MAINTAINING STATE

Greater Success with Greater Breadth of Awareness

Post versus Get messages

GET /advisor/selectBeerTaste.do?color=dark&taste=malty HTTP/1.1
Host: www.wickedlysmart.com
…
Connection: keep-alive

POST /advisor/selectBeerTaste.do HTTP/1.1
Host: www.wickedlysmart.com
…
Connection: keep-alive

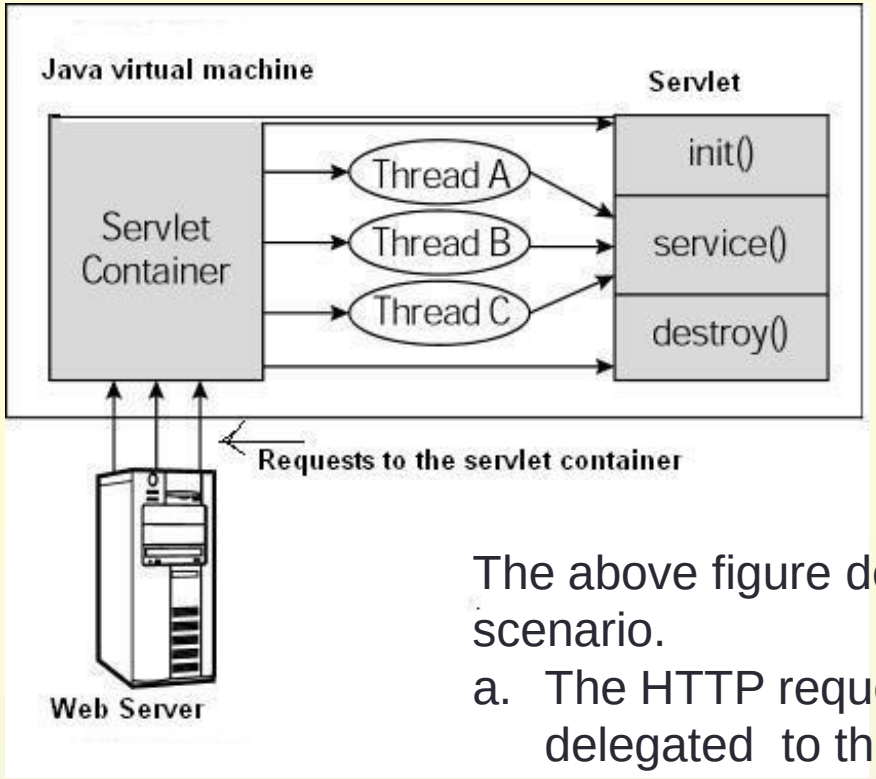
color=dark&taste=malty

- post has a body
- post more secure since parameters not visible in browser bar
- Intention of GET is to retrieve data
- POST is to send data to be processed and stored
  - GET “should” be idempotent
  - POST generally not

The Servlet Lifecycle, the Servlet Container, and Maintaining State

HTTP is a *stateless protocol*, which means that after a web server responds to a request, there is no “memory” of the request or who sent it. To maintain a client-server conversation, it is necessary to get past this barrier. In the world of servlets, keeping track of client identity was accomplished using client cookies; keeping track of client was accomplished by maintaining an HttpSession. These activities are conducted behind the scenes by the *servlet container,* which plays the role of natural law in the world of Java web applications.

Servlet Lifecycle



The above figure depicts a typical servlet life-cycle scenario.

a. The HTTP requests coming to the server are delegated to the servlet container.
b. The servlet container loads the servlet before invoking the service() method.
c. Then the servlet container handles multiple requests for the same servlet by spawning multiple threads, one thread per request, each executing the service() method of a single instance of the servlet.

Servlet Lifecycle – Initialization

- First, the container loads the servlet's class:
  - Can happen at container startup or the first time the client invokes the servlet. (`Class.forName`)
- Container creates an instance of the servlet class (using its constructor)
- Container then invokes the init() method.

