Maintaining State

Greater Success with Greater Breadth of Awareness

HTTP is Stateless

- HTTP is a stateless protocol.
  - Every request is new request to the server.
  - From the server's perspective, the request begins when the client opens a socket to the server, and it ends when the server sends the last packet back to the client and closes the connection.

Why is state necessary?

- Application often cannot function correctly being totally stateless.
  - A classic example is the online shopping website.
    - Somehow, between every request you made, the website knows those requests were coming for the same browser on the same computer and associates that with users shopping cart.
  - Remembering users throughout the *session*, is another use case.
  - Enabling Application Workflow
    - Often users need some form of workflow to complete a task
    - For e.g. checkout task involves
      - Filling billing address, shipping address
      - Filling payment details
      - Confirmation

How to maintain state?

- Container managed states
  - **Request scope**: life time is for a request-response cycle.
  - **Session scope**: life time is throughout the session.
  - **Application scope**: life time is for the life time of the application.

- Other ways
  - Cookies
    - temporary vs permanent
  - Hidden form fields

What is an attribute?

- An object bound into one of the three servlet API objects
  - `HttpServletRequest`
  - `HttpSession`
  - `ServletContext`

- Is a name value pair
  - *name* is String
  - *value* has type Object

- Methods (on request, session, or context objects)
  - `getAttribute(String)`
    - must cast to specific type as return type is Object.
  - `setAttribute(String, Object)`
  - `removeAttribute(String)`
    - To remove from scope
  - `getAttributeNames()`
    - To get an Enumeration of all attributes in scope
    - useful for looping through list of attributes

Request scope attributes

- Only be available for that request (thread safe).
  - `request.setAttribute("myAttr", val);`
  - `request.getAttribute("myAttr");`

Session scope attributes

- Available through the session
  - `request.getSession().setAttribute("myAttr", val);`
  - `request.getSession().getAttribute("myAttr");`

- Not thread safe
  - What if user opens new tab or window and do operation simultaneously?
  - Only request attributes thread safe

Context scope attributes

- Application level state
  - `request.getServletContext().setAttribute("myAttr", val);`
  - `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

Redirecting and forwarding requests

- Servlets may internally pass 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 application)

```
RequestDispatcher dispatcher = request.getRequestDispatcher("result.jsp");
dispatcher.forward(request, response);
```

  - redirect (to a specified URL)

```
response.sendRedirect("http://www.cs.mum.edu");// to external link
response.sendRedirect("result.jsp"); // within same application
```

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 (it's a new request, state info in previous request is cleared.)

Post Redirect Get (PRG) pattern

- Instead of returning a web page directly, the POST operation returns a redirection command.
- Post/Redirect/Get (PRG) is a web development design pattern that prevents some duplicate form submissions, creating a more intuitive interface for user agents (users).
  - PRG supports bookmarks and the refresh button in a predictable way that does not create duplicate form submissions.

How to choose right scope?

- Request scope:
  - Used to maintain short term computed results to pass from one servlet to another (same request)
- Session scope:
  - Used to maintain conversational state info across a series of sequential requests from a particular user.
- Application/context scope:
  - Used to maintain global info available to any other users or servlets in this application
    - Usually long term permanent or persistent info that does not change frequently.

Note: use the smallest scope possible.

