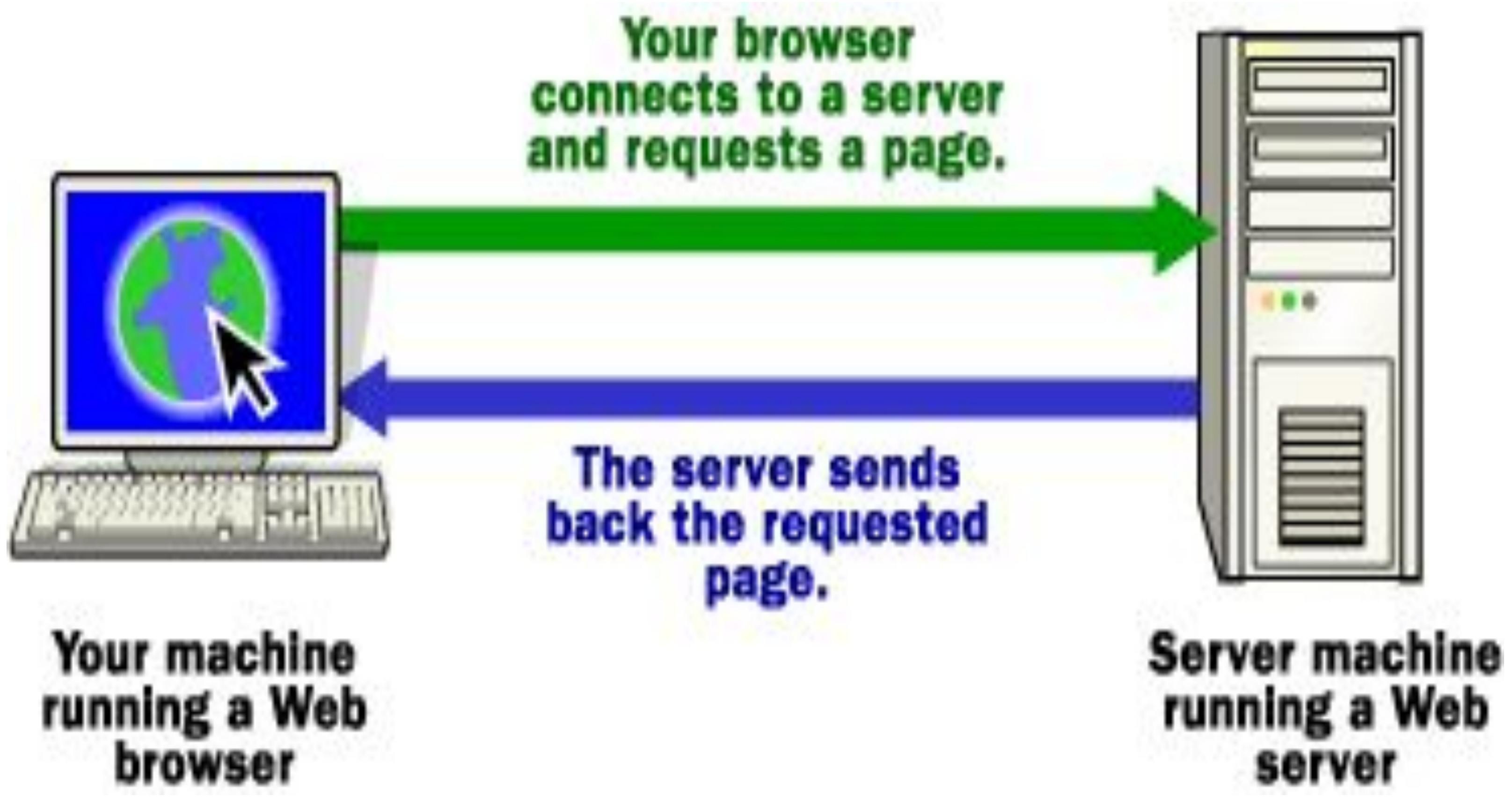
Servlets

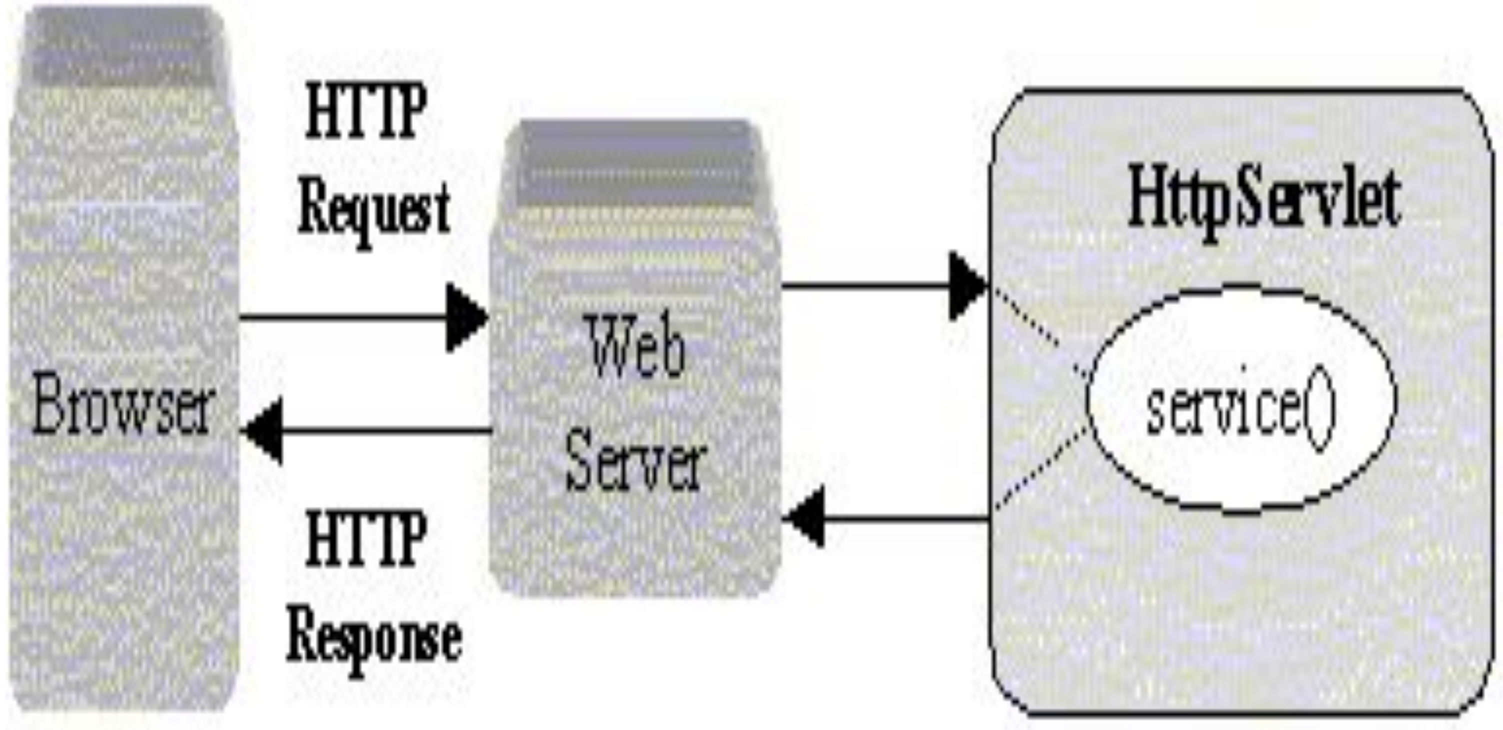
Objectives

- Understand the servlet request and response model
- Understand the servlet life cycle
- Identify the Servlet interfaces and classes
- Understand scope objects
- Understand the importance of a session tracking

How a web server works?


Your browser connects to a server and requests a page.
The server sends back the requested page.
Your machine running a Web browser
Server machine running a Web server

Servlet Request & Response Model



Sending Requests

- A user clicks on a hyperlink displayed in an HTML Page
- A user fills out a form in an HTML page and submits it
- A user enters in the browser's address field and press enter

Or

- a JavaScript function may call reload() method on the current document

Different HTTP methods

```
<form name='resultForm' method='GET'
  action='/resultServlet'>
Student-ID<input type='text'
  name='studentid'>
<input type='submit' value='GetMyResult'>
</form>
```

- By default, the browser uses the HTTP GET method in all of the above events (Ref: Sending Requests slide)
- **GET,POST,HEAD**

Comparing HTTP Methods

Feature	GET Method	POST Method
Target resource type	Active or passive	Active
Type of data	Text	Text as well as Binary
Amount of data	Maximum 255 chars	Unlimited
Visibility	Data is part of the URL and is visible to the user in the URL field of the browser	Data is not a part of the URL and is sent as the request message body. It is not visible to the user in the URL field of the browser
Caching	Data can be cached in the browser's URL history	Data is not cached in the browser's URL history

When to use GET & POST?

