Maintaining State

Greater Success with Greater Breadth of Awareness

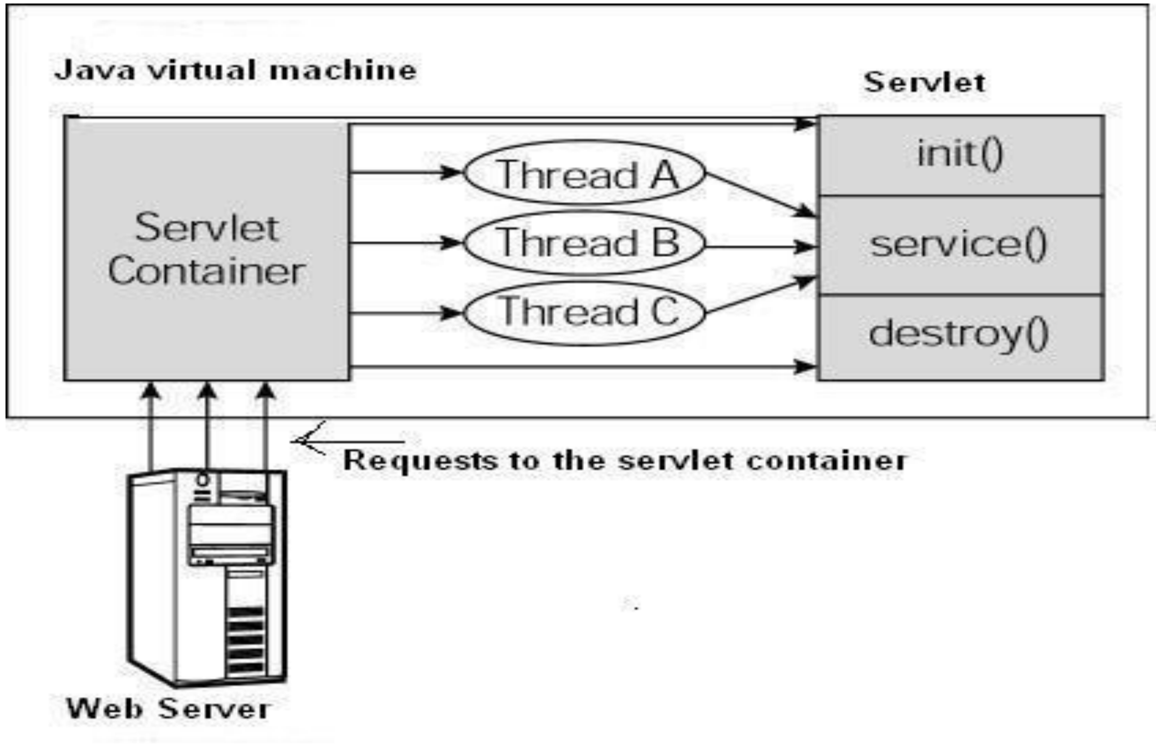
Main point 1 Preview

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: Input parameters are an important communication mechanism between entities. Clearer awareness results in better communication between ourselves and others.

Servlet Lifecycle

- The right figure depicts a typical servlet life-cycle scenario.
- The HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- 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.

ServletConfig vs ServletContext

- ServletConfig
  - One ServletConfig object per Servlet
  - Use it to pass deploy-time information to the servlet
  - Parameters are configured in DD
  - Use it to access the ServletContext
- ServletContext
  - One ServletContext per web app (should name AppContext)
  - Access web app params
  - Get Server info, name, version of Container
- See example: initparams

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
  - GET requests cached for efficiency

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**

  request.getRequestDispatcher("result.jsp").forward(request, response);

  - 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**

  response.sendRedirect("http://www.cs.mum.edu");

  - 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

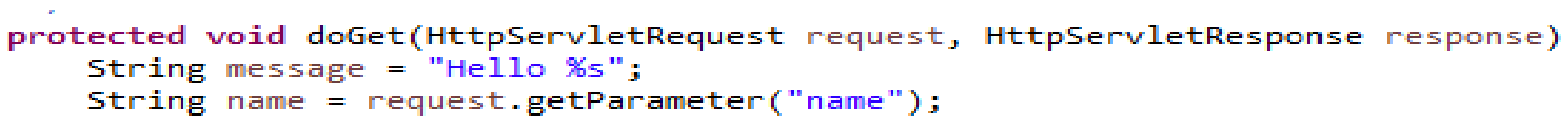
See example: forwardredirect

HTTP input parameters: request.getParameter(…)