Main point

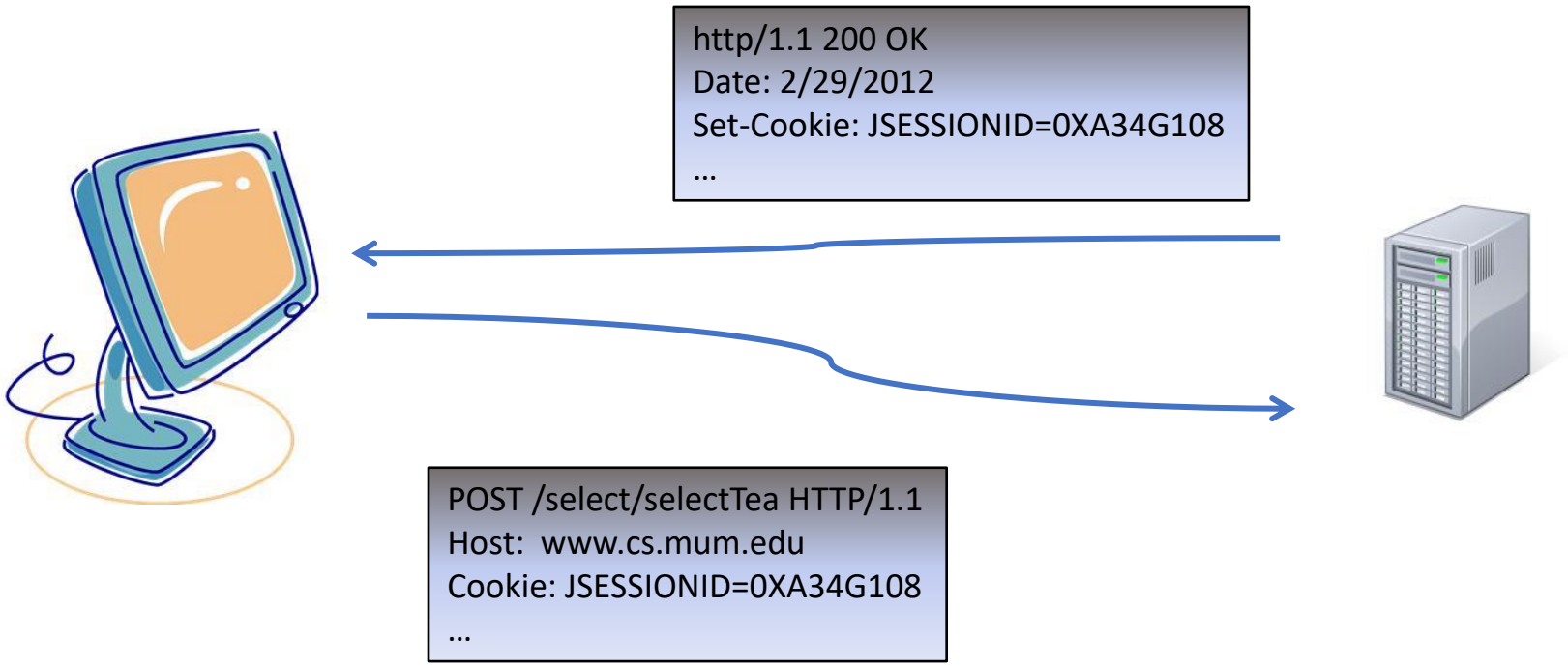
Attributes are objects on the server. They promote communication and integration between components. **Science of Consciousness**: Our experience of transcending facilitates coherent communication between different components of our brains and minds. Such communication is critical to successful thought and action.

Sessions

- Session persists for a specified time period, across more than one connection or page request from the user.
  - A session usually corresponds to one user.
- Sessions store 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

Maintaining session using cookie



Cookies are saved on user's (client) machine as plain text file. Storage location may vary based on os and/or browsers.

Session tracking via cookie exchange

- Before container sends response back to browser, it saves the session info to some data-store, and then sends an HTTP "cookie" containing session id back to the browser along with the response using set-cookie response header.
- Browser stores session id inside a cookie named JSESSIONID (for Java) and sends session id back to the server for every other request by setting it into the request header using Cookie request header.
- A web application can set more than one cookie and all cookies from a given application will be returned whenever the browser sends any request to that website.
- Container then needs to reconstitute session from storage before calling servlet with subsequent requests in the conversation.

What if cookie is disabled?

- Another popular method for transmitting session IDs is through URLs.
  - The web or application server knows to look for a particular pattern containing the session ID in the URL and, if found, retrieves the session from the URL.
  - Different technologies use different strategies for embedding and locating session IDs in the URL.
    - example, PHP uses a query parameter named *PHPSESSID*:

```
http://www.example.com/support?PHPSESSID=NRxclGg2vG7kI4MdlLn&foo=bar&high=five
```

  - Java EE applications use a different approach.
    - The session ID is placed in a matrix parameter

```
http://www.example.com/support;JSESSIONID=NRxclGg2vG7kI4MdlLn?foo=bar&high=five
```

  - URL rewriting is automatic fallback if cookies are disabled.