Use GET:

- To retrieve an HTML file or an image file

Use POST:

- To send a lot of data
- To upload a file
- To capture the username and password

HEAD: Same as GET except that for a HEAD request, the server returns only the response header but not the message. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

Handling HTTP Requests In An HttpServlet

- For every HTTP method we have in the HttpServlet

```
public void doXXX(HttpServletRequest
  request,HttpServletResponse response)
     throws ServletException,IOException
```

Sequence of events in HttpServlet

- The servlet container calls the
  service(ServletRequest request, ServletResponse response) method of HttpServlet
- The service(ServletRequest request,ServletResponse response) method of HttpServlet calls the service(HttpServletRequest request,HttpServletResponse response) method of same class.
- The service(HttpServletRequest request,HttpServletResponse response) method of HttpServlet calls appropriate doXXX()
  method of servlet.

Analyzing The Request(ServletRequest)

- The data sent by a browser includes parameters, meta information, and a text or binary data stream
- Understanding ServletRequest:
  - String getParameter(String paramName)
  - String[] getParameterValues(String paramName)
  - Enumeration getParameterNames()

Analyzing The Request(HttpServletRequest)

- Understanding the HttpServletRequest
  - It parses and interprets HTTP messages and provides the relevant information to the servlet.

```
<form action="../servlet/TestServlet " method="POST">
    Technology : <input type="text" name="searchstring"
   value="java">
  <br>
 State:<select name="city" size="5" multiple>
       <option value="HYD">Hyderabad</option>
        <option value="BAN">Bangalore</option>
        <option value="CHN">Chennai</option>
        </select>
    <br>
    <input type="submit" value="Search Job">
</form>
```

Analyzing The Request(HttpServletRequest)

```
Public void doPost(HttpServletRequest
  request,HttpServletResponse response)
{
   String searchString =
  request.getParameter("searchstring");
   String cityList =
  request.getParameterValues("city");
  // use the values and generate
  //appropriate response
}
```

Analyzing The Request (request headers)

HTTP requests include headers which provide extra information about the request

- Example of HTTP 1.1 Request:

```
GET /search? keywords= servlets+ jsp HTTP/ 1.1
Accept: image/ gif, image/ jpg, */*
Accept-Encoding: gzip
Connection: Keep- Alive
Cookie: userID= id456578
Host: www.sun.com
Referer: http:/www.sun.com/codecamp.html
User-Agent: Mozilla/ 4.7 [en] (Win98; U)
```

Analyzing The Request (request headers)

HTTP Request Headers

- Accept
  - Indicates MIME types browser can handle.
- Accept-Encoding
  - Indicates encoding (e. g., gzip or compress) browser can handle
- Authorization
  - User identification for password- protected pages
  - Instead of HTTP authorization, use HTML forms to send username/password and store info in session object

Analyzing The Request (request headers)

- Connection
  - In HTTP 1.1, persistent connection is default
  - Servlets should set Content-Length with setContentLength (use ByteArrayOutputStream to determine length of output) to support persistent connections.
- Cookie
  - Gives cookies sent to client by server sometime earlier.
  - Use getCookies, not getHeader
- Host
  - Indicates host given in original URL.
  - This is required in HTTP 1.1.

Analyzing The Request (request headers)

- If-Modified-Since
  - Indicates client wants page only if it has been changed after specified date.
  - Don’t handle this situation directly; implement getLastModified instead.
- Referer
  - URL of referring Web page.
  - Useful for tracking traffic; logged by many servers.
- User-Agent
  - String identifying the browser making the request.
  - Use with extreme caution!

