# LESSON 3: 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
- bookmarking POST requests does not work (well) in browsers
  - don't support since POST not idempotent
  - can bookmark, but loses body info and becomes a GET
- Intention of GET is to retrieve data; POST is to send data to be processed and stored
  - GET "should" be idempotent
  - POST generally not

# 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.

# 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.

# 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
    - Vs String for input parameter
  - 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”);

# 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

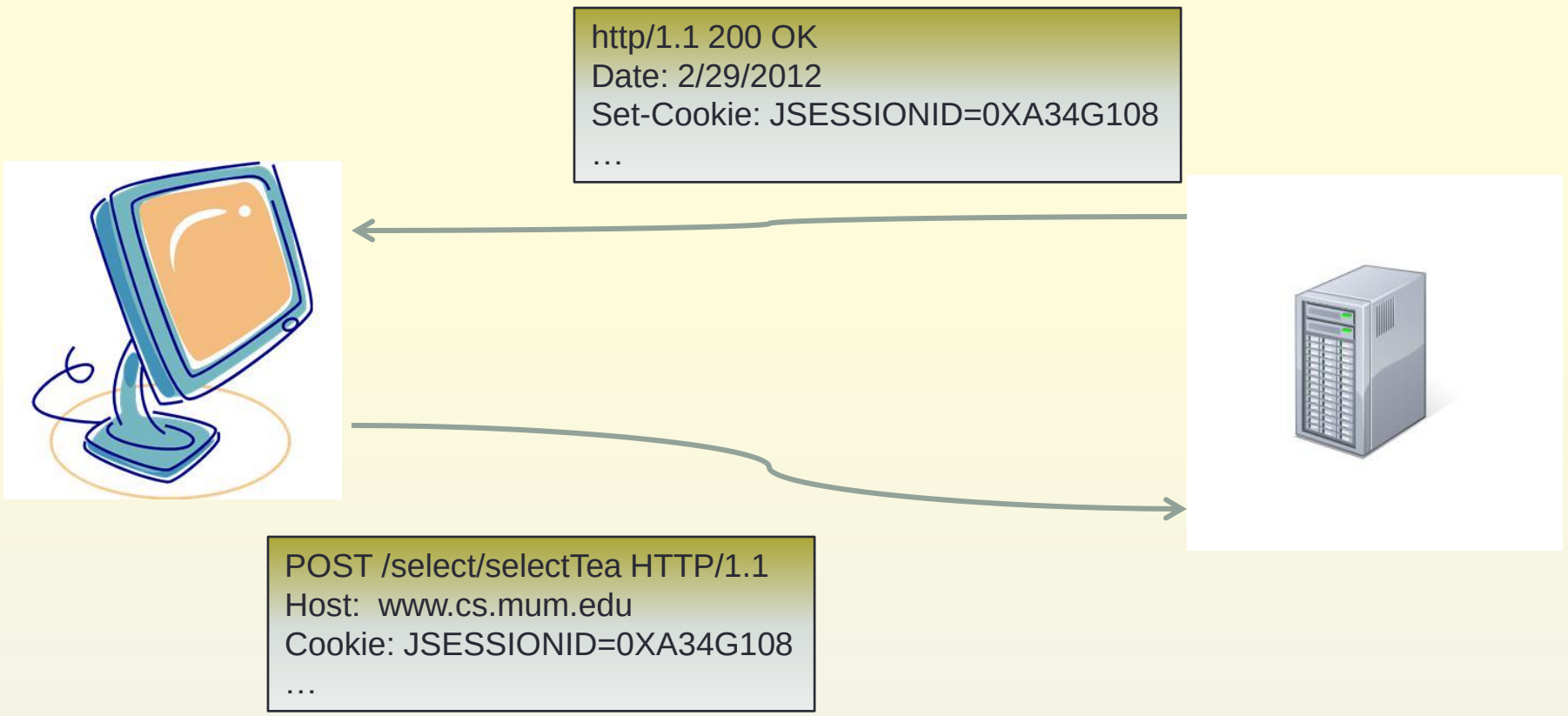
# Main point 2

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.

# 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.

# 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.

```
for (Cookie cookie : request.getCookies()) {
if (cookie.getName().equals("Name")) {
String userName= cookie.getValue();  } }
```

- RFC2109
  - specifies a way to create a stateful session with HTTP requests and responses.
  - describes two new headers, Cookie and Set-Cookie

# 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 Tomcat container stopped.

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

- Answer questions 14,15,16 for today's quiz

# 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.