(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 and Receiving a cookie

- Sending a Cookie

```
Cookie cookie = new Cookie("Name", "Jack");
cookie.setMaxAge(3600); //in seconds
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();
    }
}
```

To delete a persisting cookie you have to resend cookie with same name after setting its max age value to be negative integer.

Session lifetime

- Obtaining a session

```
HttpSession 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
```

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

```
// Programmatic ways

session.setMaxInactiveInterval(20*60); //seconds
session.invalidate();  //immediate
```

Main point

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.

Servlet Filters

Filters

- Filters are application components that can intercept requests to resources, responses from resources, or both, and act on those requests or responses in some manner.
  - Filters can inspect and modify requests and responses, and they can even reject, redirect, or forward requests.
- The `javax.servlet.Filter` interface is very simple and involves the `HttpServletRequest` and `HttpServletResponse` that we are already familiar with.
  - Like Servlets, filters can be declared in the deployment descriptor, programmatically, or with annotations, and they can have init parameters and access the `ServletContext`.

Uses of filters

- The number of uses for filters is really limited only by your imagination:
  - Logging Filters
    - Filters can be especially useful for logging activity in your application.
  - Authentication Filters
    - You can centralize your authentication code in a authentication filter.
  - Compression and Encryption Filters
  - Error Handling filters
  - more…

Using filters

- Creating a filter is as simple as implementing the Filter interface.
  - Its `init` method is called when the filter is initialized and provides access to the filter's configuration, its init parameters, and the `ServletContext`, just like the init method in a Servlet.
- Similarly, the `destroy` method is called when the web application is shut down.
- The `doFilter` method is called when a request comes in that is mapped to the filter.
  - It provides access to the `ServletRequest`, `ServletResponse`, and `FilterChain` objects.
- Within `doFilter` you can either reject the request or continue it by calling the `doFilter` method of the `FilterChain` object; you can alter the request and the response; and you can wrap the request and response objects.

Implementing Filter

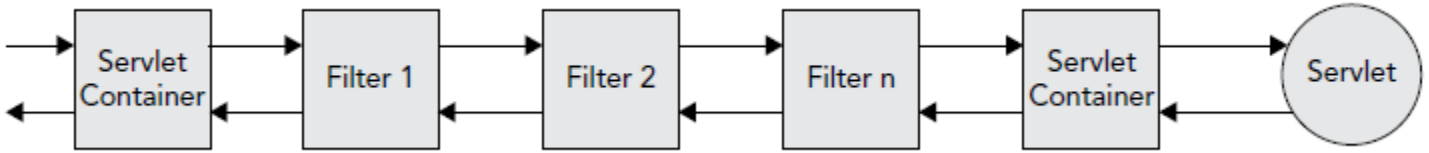
```
public class MyFilter implements Filter {
    @Override
    public void doFilter(ServletRequest request,
                                ServletResponse response, FilterChain chain)
                                throws IOException, ServletException {
        boolean test=true;
        if(test){
            // manipulate request
            chain.doFilter(request, response);
            // manipulate response
        }else{/*send response*/}
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void destroy() {}
}
```

Filter Chain

- Any number of filters may intercept a request.
  - Calling `FilterChain.doFilter()` triggers the continuation of the filter chain.
  - If the current filter is the last filter in the chain, calling `FilterChain.doFilter()` returns control to the Servlet container, which passes the request to the Servlet.
  - If the current filter does not call `FilterChain.doFilter()`, the chain is interrupted, and the Servlet and any remaining filters never handle the request.



  - A filter can take action on a request both before and after the destination Servlet services it.