Analyzing The Request (Retrieving request headers)

- HttpServletRequest methods for managing request headers
  - String getHeader(String headerName) – This method returns just one of the values associated with the given header.
  - Enumeration getHeaders(String headerName) – This method returns all the values associated with the header as an Enumeration of String object.
  - Enumeration getHeaderNames() – This method is useful when you don’t know the names of the headers

What is Servlet Response?

- Contains data passed from servlet to client
- All servlet responses implement ServletResponse interface
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information
- HttpServletResponse extends ServletResponse
  - HTTP response status code
  - Cookies

HTTP Response Status Codes

- Why do we need HTTP response status code?
  - Forward client to another page
  - Indicates resource is missing
  - Instruct browser to use cached copy

Methods for Setting HTTP Response Status Codes

- public void setStatus(int statusCode)
  - Status codes are defined in HttpServletResponse
  - Status codes are numeric fall into five general categories:
    - 100-199 Informational
    - 200-299 Successful
    - 300-399 Redirection
    - 400-499 Incomplete
    - 500-599 Server Error
  - Default status code is 200 (OK)

Example of HTTP Response Status

```
HTTP/ 1.1 200 OK
Content-Type: text/ html
<! DOCTYPE ...>
<HTML
...
</ HTML>
```

Common Status Codes

- 200 (SC_OK)
  - Success and document follows
  - Default for servlets
- 204 (SC_No_CONTENT)
  - Success but no response body
  - Browser should keep displaying previous document
- 301 (SC_MOVED_PERMANENTLY)
  - The document moved permanently (indicated in Location header)
  - Browsers go to new location automatically

Common Status Codes

- 302(SC_MOVED_TEMPORARILY)
  - Note the message is "Found“
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use sendRedirect, not setStatus,when setting this header
- 401 (SC_UNAUTHORIZED)
  - Browser tried to access password- protected page without proper Authorization header
- 404 (SC_NOT_FOUND)
  - No such page

Methods for Sending Error

- Error status codes (400-599) can be used in sendError methods.
- public void sendError(int sc)
  - The server may give the error special treatment
- public void sendError(int code, String message)
  - Wraps message inside small HTML document

Why HTTP Response Headers?

- Give forwarding location
- Specify cookies
- Supply the page modification date
- Instruct the browser to reload the page after a designated interval
- Give the file size so that persistent HTTP connections can be used
- Designate the type of document being generated Etc.

Methods for Setting Arbitrary Response Headers

- public void setHeader(String headerName, String headerValue)
  - Sets an arbitrary header
- public void setDateHeader(String name, long millisecs)
  - Converts milliseconds since 1970 to a date string in GMT format
- public void setIntHeader(String name, int headerValue)
  - Prevents need to convert int to String before calling setHeader
- addHeader, addDateHeader, addIntHeader
  - Adds new occurrence of header instead of replacing.

Methods for setting Common Response Headers

- setContentType
  - Sets the Content- Type header. Servlets almost always use this.
- setContentLength
  - Sets the Content- Length header. Used for persistent HTTP connections.
- addCookie
  - Adds a value to the Set- Cookie header.
- sendRedirect
  - Sets the Location header and changes status code.

Common HTTP 1.1 Response Headers

- Location
  - Specifies a document's new location.
  - Use sendRedirect instead of setting this directly.
- Refresh
  - Specifies a delay before the browser automatically reloads a page.
- Set-Cookie
  - The cookies that browser should remember. Don’t set this header directly.
  - use addCookie instead.

Common HTTP 1.1 Response Headers (cont.)

- Cache-Control (1.1) and Pragma (1.0)
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- Content- Encoding
  - The way document is encoded. Browser reverses this encoding before handling document.
- Content- Length
  - The number of bytes in the response. Used for persistent HTTP connections.

Common HTTP 1.1 Response Headers (cont.)

- Content- Type
  - The MIME type of the document being returned.
  - Use setContentType to set this header.
- Last- Modified
  - The time document was last changed
  - Don’t set this header explicitly.
  - provide a getLastModified method instead.