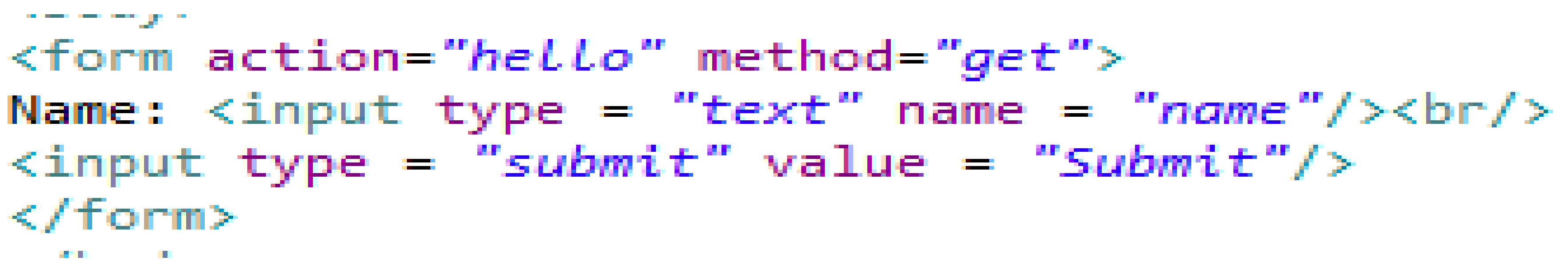
- send data from your HTML page to the servlet
- HTML
  - `<input name="userName " type="text" />`
- HTTP
  - GET /somepath/myservlet?userName=Fred HTTP/1.1
- Servlet
  - `String input = request.getParameter("userName");`
  - What if multiple values?
    - multiple selection list or check box.
  - GET /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.

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>
```

Main point 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: Input parameters are an important communication mechanism between entities. Clearer awareness results in better communication between ourselves and others.

Main point 2 Preview

Attributes are objects on the server. They promote communication and integration between components.

Science of Consciousness: Our experience of transcending facilitates coherence and communication between different components of our brains and minds. Coherence and communication are critical to successful thought and action.

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 available for that request
  - E.g., in a different servlet or a JSP page
  - request. setAttribute(“myAttr”, test);
  - request. getAttribute(“myAttr”);

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

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
  - Not thread safe
    - Nor session attributes
    - Only request attributes thread safe
    - Any updates should be synchronized
- See example: scopedemo

Main point 2

Attributes are objects on the server. They promote communication and integration between components.

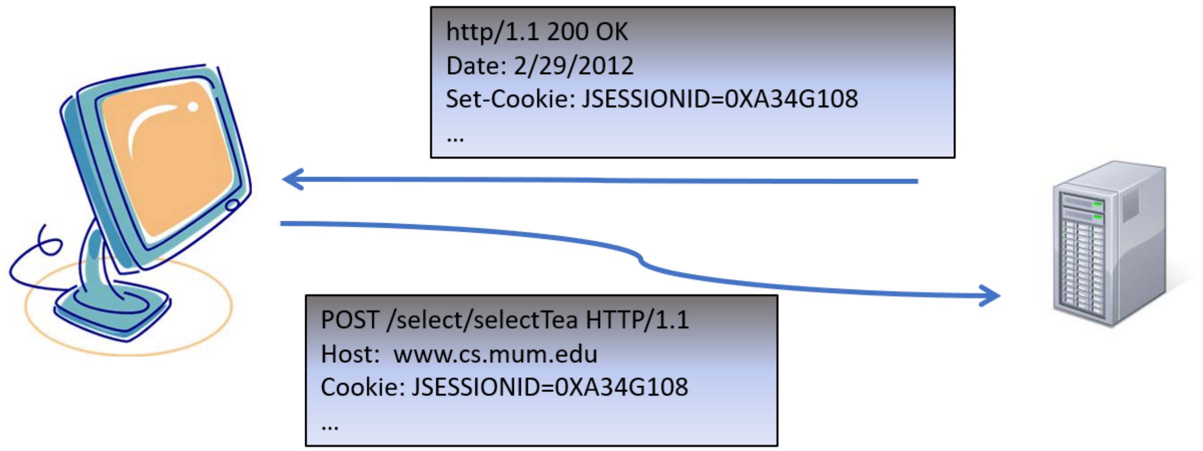
Science of Consciousness: Our experience of transcending facilitates coherence and communication between different components of our brains and minds. Coherence communication is critical to successful thought and action.

Main point 3 Preview

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



```
http/1.1 200 OK
Date: 2/29/2012
Set-Cookie: JSESSIONID=0XA34G108
...
```

```
POST /select/selectTea HTTP/1.1
Host:  www.cs.mum.edu
Cookie: JSESSIONID=0XA34G108
...
```

- A web conversation, holds information across multiple requests.
- before container responds, saves the session info to some datastore
  - then sends an HTTP "cookie" with session id
  - Set-cookie header
- browser saves cookie
  - Places in Cookie header with the next request
  - All cookies from a given website returned when browser sends any request to that website
    - Suppose have multiple tabs open to the same website
- container reconstitutes session from storage before calling servlet

Example cookie exchange

- browser request for www.example.org homepage:

```
GET /index.html HTTP/1.1
Host: www.example.org
...
```

- server responds with two Set-Cookie headers:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

- browser sends another request to visit the spec.html page:

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123
```

Cross-site request forgery (CSRF)

- Confidential information should never be stored in HTTP Cookies

- Suppose you are on a site that takes user comments and they are not filtering HTML inputs.
  - Someone might insert an image link that is actually a request for money to be moved from your account.
  - If you are logged into your account in another tab then this request would go to your bank with your session cookie
  - (Your bank should have a lot more security and safeguards for sensitive actions)

```
<img src=http://mybank.com/withdraw?account=bob&amount=10000&for=sam alt="missing picture">
```

Session lifetime

Client side

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

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

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

(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(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();
  }
}
```

- RFC2109: cookies are part of the HTTP specifications
  - specifies a way to create a stateful session with HTTP requests and responses.
  - describes two new headers, Cookie and Set-Cookie

Maintaining State Demo

6 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
6. URL Rewriting

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.

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.