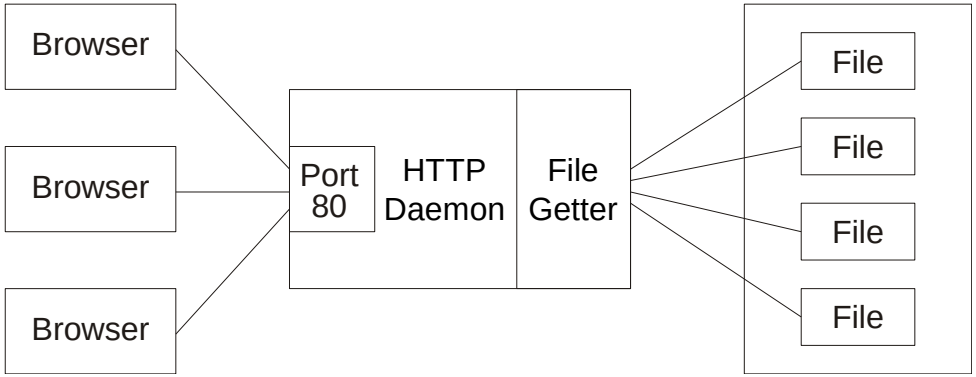
Network Computing Lecture 13

Professor Louise E. Moser

Spring 2012

Java Servlets and Web Servers

- **Java Servlet** is to **Web Server** as **Java Applet** is to **Web Browser**
- **HTTP Web Servers**
  - HTTP daemon listens for clients' requests on the well-known HTTP port 80
  - HTTP requests are typically of the form GET filename
  - The Web Server searches its document tree for the requested document and returns it to the requesting client
  - Each client request gets one thing from the Web Server
  - Several requests might be necessary to get everything (text, formatting and layout instructions, graphics) that is needed to present a Web page to the client



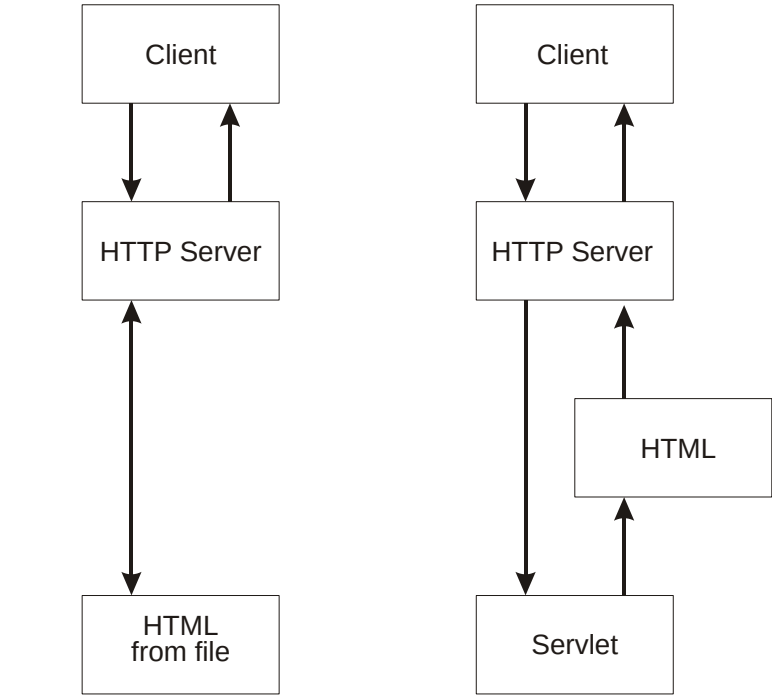
Java Servlets

- A Java Servlet is a server-side application that **dynamically** produces an HTML document; it performs database queries and integrates the results from them into the document
  - Replaces the Common Gateway Interface (CGI)
  - CGI defines environment variables using information in Request and Response message headers
  - CGI is platform-dependent, slow, not scalable
- A Servlet enables you to create **executable** or **dynamic** (rather than static) content for Web pages
- A Servlet has no defined interfaces
  - Web Server maps a client's request for a document into a Servlet call
  - Servlet creates and returns a document corresponding to the client's request
- A Servlet can be initialized, invoked, and destroyed