Refresh Sample Code

```
public class DateRefresh extends HttpServlet
{
   public void doGet(HttpServletRequest req,
      HttpServletResponse res)
      throws ServletException, IOException
    {
       res.setContentType("text/plain");
      PrintWriter out = res.getWriter();
      res.setHeader("Refresh", "5");
      out.println(new Date().toString());
    }
}
```

Writing a Response Body

- A servlet almost always returns a response body
- Response body could either be a PrintWriter or a ServletOutputStream
  - Using response.getWriter()
  - For character-based output
- PrintWriter
- ServletOutputStream
  - Using response.getOutputStream()
  - For binary (image) data

First Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
Public class HelloServlet extends HttpServlet
{
   public void doGet(HttpServletRequest request,
       HttpServletResponse response)
   throws ServletException, IOException
   {
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       out.println("<title>First Servlet</title>");
       out.println("<big>Hello Code Camp!</big>");
   }
}
```

Servlet Interfaces & Classes

- Servlet
- GenericServlet
- HttpServlet
- ServletRequest
- HttpServletRequest
- ServletResponse
- HttpServletResponse
- HttpSession

Servlet Life Cycle Methods

- Invoked by container
  - Container controls life cycle of a servlet
- Defined in
  - javax.servlet.GenericServlet class or
    - init()
    - destroy()
    - service() - this is an abstract method
  - javax.servlet.http.HttpServlet class
    - doGet(), doPost(), doXxx()
    - service() - implementation

Servlet Life Cycle Methods

- init()
  - Invoked once when the servlet is first instantiated
  - Perform any set-up in this method
- Setting up a database connection
- destroy()
  - Invoked before servlet instance is removed
  - Perform any clean-up
- Closing a previously created database connection

Setting Init Parameters in web.xml

```
<web-app>
        <servlet>
                <servlet-name>chart</servlet-name>
                <servlet-class>ChartServlet</servlet-class>
                <init-param>
                        <param-name>driver</param-name>
                        <param-value>COM.cloudscape.core.RmiJdbcDriver</paramvalue>
                </init-param>
                <init-param>
                        <param-name>url</param-name>
                        <param-value>jdbc:cloudscape:rmi:CloudscapeDB</param-value>
                </init-param>
        </servlet>
</web-app>
```

Servlet Life Cycle Methods

- service() javax.servlet.GenericServlet class
  - Abstract method
- service() in javax.servlet.http.HttpServlet class
  - Concrete method (implementation)
  - Dispatches to doGet(), doPost(), etc
  - Do not override this method!
- doGet(), doPost(), doXxx() in in javax.servlet.http.HttpServlet
  - Handles HTTP GET, POST, etc. requests
  - Override these methods in your servlet to provide desired behaviour

service() & doGet()/doPost()

- service() methods take generic requests and responses:
  - service(ServletRequest request, ServletResponse response)
- doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponse response)

Things You Do in doGet() & doPost()

- Extract client-sent information (HTTP parameter) from HTTP request
- Set (Save) and get (read) attributes to/from Scope objects
- Perform some business logic or access database
- Optionally forward the request to other Web components (Servlet or JSP)
- Populate HTTP response message and send it to client

Example: Simple doGet()

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
Public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
           HttpServletResponse response)
           throws ServletException, IOException {
// Just send back a simple HTTP response
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello J2EE Programmers! </big>");
    }
}
```

Steps of Populating HTTP Response

- Fill Response headers
- Set some properties of the response
  - Buffer size
- Retrieve an output stream from the response
- Write body content to the output stream

Example: Simple Response

```
Public class HelloServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    throws ServletException, IOException
    {
        // Fill response headers
        response.setContentType("text/html");
        // Set buffer size
        response.setBufferSize(8192);
        // Retrieve an output stream from the response
        PrintWriter out = response.getWriter();
        // Write body content to output stream
        out.println("<title>First Servlet</title>");
        out.println("<big>Hello J2EE Programmers! </big>");
    }
}
```

Scope Objects

- Enables sharing information among collaborating web components via attributes maintained in scope objects
- Attributes of Scope objects are accessed with
  - getAttribute()
  - setAttribute()
- 4 Scope objects are defined
  - Web context, session, request, page

Four Scope Objects: Accessibility

- Web context (ServletConext)
  - Accessible from Web components within a Web context
- Session
  - Accessible from Web components handling a request that belongs to the session
- Request
  - Accessible from Web components handling the request
- Page
  - Accessible from JSP page that creates the object

Four Scope Objects: Class

- Web context
  - javax.servlet.ServletContext
- Session
  - javax.servlet.http.HttpSession
- Request
  - subtype of javax.servlet.ServletRequest:
    javax.servlet.HttpServletRequest
- Page
  - javax.servlet.jsp.PageContext

What is ServletContext For?

