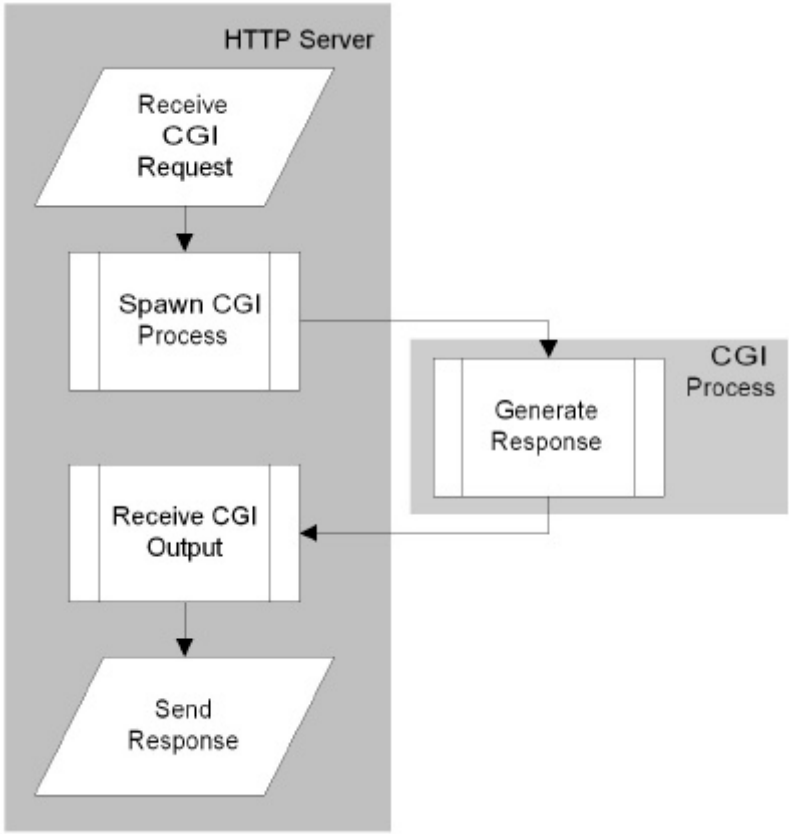
Server-Side Aspects

- **Common Gateway Interface (CGI)**
  - First standard for generating web content (responses) dynamically
  - Mechanism to pass request information to external programs, which were then run by the web server to generate responses at runtime
  - Program can be written in any language
  - Popular programming languages are scripting languages like Perl, Tcl, and Python



Server-Side Aspects (cont`d)

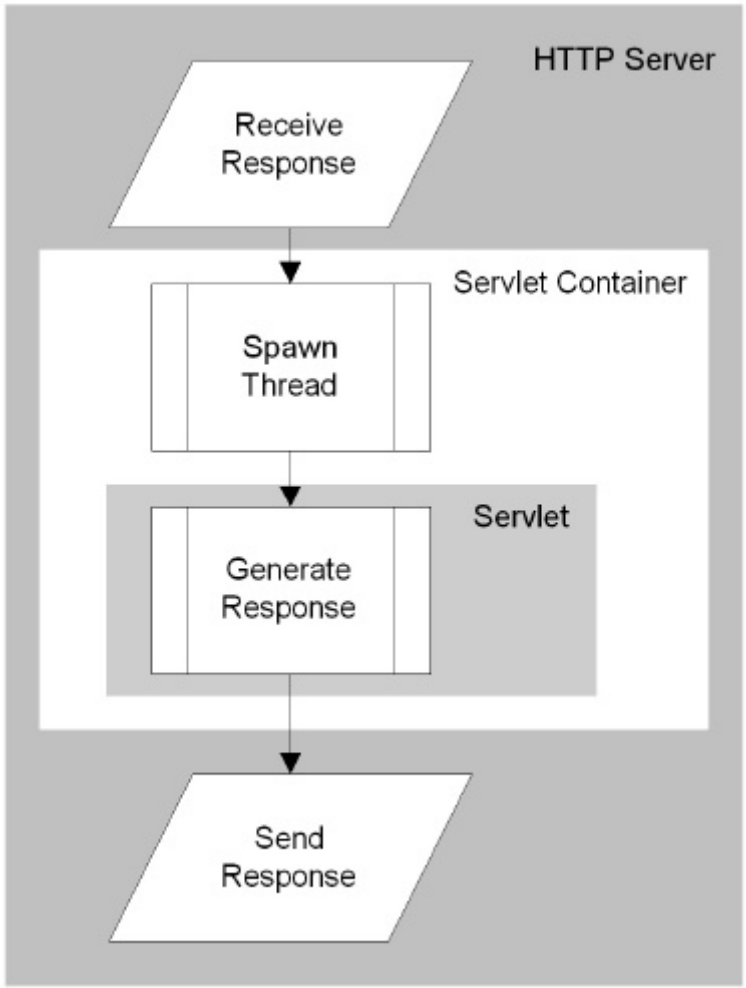
- Example

```
<HTML>
CGI to get current time on server.
<FORM METHOD="POST"
ACTION=http://at.home.at/scripts/getTime>
<INPUT type="submit" value="Get time">
</FORM>
</HTML>
```

```
main(int argc, char *argv[]) {
   printf("Content-type: text/html%c%c",10,10);
   printf ("23:59:59");
}
```

