Managing Data on the World Wide-Web

Sessions, Listeners, Filters, Shopping Cart

Elad Kravi
Web Applications

- In the Java EE platform, web components provide the dynamic extension capabilities for a web server
  - Web components for example: Java servlets, JSP pages
• Web components share information by means of objects that are maintained as attributes of four scope objects
  – We can access these attributes by using the `getAttribute` and `setAttribute` methods of the class representing the scope
# Scope Objects

<table>
<thead>
<tr>
<th>Scope Object</th>
<th>Class</th>
<th>Accessible From</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web context</td>
<td>javax.servlet.ServletContext</td>
<td>Web components within a web context</td>
</tr>
<tr>
<td>Session</td>
<td>javax.servlet.http.HttpSession</td>
<td>Web components handling a request that belongs to the session</td>
</tr>
<tr>
<td>Request</td>
<td>Subtype of javax.servlet.ServletRequest</td>
<td>Web components handling the request</td>
</tr>
<tr>
<td>Page</td>
<td>javax.servlet.jsp.JspContext</td>
<td>The JSP page that creates the object</td>
</tr>
</tbody>
</table>
Overview

1. Sessions
2. Listeners
3. Filters
4. Shopping Cart
• Many applications require that a series of requests from a client be associated with one another
  – For example, a web application can save the state of a user's shopping cart across requests

• Because HTTP is stateless, web-based applications are responsible for maintaining such state, called a session
Maintaining Sessions with Cookies

1. The client sends a request to the server

2. The server generates a new session id

3. The response from the server contains a directive to the browser to store a cookie saying the session id is 3

Does the server keep the session object forever?

The following requests from the client will include session id 3 and then can be associated with one another by the server.
Java’s servlets provide an API for managing sessions.

Sessions are represented by an HttpSession object.

The getSession() method returns the current session associated with a request object.

- If the request does not have a session, it creates one.
```java
protected void doGet(HttpServletRequest request, HttpServletResponse response) 
    throws ServletException, IOException {

    HttpSession session = request.getSession();

    List<String> shoppingCart =
        List<String>) session.getAttribute("cart");

    if (shoppingCart == null) {
        shoppingCart = new ArrayList<String>();
        session.setAttribute("cart", shoppingCart);
    }

    shoppingCart.add(request.getParameter("item"));

    response.getWriter().append("My cart: ")
        .append(shoppingCart.toString());
}
```
HttpSession - Methods

**getAttribute(String name)**
- Returns the object bound with the specified name in this session

**setAttribute(String name, Object value)**
- Binds an object to this session, using the name specified

**removeAttribute(String name)**
- Removes the object bound with the specified name from this session

**invalidate()**
- Invalidates this session
Due to inactivity, sessions can be invalidated by the server

- **Programmatically:**
  ```java
  session.setMaxInactiveInterval(int seconds)
  ```

- **Declaratively (web.xml)**
  ```xml
  <session-config>
    <session-timeout>180</session-timeout><!--minutes-->
  </session-config>
  ```

- **Negative value means the session will never expire**
Overview

1. Sessions
2. Listeners
3. Filters
4. Shopping Cart
Listeners

- Servlet API provides a way to track key events in your Web applications through **event listeners**
  - Examples of such events: initialization/shutdown of the application, creation of a session

- This functionality allows more efficient resource management and automated processing based on event status
Event Levels and Types

- There are two levels of servlet events:
  - Servlet context / application - level event
  - Session-level event

- Each of these two levels has two event categories:
  - Lifecycle changes (e.g., session invalidation)
  - Attribute changes (e.g., addition of servlet context attributes)
Common Listener Types

- For each type of a listener there is a corresponding Java interface
  - `ServletContextListener`
    - web application initialized / shut down
  - `ServletRequestListener`
    - request handler starting / finishing
  - ` HttpSessionListener`
    - session created / invalidated
  - ` ServletContextAttributeListener`
    - context attribute added / removed / replaced
  - ` HttpSessionAttributeListener`
    - session attribute added / removed / replaced
We can create a listener to manage database connections

