMaxAge(60*60*24);
- **(3) Sending the Cookie into the HTTP response headers:** You use**response.addCookie** to add cookies in the HTTP response header as follows:
- response.addCookie(cookie);