Server-Side Aspects (cont`d)

- **Servlets**
  - Small Java-based applications
  - Programming model is similar to CGI scripts
    - Mapping HTTP requests into HTTP responses
  - All of the servlets associated with a Web server run inside a single process



HyperText Transfer Protocol

- **HTTP defines how a Web browser and a Web server communicate**
- **Uses a request/response model**
- **Structure of a request**

```
GET /index.html HTTP/1.1
```

  - `GET`
    - Is the *method* that the Web server should perform
    - The GET method instructs the web server to get a web page from its storage and send it back to the client
  - `/index.html`
    - The *resource* the the web server should act on
  - `HTTP/1.1`
    - Version of the HTTP that the client supports

HyperText Transfer Protocol (cont`d)

- The remainder of the request is composed of various *HTTP headers*, which contain miscellaneous info about the client

- **Structure of a response**

```
HTTP/1.1 200 OK
Date: Sat, 13 Oct 2001 10:45:30 GMT
Connection: close
Content-Type: text/html

<html>
...
```

- First line is status line
  - `HTTP/1.1`
    - Version of HTTP that the server is using
  - `200`
    - Status code.

HyperText Transfer Protocol (cont`d)

- `OK`
  - *Reason phase* provides a brief textual explanation of the status code
    - 200 OK
    - 404 Not Found
    - 500 Internal Server Error
    - ...
- The remainder of the request is composed of various *HTTP headers*, and the *message body*, which contains the requested resource

- **Usually, a Web browser sends several requests to a Web browser in order a single HTML document**
  - It‘s not unusual for a Web page to require 25 or more requests

Server-Side Aspects (cont`d)

- Advantages
  - Benefits of the core Java platform
    - OOP, garbage collection, cross-platform portability, rich collection of Java API for accessing databases, directory servers, network resources, etc.
- Disadvantages
  - Both static and dynamic document contents reside in program source code
  - Changes requires intervention by a programmer
  - Maintainance gets difficult, because of mixing scripting and HTML
- Alternative programming model would use an object-oriented approach
  - Modeling response data by constructing a collection of Java objects
    - e.g. as an hierarchy of textual elements, including a title,various levels of headings, paragraphs, etc.
  - Popular Open Source library is Element Construction Set (ECS)
    - Supports output in HTML and XML
    - see The Jakarta Project (jakarta.apache.org)

Servlets Characteristics (cont`d)

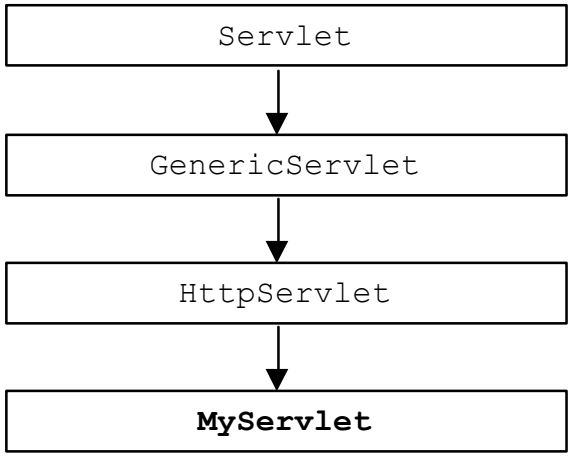
- Servlets are persistent
  - Unlike CGI scripts, a Servlet`s lifecycle extends beyond HTTP each request
  - Servlet container manages the lifecycle of the Servlet and handles the socket-level communication
- **Uses for Servlets**
  - Forwarding requests
    - A servlets can forward requests to other servers and servlets
    - Thus servlets can be used to balance load among several servers, and to partition a single logical service over serveral servers
  - Collaboration between people
    - A servlet can handle multiple requests concurrently, and can synchronize requests
    - This allows servlets to support systems such as on-line conferencing
- **Useful for implementing a three-tier architecture**

