2.8. Session management

Juan M. Gimeno, Josep M. Ribó

January, 2008
Session management. Contents

- Motivation
- Hidden fields
- URL rewriting
- Cookies
- Session management with the Servlet/JSP API
- Examples
- Scopes
**Motivation: HTTP: A stateless protocol**

When a client sends a request, the server sends back a response but does not keep any information about that request and the client state.

In most web applications a client has to access various pages before completing a specific task and the client state should be kept along all those pages

- **Example 1:** A customer wants to buy some items in an on-line store. She/he should add each one of them to his/her shopping cart and, at the end, pay for them in order to complete the purchase
- **Example 2:** A customer wants to make a reservation for a specific performance: he/she should choose the performance, the city, the theater in which it takes place, the schedule, the seat and, finally buy one or more tickets for that specific performance

Both examples involve a conversation between the customer and the web application (at the server side).

The server must maintain, at every moment, the client’s **conversational state** (i.e., the information that the client has sent to the server along his/her interaction with the web application)

How to remember the client’s conversational state using a stateless protocol?
## Keeping client information between pages

In a web application we may maintain the client state by means of different techniques:

- Hidden fields
- URL rewriting
- Cookies
- Servlet and JSP API
Hidden fields

Idea:
The client state is passed from the server to the client and back to the server in a hidden field of a form

```jsp
<% int n; if (request.getParameter("counter") == null) n = 1; else{ n = Integer.parseInt(request.getParameter("counter")) + 1; } %>
<html>
<h1>Page counter using hidden fields</h1>
<form action="index.jsp" method="get" >
This is your access number <%=n%> <p>
<input type="hidden" name="counter"
value="<%=String.valueOf(n)%>"
<input type="submit" value="New access">
</form>
</html>
```

Example location: sessions/examples/ex1
Idea:
The client state is passed from the server to the client and back to the server in the query string, accompanying the URL

```jsp
<% 
int n; 
if (request.getParameter("counter") == null) n = 1; 
else{ 
    n = Integer.parseInt(request.getParameter("counter")) + 1; 
} 
%>
<html>
<h1>Page counter using URL rewriting</h1>
This is your access number <%=n%>
<p>
<a href="index.jsp?counter=<%=String.valueOf(n)%>">New access</a>
</p>
</html>
```

Example location: sessions/examples/ex2
Cookies

A cookie is a pair (key-string, value-string) which is:

- Sent by the server to the client as a part of the response to a client request,
- Stored by the client browser (when it receives the cookie from the server) and
- Sent back by the client as a part of subsequent requests to the same server

Example of cookie: (accessNb, "12")
Cookies can be used to store the state reached by the client in the previous request so that it can be restored in the next one.

**Example:**

The cookie \((\text{accessNb}, "12")\) may mean that as a consequence of the last request (access to a page \(p\)) the number of accesses to \(p\) has been 12.

When the client sends another request to access \(p\) the server will increase that number by one and will send the new cookie \((\text{accessNb}, "13")\) to the client.
Cookies features

- Cookies are stored in the client browser as pairs (string-key, string-value). Each pair is associated to a URL domain.

- Cookies have an expiration date. After that date, the client state will be lost.

- Cookies are not a universal way to store the client state (e.g., the client may disable cookies).

- Cookies are usually used in two situations:
  - Remember client preferences (e.g., client language)
  - Support to session management (see below)
Cookies. API

Java provides support for cookies. In particular:

- It defines the class Cookie

- It provides operations to get the cookies from the client request (`request.getCookies()`) and to add cookies to the server response (`response.addCookie(...)`)
Cookies. API. The class Cookie

Summary of the Cookie class operations

- `Cookie(java.lang.String name, java.lang.String value)`
  Constructs a cookie with a specified name and value.
- `java.lang.String getName()`
  Returns the name of the cookie.
- `java.lang.String getName()`
  Returns the name of the cookie.
- `java.lang.String getPath()`
  Returns the path on the server to which the browser returns this cookie.
- `java.lang.String getValue()`
  Returns the value of the cookie.
- `void setMaxAge(int expiry)`
  Sets the maximum age of the cookie in seconds.
- `void setPath(java.lang.String uri)`
  Specifies a path for the cookie to which the client should return the cookie.
- `void setValue(java.lang.String newValue)`
  Assigns a new value to a cookie after the cookie is created.
Cookies. API. Cookies support in other classes

Interface HttpServletRequest

- Cookie[] getCookies()
  
  Returns an array containing all of the Cookie objects the client sent with this request.