Conventional vs. Servlet HTTP Request

- A Servlet creates a document on-the-fly from
  - Data submitted by the user (e.g., online store, search engine), or
  - Data that change frequently (e.g., news, sports, stock information)

  rather than simply by getting a document that has been created previously

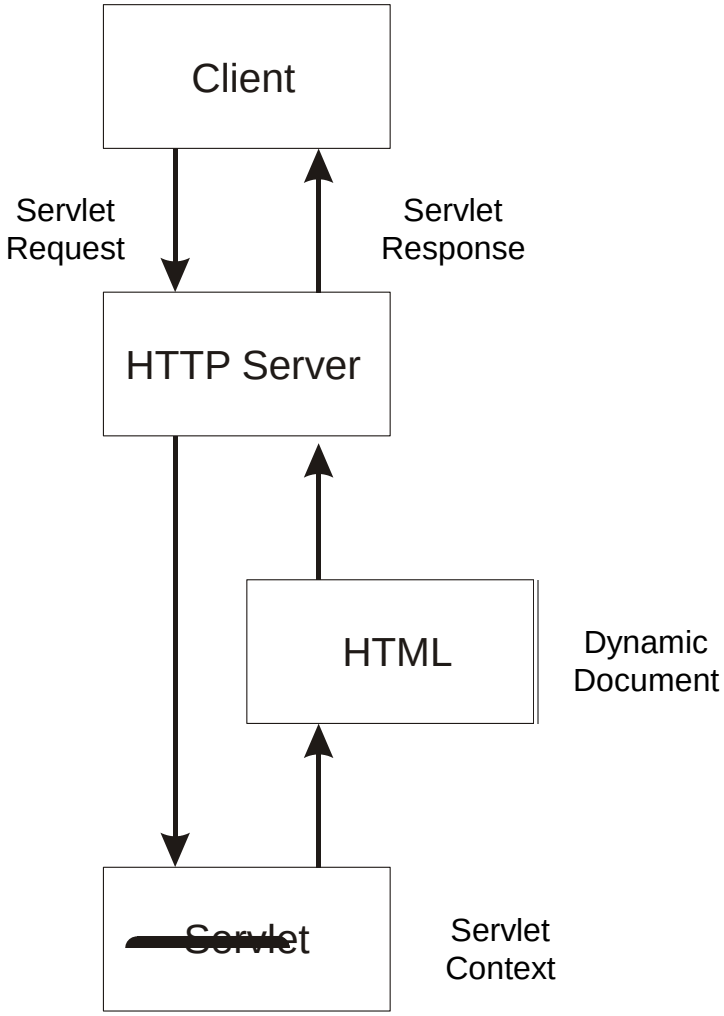


Conventional HTTP Request HTTP Servlet Request

Servlet Interface and Base Classes

- A Servlet that you create must
  - Implement the *Servlet* interface, and
  - Inherit the *GenericServlet* or *HTTPServlet* base class
    - *GenericServlet* is more “generic” and can be used, e.g, with Java RMI
    - *HTTPServlet* focuses on HTTP and on interfacing with Web Servers
- The base class creates all of the functionality required to map a client's request onto a Servlet
- To use a Servlet, you need to implement the code of the Servlet that retains a request, processes data, and sends a document back to the HTTP Server

HTTPServlet Base Class



The Servlet Lifecycle

- A Servlet runs in the same process as the Web Server, usually in a separate thread
- The Web Server is responsible for initializing, invoking, and destroying each Servlet instance
- The methods defined for the Servlet include
  - *init()*
  - *service()*
  - *destroy()*

The init() Method

- When a Servlet is first loaded, its *init()* method is invoked
- Within *init()*, the Servlet performs setup processing, such as opening files or establishing connections
- The Servlet's *init()* method has one parameter, a reference to a *ServletConfig* object
- The *ServletConfig* object
  - Provides initialization values for the Servlet
  - Has a *getServletContext()* method, which returns a *ServletContext* object
- If a Servlet is installed within a Web Server, it loads when the Web Server starts to run
- If not, the Web Server activates the Servlet when it receives the first request from a client for the services of the Servlet

ServletContext Object

- You can get the *ServletContext* object by calling *getServletContext()* on the *ServletConfig* object
- The *ServletContext* object contains information about the Servlet's environment
- The *ServletContext* object has several methods, including
  - *getServlet()* - Returns a Servlet object with a given name, which is useful when you want to access the services of the Servlet
  - *getServletNames()* - Returns an enumeration of the names of the Servlets available in the current namespace

The destroy() Method

- The Servlet's *destroy()* method cleans up resources (such as open files or database connections)
- The Web Server waits to call the *destroy()* method until all service calls are complete, or after a certain amount of time has passed (for long running services)

The service() Method

- Each Request message from a client results in a single call to the Servlet's *service()* method
- The *service()* method typically
  - Extracts information from the Request message
  - Accesses external resources (such as a database)
  - Produces the Response message

The service() Method (cont)

- The *service()* method has two parameters:
  - *HttpServletRequest* object, with data from the client, that consists of name / value pairs and either
    - *ServletInputStream*, obtained via *getInputStream()*, which provides **byte-level** data from the client, or
    - *BufferedReader*, obtained via *getReader()*, which provides **character-level** data from the client
  - *HttpServletResponse* object that represents the Servlet's reply to the client, using either
    - *ServletOutputStream*, obtained via *getOutputStream()*, which provides **byte-level** data to the client, or
    - *PrintWriter*, obtained via *getWriter()*, which provides **character-level** data to the client