- Used by servlets
  - Set and get context-wide object-valued attributes
  - Get request dispatcher
    - To forward or include web component
  - Access Web context-wide initialization parameters set in the web.xml file
  - Access Web resources associated with the Web context
  - Log
  - Access other misc. information

Scope of ServletContext

- Context-wide scope
  - Shared by all servlets and JSP pages within a "web application“
- Why it is called “web application scope”
  - A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace and possibly installed via a *.war file
- All servlets in BookStore web application share same ServletContext object
  - There is one ServletContext object per "web application" per Java Virtual Machine

How to Access ServletContext Object?

- Within your servlet code, call getServletContext()
- Within your servlet filter code, call getServletContext()
- The ServletContext is contained in ServletConfig object, which the Web server provides to a servlet when the servlet is initialized
  - init (ServletConfig servletConfig) in Servlet interface

Why Session Tracking?

- Need a mechanism to maintain state across a series of requests from the same user (or originating from the same browser) over some period of time
  - Example : Online shopping cart
- Yet, HTTP is stateless protocol
  - Each time, a client talks to a web server, it opens a new connection
  - Server does not automatically maintains "conversational state" of a user

Session Tracking Use Cases

- When clients at an on- line store add an item to their shopping cart, how does the server know what’s already in the cart?
- When clients decide to proceed to checkout, how can the server determine which previously created shopping cart is theirs?

Three “underlying” SessionTracking Mechanisms

- Cookies
- URL rewriting
- Hidden form fields
- Note that these are just underlying

  mechanisms of passing “session id”
  - do not provide high-level programming APIs
  - do not provide a framework for managing sessions
  - This is what Servlet Session Tracking feature provides

What is HTTP Cookie?

- Cookie is a small amount of information sent by a servlet to a Web browser
- Saved by the browser, and later sent back to the server in subsequent requests
  - A cookie has a name, a single value, and optional attributes
  - A cookie's value can uniquely identify a client
- Server uses cookie's value to extract information a bout the session from some location on the server

Cookies as Session Tracking Mechanism

- Advantages:
  - Very easy to implement
  - Highly customizable
  - Persist across browser shut-downs
- Disadvantages:
  - Often: users turn off cookies for privacy or security reason
  - Not quite universal browser support

URL Rewriting

- URLs can be rewritten or encoded to include session information.
- URL rewriting usually includes a session id
- Session id can be sent as an added parameter:
  - http://.../servlet/Rewritten?sessionid=688

URL Rewriting as Session Tracking Mechanism

- Advantages:
  - Let user remain anonymous
  - They are universally supported (most styles)
- Disadvantages:
  - Tedious to rewrite all URLs
  - Only works for dynamically created documents

Hidden Form Fields

- Hidden form fields do not display in the browser, but can be sent back to the server by submit
  <INPUT TYPE=”HIDDEN” NAME=”session” VALUE=”...”>
- Fields can have identification (session id) or just some thing to remember (occupation)
- Servlet reads the fields using req.getParameter()

Hidden Form Fields as Session Tracking Mechanism

- Advantages:
  - Universally supported.
  - Allow anonymous users
- Disadvantages:
  - Only works for a sequence of dynamically generated forms
  - Breaks down with static documents, emailed documents, bookmarked documents.
  - No browser shutdowns.

Now Without “Session Tracking” Feature of Servlet

- Servlet programmers have to perform the following tasks themselves by using one of three session tracking mechanisms
  - Generating and maintaining a session id for each session
  - Passing session id to client via either cookie or URL
  - Extracting session id information either from cookie or URL
  - Creating and maintaining a hash table in which session id and session information are stored
  - Coming up with a scheme in which session information can be added or removed

“Session Tracking” Features of Servlet

- Provides higher-level API for session tracking
  - Built on top of Cookie or URL rewriting
- Servlet container maintains
  - internal hash table of session id's
  - session information in the form of Http Session
- Generates and maintains session id transparently
- Provides a simple API for adding and removing session information (attributes) to Http Session
- Could automatically switch to URL rewriting if cookies are unsupported or explicitly disabled