Servlet Lifecycle - Processing

- When a client request arrives at the container, container uses the URL to determine which servlet to use.
- Container creates request and response objects
- The container invokes the service() method.
  - Called for every incoming HTTP request.
  - Container passes the request and response objects to service()

Servlet Lifecycle - Processing

- ```
  public void service(ServletRequest request,
  ServletResponse response)
      throws ServletException,
                      IOException{ }
  ```
- The service() method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- It is rare to override service() method; typically you will just override either doGet() or doPost() (or both) depending on what type of request you receive from the client
  ```

Servlet Lifecycle - Processing

- The doGet() and doPost() are the most frequently used methods for service request.
- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified; in these cases, the doGet() method should be asked to handle the request
  - `public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { }`
- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.
  - `public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { }`

Servlet Lifecycle - Destroy

- The container may decide to release a servlet instance at any time:
  - Usually when the container is shutting down.
- Container invokes the destroy method:
  - In this method, you should release resources by closing database connections, halting background threads and performing other such clean up activities.
  - public void destroy(){}
- The servlet instance is released for garbage collection.

Redirecting and forwarding requests

- servlets may internally pass (“forward”) the request processing to another local resource or to tell the client browser to issue another HTTP request to a specified URL

- forward (to another servlet or jsp in same website)

```
RequestDispatcher view = request.getRequestDispatcher(“result.jsp”);
view.forward(request, response);
```

- redirect

```
response.sendRedirect(“http://www.cs.mum.edu”);  //to another site
or
response.sendRedirect(“result.jsp”);    //within same site
```

Difference between redirect and forward

- **forward**
  - passes the request to another resource on the server
    - sometimes referred as “server side redirect“
  - request and response objects passed to destination servlet.
  - Browser is completely unaware of servlet forward and hence the URL in browser address bar will remain unchanged

- **redirect**
  - server sends HTTP status code 3xx to client along with the redirect URL (usually 302 temporary redirect)
  - client then sends a new request to the URL
  - extra round trip
  - address bar will change to new URL
  - only http message sent, request and response objects cannot be sent

HTTP input parameters

- send data from your HTML page to the servlet
- HTML
  - <input name="userName" type="text" />
- HTTP
  - GET /somepath/myservlet?userName=Fred HTTP/1.1
- Servlet
  - String input = request.getParameter("userName");
- What if the html tag may have multiple values, like a multiple selection list or check box.
  - GET /somepath/myservlet?colors=red&colors=blue&colors=green HTTP/1.1
  - String[] inputColors = request.getParameterValues("colors");
- Input parameters are always text/Strings
  - must validate and convert as needed
  - E.g., numbers, Dates, etc.

Main point 1

- 1. HTTP request messages send input parameters as name/value pairs. Input parameters are text that must be accessed and converted by a servlet. This is the main mechanism web apps use to send information from the browser to the server. **Science of Consciousness**: At the level of the unified field one experiences frictionless flow of information.

Managing state information

- different “scopes” of information a web app might need to manage
- request scope: short term computed results to pass from one servlet to another (i.e., “forward”)
- session scope: conversational state info across a series of sequential requests from a particular user
- application/context scope: global info available to any other users or servlets in this application
- long term permanent or persistent info in an external database

What is an attribute?

- An object bound into one of the three servlet API objects
  - HttpServletRequest
  - HttpSession
  - ServletContext
- Is a name value pair
  - value has type Object
  - name is String
- Methods (on request, session, or context objects)
  - getAttribute(String)
    - must cast to specific type
    - E.g., suppose set(“manager”, bob) and bob is type Person
      - Person savedManager = (Person) request.getAttribute(“manager”);
  - setAttribute(String, Object)
  - removeAttribute(String)
    - To remove from scope
  - getAttributeNames()
    - To get an Enumeration of all attributes in scope

Request scope attributes

- only be available for that request.
  - request. setAttribute(“myAttr”, test);
  - request. getAttribute(“myAttr”);

Request.getParameter(param)

The Request object reads parameters from a submitted HTML form by reading the *name* property

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    String message = "Hello %s";
    String name = request.getParameter("name");
```

```
<form action="hello" method="get">
Name: <input type = "text" name = "name"/><br/>
<input type = "submit" value = "Submit"/>
</form>
```