Interface HttpServletResponse

- void addCookie(Cookie cookie)
  
  Adds the specified cookie to the response.
Cookies. An access counter

Example location: sessions/examples/ex3

File index.jsp

```jsp
<%  
final int YEARSECS=60*60*24*365;  
int n;  
Cookie[] cookies=request.getCookies();  
String accessnb=null;  
if (cookies!=null){  
   for (int i=0; i<cookies.length;i++){  
      Cookie c=cookies[i];  
      if ((c.getName()).equals("accessNb"))  
         accessnb=c.getValue();  
   }  
}  
if (accessnb==null){  
   n=1;  
}  
else{  
   n=Integer.parseInt(accessnb);  
}  
Cookie c=new Cookie("accessNb",String.valueOf(n+1));  
c.setMaxAge(YEARSECS);  
response.addCookie(c);  
%>
```

.............
Cookies. An access counter (2)

This is a version without session support. A cookie is used to store (in the client side) the value of the last access number.

This is your access number <%=n%>

Click <a href="index.jsp"> here </a>

```html
<html>
<h2>Access counter</h2>
<p>This is a version without session support. A cookie is used to store (in the client side) the value of the last access number</p>
<p>This is your access number <%=n%></p>
<p>Click <a href="index.jsp"> here </a></p>
</html>`
Idea of sessions:

1. **The client sends a request to the server**

2. **The server creates a session** which will encompass all the interactions with that client in the next few minutes. **This session is identified with a session identifier**

3. **The server manages the client request** and elaborates a response to its request.

4. **This management may remember some objects which will be retrieved by later requests of the same session.** These remembered objects:
   - are called attributes
   - are associated to the session
   - can be retrieved using a key (string)
Servlet and JSP API. Session management

5. When the server has finished managing the request, **the server sends to the client:**
   
   • its response (html/xml code)
   • a cookie with the session identifier (**sid**)

6. **When the client sends another request to the same server, it includes the cookie** (with the identifier **sid**) that received from it. In this way the server can identify the session.

7. The server manages the new request. In particular, it may retrieve the session attributes which were set in the last session access.
counter_jsp.class is executed:
*Get the session identifier obtained from the request (sid):
  session=request.getSession();
*Manage the session: Get/Set session attributes:
  session.getAttribute("counter");

http−request: "counter.jsp"

http−response:

Cookie:JSESSIONID=sid;
html page

counter_jsp.class is executed:
*Create a session identifier (sid)
*Create a cookie JSESSIONID=sid
*Add the cookie to the server response
*Manage the request and set session attributes:
  session.setAttribute("counter",new Integer(1));

http−request: "counter.jsp"

http−request:

Cookie:JSESSIONID=sid;
"counter.jsp"

counter_jsp.class is executed:
*Get the session identifier obtained from the request (sid): session=request.getSession();
*Manage the session: Get/Set session attributes:
  session.getAttribute("counter");
The following classes and interfaces are provided by the *Servlet and JSP API* in order to manage sessions:

- `HttpSession`
- `HttpServletRequest`
- `HttpServletResponse`
- `Cookie`

In order to trace objects that are incorporated and/or removed from a session or activation/remove of the own session, the API offers the following classes and interfaces (however, these classes are not presented in these slides):

- `HttpSessionActivationListener`, `HttpSessionAttributeListener`, `HttpSessionListener`, `HttpSessionBindingListener`, `HttpSessionBindingEvent`, ` HttpSessionEvent`
Session management. Use of the API

A servlet/JSP page which manages a request which needs session support must do the following:

1. **Session creation/identification**
   
   Identify an already created session which can be associated to that request (or create a session for that request if none existed)

2. **Attribute management**
   
   Manage session attributes (get/set attributes associated to that session)

3. **Session tracking**
   
   Ensure that the client knows the session identifier so that later requests from the same client may be associated (by the server) to the same session

4. **Session destruction**
Session creation/identification

A client wants to interact with a server (e.g., the client wants to make a reservation in a theatre reservation web application). The client requires session management:

- When the client calls the first servlet/JSP page of the application, the server must create the session.

- When the client calls a posterior servlet/JSP page (after the session creation), the server must retrieve the session that was created previously.

This session is attached to the client request by means of a cookie (or a parameter in the query string) that contains the session identifier.

How is this done?

In the following slides we will see it both for *servlets* and *JSP pages*. 
Session creation/identification. Servlets

The method service(request, response)/doGet(request, response)/doPost(request, response) should incorporate the following sentence:

- HttpSession session=request.getSession();
  Creates a session (with a new identifier) if request does not contain any session identifier.

  Otherwise, it retrieves the session identifier attached to request (by means of a cookie or a parameter in a query string) and uses it to build an object of class HttpSession

- HttpSession session=request.getSession(create);
  If create=true, the same as before
  If create=false, retrieves an already existing session but it does not create a new one if request does not have any session identifier attached to
Session creation/identification. JSP pages

JSP pages that need session management should use the page directive

<%@ page session="true" %>

If this directive is used, then

A JSP page can refer to the session which is associated to it by means of the implicit variable session

In the following slides we explain how this is possible.....
Session creation/identification. JSP pages (2)

(Some aspects of the JSP to servlet translation)

JSP pages are automatically translated into a servlet. This translation consists of the following issues:

- Creation of a context for the JSP page
  This context includes session management

- Request management and response creation
  This management includes the get/set of session attributes

The next slides focus on the creation of a context for a JSP page
Session creation/identification. JSP pages (3)  
(Some aspects of the JSP to servlet translation)

This is a fragment of the automatic translation of a JSP page into a servlet that illustrates how the context of a JSP page is created:

```java
public void _jspService(HttpServletRequest request, 
                        HttpServletResponse response) 
    throws java.io.IOException, ServletException {

    //Variable declaration....... 
    try {
        jspxFactor = JspFactory.getDefaultFactory();

        pageContext = jspxFactor.getPageContext( 
            this, request, response, null, true, 8192, true);
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();

        //... Request management and response creation....
    } catch (Throwable t) {
        //....
    } finally {
        if (jspxFactor != null)
            jspxFactor.releasePageContext(pageContext);
    }
```
Session creation/identification. JSP pages (4)

(Some aspects of the JSP to servlet translation)

```java
JspFactory jspf = JspFactory.getDefaultFactory();
PageContext pageContext =
    jspf.getPageContext(this, request, response,
            null, true, 8192, true);
```

`pageContext` is an object that encapsulates the page context, which includes the following aspects associated to the JSP page:

- The request that has launched this JSP page
  That is, the first parameter of `_jspService`. It can be retrieved by:

  ```java
  request=pageContext.getRequest();
  ```

- The pending response which will be generated as a result of the execution of this JSP page and, afterwards, sent to the client

  ```java
  response=pageContext.getResponse();
  ```
• The output stream that has been associated to this JSP page

```java
out=pageContext.getOut();
```

• The session that is associated to this request

```java
getPageContext(...) calls request.getSession()
```

The session can be retrieved by:

```java
session = pageContext.getSession();
```

• The application to which this JSP page is associated (useful to define application objects. See below)

```java
application = pageContext.getServletContext();
```
Session creation/identification. JSP pages (5)

(Some aspects of the JSP to servlet translation)

As a consequence of this translation, in the JSP page the following implicit variables are available (no need to declare or initialize them):

- request
- response
- out
- pageContext
- session
- application
- config
- page
- exception

Therefore the JSP page can refer to the session which has been associated to it by means of the implicit variable session.
Attribute management

Servlets and JSP pages may remember several objects across different requests within the same session.

- **Remembering an object**
  
  This is done by associating the object which is to be remembered to an attribute name (a key string).
  
  In turn, this attribute name will be associated to the session.