Servlet Overview

- **Servlet Package (`javax.servlet`)**
  - Interfaces
    - `RequestDispatcher, Servlet, ServletConfig, ServletContext, ServletRequest, ServletResponse, SingleThreadModel`
  - Classes
    - `GenericServlet, ServletInputStream, ServletOutputStream, ServletException, UnavailableException`
- **Servlet Package (`javax.servlet.http`)**
  - Interfaces
    - `HttpServletRequest, HttpServletResponse, HttpSession, HttpSessionBindingListener, HttpSessionContext`
  - Classes
    - `Cookie, HttpServlet, HttpSessionBindingEvent, HttpUtils`

Servlet Overview (cont`d)

- **The `Servlet` Interface**
  - Implemented by all servlets
  - Defines a protocol (abstract methods) that a servlet subclass must implement
- **A `HttpServlet` class provides HTTP specific methods**



Interface that defines `init, destroy, service getServletConfig, getServletInfo` methods

Class that implements `Servlet` in a protocol independent way and adds a variety of methods useful to all Servlets

Class that extends `GenericServlet` for HTTP with `doDelete, doGet, doPost, doPut, service` methods

Servlet Overview (cont`d)

- **`HttpServlet` class Overview**

Category	Method	Comments	Usually overridden
Lifecycle Methods	`init, destroy`	Startup and shutdown	yes
Get and Post Methods	`doGet, doPost`	Main service methods used	yes
WebDAV Methods	`doDelete, doPut`	For WebDAV	yes
Misc	`service, doTrace, log`	`service` method is for dispatching the HTTP; e.g. `doGet`, `doPost`, etc. methods	no
Environment	`getInitParameter, getServletConfig`		no

  - WebDAV stands for *Web-based Distributed Authoring and Versioning*
    - It is a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on remote web servers. (see www.webdav.org)

Servlet Overview (cont`d)

- **Input and Output**
  - Defined by two interfaces
  - `ServletRequest`
    - Encapsulates communication from client to server
      - Request parameters, protocol used, client and server hostnames, etc.
    - Request data can be retrieved via the `ServletInputStream` interface
  - `ServletResponse`
    - Encapsulates communication from server to client
      - Setting the content type (MIME), length, etc.
    - Servlet composes the reply through the `ServletOutputStream` interface

Servlet Overview (cont`d)

- **A simple, but complete Servlet code example**

```
public class SimpleServlet extends HttpServlet {
/* Handle the HTTP GET method by building a simple web page. */
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
   throws ServletException, IOException {
     PrintWriter out;
     String title = "Simple Servlet Output";
     // set content type and other response header fields first
     response.setContentType("text/html");
     // then write the data of the response
     out = response.getWriter();
     out.println("<HTML><HEAD><TITLE>");
     out.println(title);
     out.println("</TITLE></HEAD><BODY>");
     out.println("<H1>" + title + "</H1>");
     out.println("<P> This is output from SimpleServlet.");
     out.println("</BODY></HTML>");
     out.close();
   }
}
```

Servlet Overview (cont`d)

- **Calling Servlets from a Browsers**

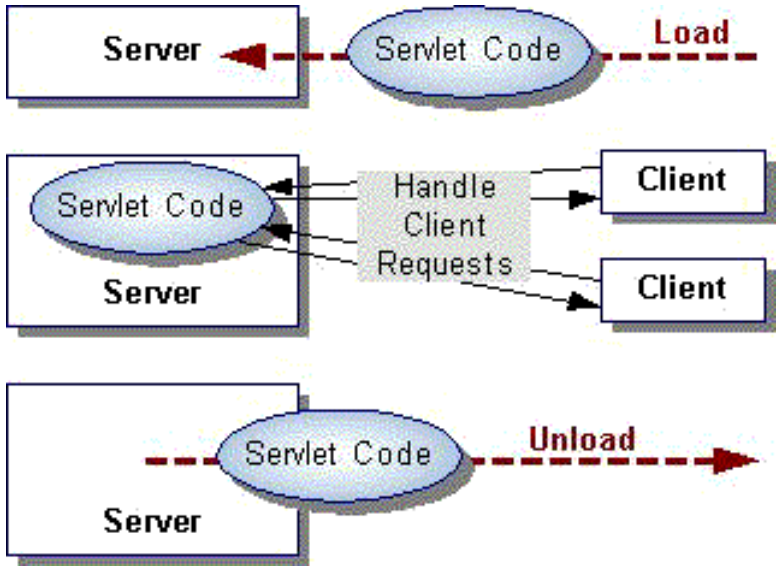
```
(Tomcat) http://machine-name:port/Context-root/Servlet-name
(JSDK) http://machine-name:port/servlet/Servlet-name
```

