Handling State in Web Applications

Jeff Offutt

http://www.cs.gmu.edu/~offutt/

SWE 642
Software Engineering for the World Wide Web

sources: Professional Java Server Programming, Patzer, Wrox
Web Technologies: A Computer Science Perspective, Jackson, Prentice-Hall

Tracking State Information

• The initial versions of the web suffered from a lack of state:

  HTML Form ➔ Data ➔ Server ➔ Info ➔ HTML Page

• If you wanted multiple screens, there was no way for data to be accumulated or stored

  Form1 ➔ D1 ➔ Server
  Form2 ➔ D1+D2 ➔ Server
  Form3 ➔ D1+D2+D3 ➔ Server
  Form4 ➔ D1+D2+D3+D4 ➔ Server
Session Tracking

• Web applications must maintain user states

• This is called *session tracking*

---

Session Tracking (2)

**Session**: A series of related interactions between a client and a web server (similar to a use case)

• Session tracking refers to keeping data between multiple HTTP requests

• This problem is essential to maintaining state, which we understand quite well in the context of traditional procedural programming and object-oriented programming

• The Web brings in unique constraints

**HTTP is connectionless**  **Distributed**
New Control Flow and State Handling

To support session handling (and other issues)

J2EE introduced new language mechanisms

1. New control flow mechanisms
2. New state management
3. New variable scopes

Traditional Control Flow

• Procedural languages
  – Method / function calls
  – Decisions – if, while, for, repeat-until, switch, …
  – Static includes – other code pulled in before compiling
• OO languages
  – Dynamic binding via polymorphism
• Client / Server
  – Message passing
Web App Control Flow (1)

Traditional Control Flow Mechanisms
1. Same as traditional – Software on server and client
2. Synchronous message passing – Client to server, HTTP
   - Also server to other servers
3. Event handling – On the client

Web App Control Flow (2)

New Control Flow Mechanisms
4. Asynchronous message passing – Client to server, Ajax
5. Forward – Transfers control from one server component to another, no return
6. Redirect – Ask client to send request elsewhere
7. URL rewriting by users
8. Dynamic include – Control passes to another component, then returns, no parameters
9. Dynamic binding – Reflection allows new components to be added and used dynamically
**Ramifications of New Control Flow**

- The traditional control flow graph does not model essential parts of web app execution!
- UML diagrams do not model many of these.
- Most developers learn the syntax, but not the concepts behind these new control connections.

**Lots of poorly designed software ... and lots and lots of poorly understood software faults!**

**New Control Flow and State Handling**

To support session handling (and other issues)

**J2EE introduced new language mechanisms**

1. New control flow mechanisms
2. New state management
3. New variable scopes
Handling State in Procedural Languages

• The C programming language has simple ways to handle state

```
char name[25];
main()
{
    int x, y, z;
    ...
```

• We added several layers of scope in OO languages

State in Object-Oriented Languages

• In addition to local and global variables, OO languages have other scopes
  – Nonlocals: package, protected, default, …

• Data sharing in OO languages
  – Two components can share data if they are in the same scope
  – Two components can share data by passing parameters

• OO languages also are based on the concept of objects, which are instances of classes
  – Classes define types, which are global
  – Objects can be defined at multiple scopes
Handling State in Java

State on the Web

- These schemes have two simple, subtle, assumptions:
  1. The software components share physical memory
  2. The program runs to completion with active memory

- But these assumptions are violated in web applications!
  1. Distributed software components
  2. Connectionless nature of HTTP

- To keep state in web applications, we need different ways to store and access variables and objects

**Public access and parameter passing are not sufficient in Web applications!**
State and Session Tracking

- Session tracking refers to passing data from one HTTP request to another
- A web application is comprised of several software components
- The characteristics of a Web app means that the components do not communicate directly
  - Independent processes (threads)
  - Connectionless protocol
  - Client-server or N-tier architecture
  - Execution flow always goes through a client

How can these independent components share data?