  ```java
  ObjectClass object=new Object(...);
  session.setAttribute("attrName", object);
  ```

- **Retrieving the object**
  
  Another request associated to the same session may retrieve the object associated to the attribute called attrName in the following way:

  ```java
  ObjectClass oc=(ObjectClass) session.getAttribute("attrName");
  ```
Attribute management (2)

Some considerations:

- Only objects can be stored as attributes

  Primitive types (e.g., int, float...) must be wrapped using the *wrapper classes* (e.g., Integer, Float...)

- Different servlets/JSP pages share the same session attributes. Beware of name clash.
Session tracking

In order to identify and manage the session associated to a request:

- The server should send the session identifier to the client in its response
- The client should send the session identifier (obtained from the server) in subsequent requests

This can be done in two different ways:

1. By means of a cookie
2. By adding the session identifier in the query string of the request
Session tracking. Cookies

• When a servlet creates a session by means of `getSession()` or `getSession(true)` it adds a cookie to the response to be sent to the client.

In particular, it adds a header to the response with the following code:

```
Set-Cookie: JSESSIONID=123456789poiuyt
```

• When the client sends subsequent requests to the server, it adds to the request the previous cookie:

```
Cookie: JSESSIONID=123456789poiuyt
```

• When the servlet executes the operation `request.getSession()` for a request that contains a cookie `JSESSIONID=sessionId`, it returns a session with the identifier `sessionId`
Session tracking. Query string

The client may not accept cookies. However, session management should be equally possible, even in this case.

**Idea:**
If the client does not accept cookies, include the session identifier as a parameter of any subsequent request sent by the client.
The parameter will be codified as a query string accompanying the URL of the request.

**Example:**

```html
<a href="counter.jsp;jsessionid=9898988787xcds8">link</a>
```

How can we do this?......
Session tracking. Query string

How can we do this?

The HttpServletResponse interface provides the method:

```
java.lang.String encodeURL(java.lang.String url)
```

This method encodes the url by including the session identifier only in the case that this is necessary (client browser does not accept cookies and the servlet requires session management). It returns the encoded url or the url unchanged (if encoding is not necessary)
Session tracking. Query string

Example in a JSP page:

```html
<a href="<%=response.encodeURL("counter.jsp") %>">
  link </a>
```

The client will receive:

- If it does not accept cookies and session management is enabled for that JSP page:

```html
<a href="http://ingrid.udl.net:8080/counter.jsp?
  jsessionid=9898988787xcds8">
  link </a>
```

- If it accepts cookies and session management is enabled for that JSP page:

```html
<a href="http://ingrid.udl.net:8080/counter.jsp">
  link </a>
```
Session destruction

• Sessions are automatically cancelled (and their resources deallocated) when they reach a certain timeout

  – This timeout is specified in the configuration file web.xml of the server container. In the case of Tomcat:

    ```xml
    <web-app>
    ...
    <session-config>
    <session-timeout> 30 </session-timeout>
    </session-config>
    ...
    </web-app>
    
    In this case, the session timeout is established in 30 minutes

  – It is possible to change the timeout for a specific session using the method `setMaxInactiveInterval(n)` of the interface HttpSession:

    ```java
    session.setMaxInactiveInterval(300);
    
    This states the timeout for this session to 300 seconds (5 minutes).```
Session destruction

- Sessions can be explicitly cancelled by using the method `invalidate()`:

  ```java
  session.invalidate();
  ```
Example 1: A JSP page to count page accesses

Example location: sessions/examples/ex4

```jsp
<%@ page session="true" %>
<html>
<h2>Access counter with session support</h2>
<p>This version relies on cookies. If cookies are disabled in the client, this does not work</p>
<%
    if (session.getAttribute("counter") == null) {
        session.setAttribute("counter", new Integer(1));
    }
    int c = ((Integer) session.getAttribute("counter")).intValue();
%
    This is your access number <%=c%>
<%
    session.setAttribute("counter", new Integer(c+1));
%
    Click <a href="index.jsp">here</a>
</html>
```

Warning: This example only works if the cookies are enabled in the client
Ex. 1: A JSP page to count page accesses

Example location: sessions/examples/ex5

```jsp
<%@ page session="true" %>
<html>
<h2>Access counter with session support</h2>
<b>This version does not need cookies to work</b>
<% 
    if (session.getAttribute("counter") == null) 
        session.setAttribute("counter", new Integer(1)); 

    int c = ((Integer) 
        session.getAttribute("counter")).intValue(); 
%>
This is your access number <%=c%>
<% session.setAttribute("counter", new Integer(c+1)); %> 
<p>
Click
<a href="<%=response.encodeURL("index.jsp") %>">
here</a>
</html>
```

This version works both if the client has the cookies enabled or disabled

Notice the use of:

```jsp
<%=response.encodeURL("index.jsp") %>
```
Example 2: A servlet to count page accesses

Example location: sessions/examples/ex6

```java
import javax.servlet.*;
import java.io.*;
import javax.servlet.http.*;
import java.util.*;

public class Counter extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
                       throws IOException, ServletException {
        doPost( request, response ) ;
    }
}
```

continues.....
public void doPost (HttpServletRequest request, HttpServletResponse response) {
    try {
        response.setContentType("text/html;charset=ISO-8859-1");
        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession();

        //getSession is responsible for including the cookie JSESSIONID to the response

        if (session.getAttribute("counter") == null)
            session.setAttribute("counter", new Integer(1));

        int c = ((Integer)
                    session.getAttribute("counter")).intValue();
Example 2: A servlet to count page accesses

......continues

    out.write("\n\n");
    out.write("<html>\n\n");
    out.write("<h2> Access counter"');
    out.write("</h2>\n\n");
    out.write("\n\nThis is your access number ");
    out.print(c);
    out.write("\n\n");

    session.setAttribute("counter",new Integer(c+1));

    out.write("\n");
    out.write("<p>\nClick \n\n\n");
    out.write(
        "<a href=" +
        "counter">"+
        "here ");
    out.write("</a>\n\n");
    out.write("</html>\n");

    if (c>=5) session.invalidate();

} catch (Exception ex) {   ex.printStackTrace();
}

}
Example 3: A shopping cart

- This example shows a primitive bookstore on-line
- It uses java beans, session management and JDBC.
- It is presented in java beans module (ex. ***)

Sessions. Servlet and JSP API. Example 3
Scopes within a web application

In JSP pages and servlets it is possible to define 4 different scopes for objects (attributes):

- Page scope
- Request scope
- Session scope
- Application scope
Sessions. Scopes within a web application

Page scope

An object (attribute) associated to a page scope is visible only within the JSP page to which that scope refers.

The page scope corresponds to the execution of the method _jspService of the servlet to which that JSP page is translated.

Object declaration within the page scope

```java
pageContext.setAttribute("attrId", object, PageContext.PAGE_SCOPE);

pageContext.setAttribute("attrId", object);
```

Recall that pageContext is an implicit object that can be used in a JSP page.
Request scope

An object (attribute) associated to a request scope is visible:

- Within the JSP page to which that http request refers and

- Within the pages invoked by means of `<jsp:include ..>` or `<jsp:forward ...>` from that page

Object declaration within the request scope

```
pageContext.setAttribute("attrId",object,PageContext.REQUEST_SCOPE);
```

```
request.setAttribute("attrId",object);
```

Recall that both pageContext and request are implicit objects that can be used in a JSP page
Session scope

An object (attribute) associated to a **session scope** is visible within all the requests associated with an HttpSession object with a specific session ID

Object declaration within the request scope

```java
pageContext.setAttribute("attrId",object,PageContext.SESSION_SCOPE);

session.setAttribute("attrId",object);
```

Recall that both pageContext and session are implicit objects that can be used in a JSP page

The session scope is used normally when the following situation occurs:

- The application maintains a dialogue with the user through different http requests **and**

- Subsequent requests within the dialogue should remember the state of some application objects stated by previous requests
Application scope

An object (attribute) associated to an application scope is visible within all the servlets and JSP pages of an application.

Object declaration within the request scope

```java
pageContext.setAttribute("attrId",object,PageContext.APPLICATION_SCOPE);

application.setAttribute("attrId",object);
```

Recall that both pageContext and application are implicit objects that can be used in a JSP page.
Initialization parameters of the application scope

- It is possible to declare and initialize some constant parameters which should be visible throughout the application.

- This initialization is done in the `web.xml` deployment descriptor:

```xml
<context-param>
  <param-name>jdbc.driver</param-name>
  <param-value>jdbc.postgresql</param-value>
</context-param>

<context-param>
  <param-name>...</param-name>
  <param-value>...</param-value>
</context-param>

... 

- This parameters can be retrieved within a JSP page/servlet:

```java
String driver = application.getInitParameter("jdbc.driver");
```