Mapping Filters

- Just like Servlets, filters can be mapped to URL patterns.
  - This determines which filter or filters will intercept a request.
  - Any request matching a URL pattern to which a filter is mapped first goes to that filter before going to any matching Servlets

- Instead of mapping your filter to a URL or URLs, you can map it to one or more Servlet names.
  - If a request is matched to a Servlet, the container looks for any filters mapped to that Servlet's name and applies them to the request

Using the Deployment Descriptor

```
<filter>
    <filter-name>myFilter</filter-name>
    <filter-class>com.example.MyFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>myFilter</filter-name>
    <url-pattern>/foo</url-pattern>
    <url-pattern>/bar/*</url-pattern>
    <servlet-name>myServlet</servlet-name>
</filter-mapping>
```

Using Annotations

```
@WebFilter(
        filterName = "myFilter",
        urlPatterns = { "/foo", "/bar/*" },
        servletNames = { "myServlet" }
       )
public class MyFilter implements Filter
```

Demo – a simple Logging Filter

- The `RequestLogFilter` class in demo is the first filter in the filter chain for all requests to the application.
- It times how long a request takes to process and logs information about every request that comes in to the application — the IP address, timestamp, request method, protocol, response status and length, and time to process the request — similar to the Apache HTTP log format.
- The logging happens in the finally block so that any exceptions thrown further down the filter chain do not prevent the logging statement from being written.

Servlet Listeners

Events and Listeners

- Events are basically occurrence of stimuli.
  - State change of the server side objects (request, session, context) are stimuli of our interest.
  - We can bind listener (method) to the subject, for event of our interest and take our actions when specified event fires.

Events interfaces

- `ServletRequestListener`
- `ServletRequestAttributeListener`
- `ServletContextListener`
- `ServletContextAttributeListener`
- `HttpSessionListener`
- `HttpSessionAttributeListener`
- `HttpSessionBindingListener`
- `HttpSessionActivationListener`

  - Implement the one you are interested in.

Events classes

- `ServletRequestEvent`
- `ServletContextEvent`
- `ServletRequestAttributeEvent`
- `ServletContextAttributeEvent`
- `HttpSessionEvent`
- `HttpSessionBindingEvent`

  - Corresponding event object types that is passed to the listeners when they are fired.

Attaching (listing) Listeners

```
@WebListener
public class MyListenerClass implements xxxListener
{
    ...
}
```

```
<listener>
    <listener-class>com.example.MyListenerClass</listener-class>
</listener>
```

Demo 1 – ServletConextListener

```
@WebListener
public class MyContextListener implements ServletContextListener {
    @Override
    public void contextDestroyed(ServletContextEvent arg0) {
        System.out.println("Context Destroyed");
    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
        System.out.println("Context Initialized");
    }
}
```

Demo 2 – HttpSessionListener

```
@WebListener
public class SessionListener implements HttpSessionListener {

@Override
public void sessionCreated(HttpSessionEvent e) {
System.out.println(Instant.now() + ": Session " + e.getSession().getId() + " created.");
}

@Override
public void sessionDestroyed(HttpSessionEvent e) {
System.out.println(Instant.now() + ": Session " + e.getSession().getId() + " destroyed.");
}
}
```

Reading

- `HttpSessionBindingListener` from "*Head First Servlets and JSP, 2nd Edition*"
- https://www.w3schools.com/html/html5_webstorage.asp

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Managing State: Continuum of Awareness

1. State information can be stored on the server in the session, and on the browser with cookies.

2. Sessions are managed by the container, which typically use cookies to store session identifiers on browsers.

3.Transcendental consciousness is the direct experience of the unified field. This experience brings orderliness and removes impurities and stresses from memory and thought, resulting in broader awareness and more powerful thoughts.

4. **Impulses within the Transcendental Field:** When one's awareness is connected to the transcendental field then even dynamic focused thoughts have the spontaneous support of the powerful unbounded awareness of pure consciousness.

5. **Wholeness moving within itself:** In unity consciousness, one not only experiences a continuum of unbounded awareness, pure consciousness, but one also appreciates all external entities as expressions of this continuum of awareness.