- When the application is initialized, the listener logs in to the database and stores the connection object in the servlet context
- Prior to application shutdown, the listener closes the database connection
• Listeners can be registered in web.xml or by using the `@WebListener` annotation (Servlet 3.0)

Registration in `web.xml`:

```xml
<listener>
    <listener-class>SessionMonitor</listener-class>
<listener>
```
• In the next slides we will see an example of a listener that monitors the current and maximum number of active sessions
  – Our class implements both **HttpSessionListener** and **ServletContextListener**
  – That is, the listener is notified when the application is initialized and when a session is created or invalidated
@WebListener
def public class SessionMonitor implements HttpSessionListener, ServletContextListener {

    public void contextInitialized(ServletContextEvent ctxEvent) {
        ServletContext ctx = ctxEvent.getServletContext();
        ctx.setAttribute("sessions_active", 0);
        ctx.setAttribute("sessions_max", 0);
    }

    public void contextDestroyed(ServletContextEvent ctxEvent) {}

    ...
}

... (next slide)
public void sessionCreated(HttpSessionEvent sessionEvent) {
    ServletContext ctx = sessionEvent.getSession().getServletContext();
    int active = (int) ctx.getAttribute("sessions_active");
    int max = (int) ctx.getAttribute("sessions_max");

    active++;
    if (active > max) {
        max = active;
        ctx.setAttribute("sessions_max", max);
    }

    ctx.setAttribute("sessions_active", active);
}

public void sessionDestroyed(HttpSessionEvent sessionEvent) {
    ServletContext ctx = sessionEvent.getSession().getServletContext();
    int active = (int) ctx.getAttribute("sessions_active");

    active--;
    ctx.setAttribute("sessions_active", active);
}
Overview

1. Sessions
2. Listeners
3. Filters
4. Shopping Cart
Filters

- A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource (or both)

- The main tasks of filters:
  - Query the request and act accordingly
  - Block the request/response from passing any further
  - Modify the request/response headers and data
  - Interact with external resources
Filters - Examples

- Authentication Filters
- Logging and Auditing Filters
- Image conversion Filters
- Data compression Filters
- Encryption Filters
- Tokenizing Filters
- Filters that trigger resource access events
- XSL/T filters
- Mime-type chain Filter
The \texttt{@WebFilter} annotation is used to define a filter in a web application

- Classes annotated with this annotation must implement the \texttt{javax.servlet.Filter} interface

Filters perform filtering in the \texttt{doFilter} method

\begin{verbatim}
   doFilter(ServletRequest request, 
            ServletResponse response, 
            FilterChain chain)
\end{verbatim}

The \texttt{FilterChain} object allows the Filter to pass on the request and response to the next filter in the chain
@WebFilter(
    filterName = "LogFilter",
    urlPatterns = {"/*"},
    initParams = {
        @WebInitParam(name = "param1", value = "some value")
    }
)

public class LogFilter implements Filter {

    public void init(FilterConfig config) throws ServletException {
        // Get init parameter and log it
        String param1 = config.getInitParameter("param1");
        System.out.println("Test Param: " + param1);
    }

    public void destroy() {
        // add code to release any resource
    }

    ... (next slide)
public void doFilter(HttpServletRequest req,
            HttpServletResponse res, FilterChain chain)
    throws IOException, ServletException {
    HttpServletRequest request = (HttpServletRequest) req;

    // Get the IP address of client machine and log it.
    String ipAddress = request.getRemoteAddr();
    System.out.println("IP "+ ipAddress
                      + ", Time "+ new Date().toString());

    // Continue calling the rest of the filters and
    // eventually the servlet
    chain.doFilter(req, res);
}
Filter Chain

• A web resource can be filtered by a chain of zero, one, or more filters in a specific order
  – To define execution order of filters, the filters must be declared by using web.xml (not possible with annotations only)

