

Advanced Java

Servlets

web.xml -- Servlet config

- Servlet config in web.xml

```
<servlet>
  <servlet-name>DMC</servlet-name>
  <servlet-class>com.sunbeam.DmcServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DMC</servlet-name>
  <url-pattern>/mobile</url-pattern>
</servlet-mapping>
```

Servlet hierarchy

- javax.servlet.Servlet interface
 - void init(ServletConfig config) throws ServletException;
 - void service(ServletRequest req, ServletResponse resp) throws IOException, ServletException;
 - void destroy();
- GenericServlet is abstract class that represents protocol-independent servlet.
- HttpServlet represent http based servlet class and user defined servlet classes are inherited from it.
 - Overrides service() method.
 - Provide doGet(), doPost(), doPut(), doDelete(), doHead(), doTrace(), doOptions()
 - Docs: <https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>

Servlet Life Cycle

- Refer slides

ServletConfig

- Each servlet is associated with a config object -- ServletConfig.
- It stores information about servlet like name, init parameters, url-patterns, load-on-startup, etc.
- This can be accessed in the servlet class in init() method (as argument) or other methods using `ServletConfig cfg = this.getServletConfig();`.
- Note that all servlet classes are indirectly inherited from ServletConfig, so ServletConfig methods are directly available on servlet object (this).

Init parameters

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

```
<servlet>
  <servlet-name>DMC</servlet-name>
  <servlet-class>com.sunbeam.DmcServlet</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>DMC</servlet-name>
  <url-pattern>/mobile</url-pattern>
</servlet-mapping>
```

- 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"
```

Servlet Exception Handling

- `void init(ServletConfig config)` throws `ServletException`;
 - If initialization of servlet fails, it should throw `ServletException`; so that web-server will stop further processing of the servlet.
- `void service(ServletRequest req, ServletResponse resp)` throws `ServletException`, `IOException`;
 - If `service()` method fails, it should throw `ServletException` or `IOException`.
 - In this case, web server generates a default response indicating 500 - internal server error.
- code example:

```
public class DbServlet extends HttpServlet {  
    public void init(ServletConfig config) throws ServletException {  
        super.init(config); // attaches given config to current servlet "this" object  
        try {  
            // JDBC connection code e.g. DriverManager.getConnection();  
        } catch(SQLException ex) {  
            throw new ServletException(ex); // exception chaining  
        }  
    }  
    // ...  
}
```

Servlet Init parameters

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

```
@WebServlet(value="/mobile",
    initParams = {
        @WebInitParam(name="color", value="green"),
        @WebInitParam(name="greeting", value="Hi")
    },
    name = "DMC")
public class DmcServlet 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 using web.xml or annotation.

```
@WebServlet(value="/mobile",
    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.
- If number after "loadOnStartup" is negative, the servlet is not loaded at startup. It will be loaded on first request.

web.xml

- web.xml is deployment descriptor of web applications. It contains deployment information like servlet configs, jsp configs, session timeout, application security, etc.
- Servlet config in web.xml

```
<servlet>
  <servlet-name>DMC</servlet-name>
  <servlet-class>com.sunbeam.DmcServlet</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>DMC</servlet-name>
```

```
<url-pattern>/mobile</url-pattern>  
</servlet-mapping>
```

HttpServletRequest

- Request Parameters
 - req.getParameter()
 - req.getParameterValues()
- Request Headers
 - req.getHeader()
 - req.getHeaderValues()
- Request upload
 - req.openInputStream()
- <https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html>

HttpServletResponse

- Response content type
 - resp.setContentType()
- Response download
 - resp.openOutputStream()
- <https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html>

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).

- Used to implement PRG pattern (Post-Redirect-Get).
- 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.

Assignment

1. Movie Review System -- Hackathon.