  - `Context-root` corresponds to the subdirectory of `TOMCAT_HOME/webapps` installation directory
  - `Servlet-name` corresponds to the name of the Servlet
  - Example:

```
(Tomcat) http://localhost:8080/bookstore/bookdetails?bookId=203
(JSDK) http://localhost:8080/servlet/bookdetails?bookId=203
```

Servlet Life Cycle

- **Each servlet has the same life cycle:**
  - A server loads and initializes the servlet
  - Servlet handles zero or more client requests
  - Server removes the servlet
    - Some servers do this step only when they shut down



Servlet Life Cycle (cont`d)

- **Initializing a Servlet**
  - Server calls the `init` method once, and
  - Server will not call the `init` method again unless the server is reloading the servlet
  - Server can not reload a servlet until the server has destroyed the servlet by running the `destroy` method
  - A user has to override `init` method to provide a specific implementation
- **Rules for overriding `init`**
  - If an initialization error occurs, throw an `UnavailableException`
    - e.g. if a required network connection cannot be established
  - Do not call the `System.exit` method

Servlet Life Cycle (cont`d)

- **Destroying a Servlet**
  - A user has to override `destroy` method to provide a specific implementation
  - The `destroy` method should undo any initialization work and synchronize persistent state with the current in-memory state
  - A server calls the `destroy` method after all service calls have been completed, or a server-specific number of seconds have passed, whichever comes first
    - If a servlet handles any long-running operations, service methods might still be running when the server calls the `destroy` method
    - Developer is responsible for making sure those threads complete

Web Application

- **A web application is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors.**
- **A web application may consist of the following items:**
  - Servlets
  - JavaServer Pages
  - Utility Classes
  - Static documents (html, images, sounds, etc.)
  - Client side applets, beans, and classes
  - Descriptive meta information which ties all of the above elements together
- **By default an instance of a web application must only be run on one VM at any one time**
  - This behavior can be overridden if the application is marked as “distributable” via its deployment descriptor

Web Application (cont`d)

- **A web application exists as a structured hierarchy of directories**
- **The root of this hierarchy serves as a document root for serving files that are part of this context**
  - e.g. a web application located at `/catalog` in a web server, the `index.html` file located at the base of the web application hierarchy can be served to satisfy a request to `/catalog/index.html`
- **A special directory exists within the application hierarchy named `WEB-INF`**
  - This directory contains all things related to the application that aren't in the document root of the application
  - The `WEB-INF` node is not part of the public document tree of the application
  - No file contained in the `WEB-INF` directory may be served directly to a client

Web Application (cont`d)

- **The contents of the `WEB-INF` directory are:**
  - `/WEB-INF/web.xml`
    - Deployment descriptor
  - `/WEB-INF/classes/*`
    - Directory for servlet and utility classes. The classes in this directory are used by the application class loader to load classes from.
  - `/WEB-INF/lib/*.jar`
    - Area for Java ARchive files which contain servlets, beans, and other utility classes useful to the web application
    - All such archive files are used by the web application class loader to load classes from
  - `/WEB-INF/tlds/*.tld`
    - Area for JSP type library definitions (TLDs)
    - Not specificied in JSP specification, but widely used for convenience

Web Application (cont`d)

- **Sample directory structure**

```
/index.html
/howto.jsp
/feedback.jsp
/images/banner.gif
/images/jumping.gif
/WEB-INF/web.xml
/WEB-INF/lib/jspbean.jar
/WEB-INF/classes/com/mycorp/servlets/MyServlet.class
/WEB-INF/classes/com/mycorp/util/MyUtils.class
```