Advanced Java

Agenda - State management

- Cookie
- Session
- Request
- ServletContext (application)
- QueryString
- Hidden Fields

State Management

- HTTP is stateless protocol.
- State management is maintaining information of the client.
- Client side state management
  - Cookie
  - QueryString
  - Hidden form fields
  - HTML5 storage (SessionStorage and LocalStorage)
- Server side state management
  - Session
  - ServletContext
  - Request

Cookie

- Cookie is a text information in form of key-value pair maintained at the client (browser).
- Server creates a cookie and send to the client in a response.

```
Cookie c = new Cookie("key", "value");
resp.addCookie(c);
```

- Thereafter with each request client send that cookie back to the server.

```
Cookie[] arr = req.getCookies();
for(Cookie c:arr) {
    if(c.getName().equals("key")) {
        String value = c.getValue();
        // ...
    }
}
```

- Temporary cookies
  - Cookies are stored in browser memory. By default, cookies are destroyed when browser is closed.
- Persistent cookies
  - Server can set expiry date for the cookie. Such cookies are stored on client machine (disk) until expiry time.

    ```
    Cookie c = new Cookie("key", "value");
    c.setMaxAge(seconds);
    resp.addCookie(c);
    ```

  - Such cookie is accessible even after browser is restarted.
  - Such cookie can be destroyed forcibly by setting max age = -1.

    ```
    Cookie c = new Cookie("key", "value");
    c.setMaxAge(-1);
    resp.addCookie(c);
    ```

- Limitations/Drawbacks
  - Cookies are stored on client machine. So they are visible to client. Never store sensitive information into cookies.
  - Clients may delete/tamper the cookies (using browser plugins).
  - Cookie max size is 4 KB. Also sending cookie in each request consumes bandwidth.

Session

- https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html
- Http Session is used to save data/state of user on server side in form of key-value pair.
- One session is created for each user/client for first call to req.getSession(). The sub-sequent calls returns the same session object (for that user).

```
HttpSession session = req.getSession();
// HttpSession getSession();
```

- The data can stored in session as attributes -- (String)key-value(Object) pairs.

```
session.setAttribute("key", value);
// void setAttribute(String key, Object value);
```

- This data can be retrieved back (from same user session).

```
value = session.getAttribute("key");
// Object getAttribute(String key);
```

- The session data can be destroyed while logout.

```
session.invalidate();
// void invalidate();
```

- `HttpSession session = req.getSession();`
    - Check if JSESSIONID cookie is present in current request. If present, then get the client's HttpSession (from server's internal map) and return it.
    - Check if JSESSIONID cookie is not present in current request (i.e. req.getSession() is called first time for that client), it creates a new session with a new session id. It add that sessionid into server's internal session map. Then it sends the sessionid to the client in form of a cookie JSESSIONID.
- Session configuration
    - Session tracking and other details can be configured in web.xml.

```
<session-config>
    <session-timeout>30</session-timeout>
    <cookie-config>
        <name>JSESSIONID</name>
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

- Session tracking: Which HttpSession belongs to which client. It can be tracked by session id maintained in "cookie" or "url".
- If `<tracking-mode>COOKIE</tracking-mode>`, then by a default temporary cookie of name JSESSIONID is sent to the client containing session id when session is created. (as mentioned in req.getSession()).
- If `<tracking-mode>URL</tracking-mode>`, then session id is maintained in the URL by "URL rewriting". Example: `http://localhost:8080/bookshop1/subjects` is rewritten as `http://localhost:8080/bookshop1/subjects;JSESSIONID=3984732984092340923409`. Programmer should use resp.encodeURL() or resp.encodeRedirectURL() for url rewriting.

```
String encUrl = resp.encodeRedirectURL("subjects");
resp.sendRedirect(encUrl);
```

```
String encUrl = resp.encodeURL("addcart");
RequestDispatcher rd = req.getRequestDispatcher(encUrl);
rd.forward(req, resp);
```

```
String encUrl = resp.encodeURL("showcart");
out.println("<a href='"+encUrl+"'>Show Cart</a>");
```

```
String encUrl = resp.encodeURL("books");
out.println("<form method='post' action='"+encUrl+"'>");
```

- Session timeout can also be configured in web.xml. If session is not used for the given time (in minutes), the session gets invalidated.

Request

- Some data can be transferred from a servlet to another when forwarding (or including) current request.
- The request object holds map of attributes.
- To add data into request attributes.

```
req.setAttribute("key", value);
```

- To retrieve data from request attributes.

```
value = req.getAttribute("key");
```

- When response is generated, the request and response objects are destroyed (by web server) and hence all request attributes are lost (if saved).

Request parameter vs Request attribute

- Request parameter represents the data coming from the client along with http request (from HTML form controls or query string). It is accessed using req.getParameter() or req.getParameterValues(). Request param are always String.
- Request attributes are added by one web component and forwarded to the next web component. This server side state management is done using req.setAttribute() and req.getAttribute(). Request attribute can be of any type (Object).

ServletContext

- Web server creates a ServletContext object for each web application. It represents the whole "application".
- We can access current application's servlet context by several ways

```
ctx = req.getServletContext();
// OR
ctx = session.getServletContext();
// OR
ctx = config.getServletContext(); // ServletConfig
// OR
ctx = this.getServletContext(); // current servlet
```

- It keeps application metadata/information.
- It can also store state in form of ServletContext attributes (String-Object key-value).

```
ctx.setAttribute("key", value);
```

```
value = ctx.getAttribute("key");
```

- Context attributes are accessible in every request to every web component from every client/user.
- Servlet context can also be used to access context parameters of the application (from web.xml).

```xml
<context-param>
    <param-name>app.title</param-name>
    <param-value>Online Book Store</param-value>
</context-param>
<context-param>
    <param-name>color</param-name>
    <param-value>pink</param-value>
</context-param>
```

```java
String paramValue = ctx.getInitParameter("app.title");
out.println("<h3>" + paramValue + "</h3>");
```

```java
String color = ctx.getInitParameter("color");
out.printf("<body bgcolor='%s'>\r\n", color);
```

QueryString

- Data can be added into URL after '?' in key=value pairs to send along with the request to that URL.
- If there are multiple key-value pairs, they should be separated by &.
- Examples

```html
<a href='url?key1=value1&key2=value2'>Link</a>
```

```
out.printf("<a href='url?key1=%s&key2=%s'>Link</a>", value1, value2);
```

```
out.printf("<form action='url?key1=%s&key2=%s'>", value1, value2);
```

```
String url = String.format("url?key1=%s&key2=%s", value1, value2);
resp.sendRedirect(url);
```

- In next servlet (of given url) this data can be accessed using req.getParameter().

```
String value1 = req.getParameter("key1");
String value2 = req.getParameter("key2");
```

Hidden Form Fields

- Some data can be added into HTML form, which is to be submitted back to the server, but not to render on HTML page (in browser).
- Such data should be added as hidden form field.

```
<input type="hidden" name="key" value="value"/>
```

- When form is submitted, in the servlet this data can be accessed just like other input controls.

```
String value = req.getParameter("key");
```