Special Service Methods

- The *service()* method dispatches the HTTP messages to one of several special service methods that include:
  - *doGet()* - Replies to the client's request by sending an HTML page back to the client and the form that the client is to use
  - *doPost()* - Receives the form data that the client submits, does some processing, and sends an HTML page back to the client
- These methods correspond directly to the HTTP GET and POST methods

The doGet() Method

- The *doGet()* method has two parameters:
  - *HttpServletRequest* object - Provides information about the client's request, such as the types of parameters
  - *HttpServletResponse* object - Enables you to set the stream to which to write a dynamic document

public void doGet(HttpServletRequest req,

HttpServletResponse res)

throws ServletException, IOException

The doGet() Method (cont)

- Within the *doGet()* method, you need to specify the *ContentType* field of the Response header
- The ***ContentType*** field of the Response header defines the file type
- To set the *ContentType* field of the Response header, use the *setContentType()* method of the *HttpServletResponse* object, as in

res.setContentType("text/html");

Example: doGet() Method

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
        throws IOException, ServletException
{
   res.setContentType("text/html");

   PrintWriter out = res.getWriter();

   out.println("<html>");
   out.println("<head>");
   out.println("<title>Hello World!</title>");
   out.println("</head>");
   out.println("<body>")
   out.println("<h1>Hello World!</h1>");
   out.println("</body>");
   out.println("</html>");
}
```

Creating a Dynamic Document

- To create a dynamic document and send it back to the client, you create an *HtmlPage* object that handles the HTML formatting
- To do this, you need to get a *ServletOutputStream* to which you can write the dynamic document
- Then, you simply write HTML strings to that output stream
- The data are sent back to the client browser via the Web Server

Example: Creating a Dynamic Document

```
// Get the dynamic data for the HTML page from the
// client's request message (in this example)
String browserAddr = req.getRemoteAddr();
String userAgent = req.getHeader("user-agent");
if (userAgent == null)
    userAgent = "Unknown browser";
String method = req.getMethod();
String path = req.getServletPath();
String server = req.getServerName();
int port = req.getServerPort();
```

Example: Creating a Dynamic Document (cont)

```
// Get the output stream for the dynamic document
ServletOutputStream out = res.getOutputStream();

// Build the HTML document with the dynamic content
out.println("<html> " +
            "<head> " +
            "<title>Remote User Information"</title> " +
            "</head> " +
            "<body> " +
            "<p>Remote IP Address: " +  browserAddr +
            "<br>Remote Browser: " +  userAgent +
            "<br>Request Method: " +  method +
            "<br>Servlet: " +  path +
            "<br>Server: " +  server +
            "<br>HTTP Port: " +  port +
            "</body> " +
            "</html>");
```

HTML Document Produced by doGet()

<html>
<head>
<title>Remote User Information</title>
</head>
<body>
<p>Remote IP Address: 127.0.0.1
*
Remote Browser: Mozilla/4.5 [en] (Win00; U)*
*
Request Method: GET*
*
Servlet: /servlet/GetBrowserDataServlet*
*
Server: localhost*
*
HTTP Port: 8080*
</body>
</html>

Servlets and HTML Forms Processing

- **Forms** are the most common way to get data from a Web page to a Web Server
- Data can be processed or stored on the Web Server machine
- The data in a form are sent to the URL specified in the form

Form Tag

<form
***name**=“aname"*
***action**="http://servermachine.com/servlet/aservlet"*
***method**="get">*
Username: <input type="text" name="user">
Companyname: <input type="text" name="company">
</form>

where

- ***name*** - No embedded blanks
- ***action*** - URL that indicates where to send the data in the form
- ***method*** - Indicates to the browser how data from the form is to be sent to the Web Server, using either GET or POST

- The rest of the form consists of name-value pairs

Form Tag (cont)

- For ***method***, note that
  - *GET* - Attaches the data in the form to the URL sent to the server
    - The data are attached to the end of the URL after a ? as in:
      *http://google.com/servlet/aservlet**?user=john+dough&company=ibm***
    - The data are made available to the Servlet by invoking *getQueryString()* or *getParameterValues()*
  - *POST* - Instructs the browser to send the data as part of the Request header
    - The data are made available to the Servlet by reading the *ServletInputStream* or invoking *getParameterValues()*

Servlet Testing and Deployment

- Servlets are cached as soon as they are loaded, which is great for performance but difficult for testing
- To test a new version of a Servlet, you would have to get the Web Server administrator to
  - Turn off the Web Server, in order to clear the cache
  - Turn it back on again, in order to load the new version of the Servlet
- Instead, you should use the **servletrunner** software that allows you to test Servlets on your own computer