Context scope attributes

- Application level state
  - request.getServletContext().setAttribute(“myAttr”, test);
  - request.getServletContext().getAttribute(“myAttr”);
- Caution: global!!
  - Shared by every servlet and every request in the application
  - Like nuclear power
    - very powerful
    - have to be careful
  - Not thread safe
    - Nor session attributes
    - Only request attributes thread safe

Session scope attributes

- sessions are collections of objects (attributes) that are unique to a set of connected requests from a single browser
- Sessions are a critical state management service provided by the web container

Main point 2

Attributes are objects on the server. They promote communication between components. Only request attributes are thread-safe. **Science of Consciousness**: Our experience of transcending facilitates coherence and communication between different components of our brains and minds. Such communication is critical to successful thought and actiion.

Session tracking via cookie exchange

Session tracking via cookie exchange

- A web conversation, holds information across multiple requests.
- before container sends back to browser, it saves the session info to some datastore, and then sends an HTTP “cookie” with session id back to the browser
  - Set-cookie header
- browser stores the info and puts cookie/sessionid back into a header with the next request
  - Cookie header
  - All cookies from a given website will be returned whenever the browser sends any request to that website
    - Suppose have multiple tabs open to the same website
- container then needs to reconstitute session from storage before calling servlet with subsequent requests in the “conversation”

Session lifetime

Client side

- Browser discards all “temporary” cookies when it closes
- Every tab or window of browser will have access to all cookies

Server side

- How to get a session
  - session = request.getSession(); //creates new session if none exists
    - session.isNew(); //checks whether is a new session
    - request.getSession(false); //returns null if none exists
- How to get rid of the session
  - sessions can become a memory resource issue
  - container can’t tell when browser is finished with session
  - 3 ways for container to remove sessions
    - session timeout in the DD
    - session.setMaxInactiveInterval(20*60); //seconds
    - session.invalidate(); //immediate

```
<session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

(HTTP) Cookies

- can be used for other things besides implementing sessions
  - temporary cookie
    - browser removes when it closes
    - this is default
    - session cookies are like this
  - permanent cookie
    - a cookie that has a max age set

- Sending a Cookie

```
Cookie cookie = new Cookie(“Name”, “Jack”);
cookie.setMaxAge(minutes);
response.addCookie(cookie);
```

- Reading a cookie
  - Cookies come with the request
  - can only get all cookies, then search for the one you want.

Cookies

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

Cookies

- Obtain a list of cookies set by the browser by calling

  request.getCookies()

  Returns an array of Cookie objects

- Example:

```
final Cookie[] cookies = request.getCookies();
String userId = null;
for (int i = 0; i < cookies; ++i) {
if (cookies[i].getName().equals("UserId"))
        userId = cookieList[i].getValue();
}
```

Maintaining State Demo

5 ways to maintain state

Container managed state (3 scopes)

1. request scope: destroyed when servlet finishes processing request
2. session scope: destroyed when user closes browser
3. application scope destroyed when Container stopped.

4. Cookies saved on browser,
   temporary (deleted when the browser closes)
   permanent
5. Hidden fields on a form

Main point 3

Web applications can use many different types of memory management. State information can be stored in request, session, or context scope, and also as hidden fields or cookies. **Science of Consciousness**: The ultimate scope of memory is the infinite scope and comprehension of pure consciousness, We experience this level when we transcend, and having this experience supports the efficient operation and integration of all the more concrete levels of memory and thought in the human mind.