Enterprise Java

Agenda

- Servlet Life Cycle - Revision
- Request parameters
- Servlet Communication
- State Management
    - Cookie

Servlets

Servlet config

Init parameters

- ServletConfig may have some configurable values like JDBC url, username, password, etc.
- They can be attached to config using init-params.

```
@WebServlet(value="/hi",
    initParams = {
        @WebInitParam(name="color", value="green"),
        @WebInitParam(name="greeting", value="Hi")
    },
    name = "DAC")
public class DacServlet extends HttpServlet {
    // ...
}
```

- These init params can be accessed in servlet class using getInitParameter() method.

```
ServletConfig cfg = this.getServletConfig();
String color = cfg.getInitParameter("color"); // returns "green"
```

```
String message = this.getInitParameter("greeting"); // returns "hi"
```

Load On Startup

- By default servlet is loaded and initialized on first request. If init() includes heavy processing, the first request will execute slower.
- Alternatively servlets can be loaded while starting the web server. This can be done by marking servlet as load-on-startup.

```
@WebServlet(value="/hi",
    loadOnStartup = 1,
    name = "DMC")
public class DmcServlet extends HttpServlet {
    // ...
}
```

- The number after "loadOnStartup" indicate the sequence of loading the servlets if multiple servlets are marked as load-on-startup. If multiple servlets load-on-startup number is same, web container arbitrarily choose the sequence.
- Servlet config in web.xml

```
<servlet>
    <servlet-name>DAC</servlet-name>
    <servlet-class>com.sunbeam.DacServlet</servlet-class>
    <init-param>
        <param-name>color</param-name>
        <param-value>pink</param-value>
    </init-param>
    <init-param>
        <param-name>greeting</param-name>
        <param-value>Good Afternoon</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>DAC</servlet-name>
    <url-pattern>/hi</url-pattern>
</servlet-mapping>
```

Servlet communication/navigation

- HTTP redirection
  - resp.sendRedirect("url");
  - Can navigate from one web component to another web component (within or outside the current application).
  - resp.sendRedirect() sends a minimal response to the client which contain status code 302 and location (url) of next web component.
  - The client (browser) receives this response and send new request to the next web component.
  - In browser, URL is modified (i.e. client is aware of navigation).
- RequestDispatcher
  - https://docs.oracle.com/javaee/7/api/javax/servlet/RequestDispatcher.html
  - RequestDispatcher rd = req.getRequestDispatcher("url");
    - url is w.r.t. current request.
  - RequestDispatcher rd = ctx.getRequestDispatcher("/url");
    - url is w.r.t. application (context) root.

- RequestDispatcher – forward()
  - rd.forward(req, resp);
  - Forwards the current request to the given web component (within application only).
  - The next web component produces final response (to be sent to the client).
  - Note that new request & response objects are not created.
  - In browser, URL is not modified (i.e. client is not aware of navigation).
  - Faster than HTTP redirection.
  - Used in Spring MVC by the controller.
- RequestDispatcher – include()
  - rd.include(req, resp);
  - Calling given web component (within application only) to produce partial response.
  - The final response is generated by the current (first) web component itself.
  - Note that new request & response objects are not created.
  - In browser, URL is not modified (i.e. client is not aware of navigation).
  - Slower than RequestDispatcher – forward().
  - Mostly used for rendering header/footer in dynamic web pages.

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.