• A FilterChain is an object provided by the servlet container to the developer

• The developer is then provided with a view into the invocation chain which he can utilize for his needs
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

    // do pre-servlet work here

    // Continue calling the rest of the filters and eventually the servlet
    chain.doFilter(request, response);

    // do post servlet work here
}
Filter Configuration in web.xml

```xml
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>mypackage.LogFilter</filter-class>
  <init-param>
    <!-- optional -->
    <param-name>param1</param-name>
    <param-value>some value</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

The execution order of filters is determined by the order of the filter-mapping elements in web.xml
Filters or Listeners?

- Filters are used for monitoring/modifying requests/responses
  - Use a Filter if you want to intercept on HTTP requests matching a specific URL pattern

- Listeners are used for listening to events in a web containers
  - Use a Listener if you want to intercept changes in the lifecycle of objects
Overview

1. Sessions
2. Listeners
3. Filters
4. Shopping Cart
We will now analyze a web application implementing shopping cart.

Requirements:
- add/remove single product to/from the cart
- remove all the products from the cart
- view cart state
- future: start with some initial products (think about weekly shopping list in the grocery store)
Server creates new session for each new client

The client can call add/remove/remove all on server
– in each request the client attaches his id in a cookie

Server identifies the client by the cookie and updates his cart
– how can the cart be saved?

The client can view the cart’s content at anytime

In this solution cart is attached to session and not to an account
@WebServlet("/ShoppingCart")
public class ShoppingCart extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @SuppressWarnings("unchecked")
    private Collection<String> getItems(HttpSession session) {
        Collection<String> items =
            (Collection<String>)session.getAttribute("items");
        return items != null ? items : new ArrayList<String>();
    }

    private String createHtmlItemsTable(Iterable<String> items) {
        StringBuilder builder = new StringBuilder("<table>");
        for (String item : items) {
            builder.append("<tr><td>");
            builder.append(item);
            builder.append("</td></tr>");
        }
        builder.append("</table>");
        return builder.toString();
    }

    ... (next page)
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    String item = request.getParameter("item");

    HttpSession session = request.getSession();
    Collection<String> items = getItems(session);
    items.add(item);
    session.setAttribute("items", items);

    response.getOutputStream().println("<!DOCTYPE html PUBLIC -//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
    "<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1255">
    "<title>Shopping Cart</title>
    "</head>
    "<body>"

    "<h1>Your Shopping CartConstains:<h1>" +
    createHtmlItemsTable(items) +

    "</body>" +
    "</html>"

    }
Pros. & Cons. of Session approach

• Pros.
  – easy to implement
  – widely supported

• Cons.
  – how many sessions can we save?
    • does that limit number of clients we can serve?
    • does sessions have timeout? what happens if a user doesn’t use his browser for an hour? a week?
    • what happens if the client’s browser crashes?
    • reset the cookie after login
  – can we start with half full cart?
Saving the Cart

• Where should the cart be saved?
  – in memory?
    • how many clients we can serve?
    • what if we want to distribute the server?
    • what about same client connecting from different browsers simultaneously?
  – in DB?
    • latency
    • handling scale is complex
After authentication phase, the server can identify the user by a cookie sent with a request

Cart state is represented by URL only
- `/cart/item/5/item/12`
- adding/removing item, simply by changing url
  - `/cart/item/5`  /`cart/item/5/item/12/item/13`
  - removing all items: `/cart/
- browsing while saving the state
  - `/shop/dairy/cart/item/5/item/12`
- starting with half full cart – just by changing the url
Pros. & Cons. of REST approach

• Pros.
  – client can start from scratch in new browser
    • just copy the url
  – easy and flexible change cart state

• Cons.
  – does /cart/item/5/item/12 and /cart/item/12/item/5 refers to the same cart?
    • we can sort the items by id
  – complex URLs
    • think about cart with 100 products
References

• The Java EE Tutorial: https://docs.oracle.com/javaee/7/tutorial/partwebtier.htm