Http Session

- To get a user's existing or new session object:

  HttpSession session = request get Session (true);

  - "true" means the server should create a new session object if necessary
- HttpSession is Java interface
- Container creates a object of Http Session type

Example: Getting Http Session Object

```
public class Catalog Servlet extends Http Servlet
   {
      public void doGet (Http Servlet Request
   request, HttpServletResponse response)
       throws Servlet Exception, IOException {

       // Get the user's session and shopping cart
    Http Session session
   =request.getSession(true);
    ...
    out = response.getWriter();
    ...
       }
    }
```

Http Session Java Interface

- Contains Methods to
  - View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
  - Bind objects to sessions, allowing user information
  - To persist across multiple user connections

Store and Retrieve of Attribute

- To stores values:
  - session.setAttribute("cartItem", cart);
- To retrieves values:
  - session.getAttribute("cartItem");

If Cookie is turned off..

- If your application makes use of session objects
  - you must ensure that session tracking is enabled by having the application rewrite URLs whenever the client turns off cookies
  - by calling the response's encodeURL(URL) method on all URLs returned by a servlet
  - This method includes the session ID in the URL only if cookies are disabled; otherwise, it returns the URL unchanged

String response.encodeURL(URL)

- Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged
  - Implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL
  - For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary
- For robust session tracking, all URLs emitted by a servlet should be run through this method
  - Otherwise, URL rewriting cannot be used with browsers which do not support cookies

Example: URL

- If cookies are turned off
  - http://localhost:8080/bookstore1/cashier;jsessionid=c0o7fszeb1
- If cookies are turned on
  - http://localhost:8080/bookstore1/cashier

Session Timeout

- Used when an end-user can leave the browser without actively closing a session
- Sessions usually times out after 30 minutes of inactivity
  - Product specific
  - A different timeout may be set by server admin
- getMaxInactiveInterval(), setMaxInactiveInterval() methods of HttpSession interface
  - Gets or sets the amount of time, session should go without access before being invalidated

Session Invalidation

- Can be used by servlet programmer to end a session proactively
  - when a user at the browser clicks on "log out" button
  - when a business logic ends a session ("checkout" page in the example code in the following slide)
- public void invalidate()
  - Expire the session and unbinds all objects with it
- Caution
  - Remember that a session object is shared by multiple servlets/JSP-pages and invalidating it could destroy data that other servlet/JSP-pages are using

Concurrency Issues on a Servlet

- The service() method of a servlet instance can be invoked by multiple clients (multiple threads)
- Servlet programmer has to deal with concurrency issue
  - shared data needs to be protected
  - this is called "servlet synchronization"
- 2 options for servlet synchronization
  - use of synchronized block
  - use of SingleThreadModel

Use of synchronized block

- Synchronized blocks are used to guarantee only one thread at a time can execute within a section of code

```
synchronized(this) {
   myNumber = counter + 1;
   counter = myNumber;
}
...
synchronized(this) {
counter = counter - 1 ;
}
```

SingleThreadModel Interface

- Servlets can also implement javax.servlet.SingleThreadModel
- The server will manage a pool of servlet instances.
- Guaranteed there will only be one thread per instance.
- This could be overkill in many instances

```
 Public class SingleThreadModelServlet
extends HttpServlet implements
SingleThreadModel
 {
        ...
 }
```

Including another web resource

- Get RequestDispatcher object from ServletConext object
  - RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner");
- Then, invoke the include() method of the RequestDispatcher object passing request and response objects
  - dispatcher.include(request, response);

Forwarding to another web resource

- Get RequestDispatcher object from HttpServletRequest object
  - Set “request URL” to the path of the forwarded page RequestDispatcher dispatcher = request.getRequestDispatcher("/template.jsp");
- If the original URL is required for any processing, you can save it as a request attribute
- Invoke the forward() method of the RequestDispatcher object
  - dispatcher.forward(request, response);

Bibliography

- SCWCD Exam study kit – Hanumant Deshmukh and Jignesh Malavia
- Core Servlets and JavaServer Pages, volume1:core Technologies, 2nd edition – Marty Hall and Larry Brown
- ServletBasics_speakernoted from Sun Microsystems
- ServletAdvanced_speakernoted from Sun Microsystems