Session Tracking Methods

1. Include data as extra parameters (URL rewriting)
2. Hidden form fields
3. Cookies
4. Servlet API session tracking tools

All four methods work by exchanging a *token* between the client and server
Non-servlet Methods—(1) URL Rewriting

• Forms usually add parameters
  URL ? P1=v1 & P2=v2 & P3=v3 & ...
• You can add values in the URL as a parameter:
  HREF = "./servlet/X ? SneakyParam=42" >
  or : User=george >
• This is used as a key to find the saved information about the user george
  – Messy and clumsy
  – Long URLs
  – Information on URL is public
  – All HTML pages must be created dynamically
  – Often limited in size

Non-servlet Methods—(2) Hidden Form Fields

• Flows of control go through the client
• Data that must be passed from one software component to another can be stored in hidden form fields in the HTML
• Generate HTML pages with forms that store “hidden” information:
  <INPUT type="hidden" name="User" value="george">
• Several problems:
  – Insecure – users can see the data
  – Unreliable – users can change the data
  – Undependable – users can use the back button, direct URL entry, and URL rewriting to skip some hidden form fields
• Still useful in limited situations
Non-servlet Methods—(3) Cookies

- **Cookies** are small files or text strings stored on the client’s computer
- Created by the web browser
- Arbitrary strings, but sometimes var=value pairs or XML
- Java coding:
  
  ```java
  Cookie c = new Cookie ("user", "george");
  c.setMaxAge (5*24*60*60); // expires in 5 days, in seconds
  response.addCookie (c); // sends cookie to client
  ```

Non-servlet Methods—(3) Cookies

- Cookies are very useful and simple
- Not visible as part of the HTML content
- Convenient way to solve a real problem
- But cookies are scary!
  - It’s as if I stored my files at your house
  - Cookies go way beyond session tracking
  - Cookies allow **behavior tracking**
(4) Servlet Sessions

The servlet API uses cookies to provide a simple, safe, flexible method for session tracking

- Cookies are handled automatically
- HttpSession stores data in the current active object
- Data disappears when the object is destroyed
- Object is destroyed after the session ends, usually 30 minutes after the last request

Sessions—Big Picture

![Diagram showing session tracking](image-url)
**Sessions—Big Picture**

Client stores the ID and sends it to the server in subsequent requests. Server recognizes all the requests as being from the same client. This defines a *session*.

Server recognizes these requests as being from a different client.

**Servlet API for Session Methods**

- `void setAttribute (String name, Object attribute)`: Adds an item to the session
- `Object getAttribute (String name)`: Returns the value stored for the given name, or null if no attribute with this name
- `void removeAttribute (String name)`: Removes an item from the session
- `Enumeration getAttributeNames()`: Returns an enumeration of all the value names that are stored for this session
- `String getID()`: Returns the session ID
- `void invalidate()`: Removes the current session
Servlet API Session Methods (2)

- These methods are **not** synchronized
- Multiple servlets can access the same session object at the same time
- If this can happen, your program should synchronize the code that modifies the shared session attributes

How to Use Session Objects

- Get a session object:
  
  ```java
  HttpSession s = request.getSession (true);
  ```

  - `true`: create if session does not exist.
  - `false`: return null if session does not exist.

- Put objects into the session object (not primitive types):
  
  ```java
  s.setAttribute ("answer", 42); // does not work
  s.setAttribute ("answer", new Integer (42));
  ```

- Get values from session objects (just like a hash table):
  
  ```java
  Integer ansobj = (Integer) s.getAttribute ("answer");
  int ans = ansobj.intValue ();
  ```

- Deleting session:
  
  ```java
  s.invalidate (); // Information is thrown away
  ```

- Setting the time-out duration
  
  ```java
  s.setMaxInactiveInterval (int); // seconds until deletion
  ```
Session Definition

A session is defined by

- The web server
  - Servlet container
  - Servlet context
- The client
  - IP address
  - Browser
- Session objects are kept on the server
- Each session object uses different parts of memory (instances of data values) on the server

Example

Consider a small Web app with 2 servlets and 3 JSPs

How can the servlets and JSPs share data?
Sharing Data: Session Object

- One program component can store a value in the session object.
- Another component can retrieve, use, and modify the value.
- Depends on the servlet container:
  - Software components are threads, not processes.
  - Servlet container stays resident and can keep shared memory.

Session Data Example

Software components share “container” access data.
Login Example

1. User request
2. Check isLoggedIn
3. if isLoggedIn false
4. Set isLoggedIn true and set userID
5. User request
6. Check isLoggedIn
7. if isLoggedIn false

Session Scope and Context Scope

- The session object is available to software components in the same request and session
  - They have access to the session ID
  - This is called **session scope**
- Sometimes we need a wider scope
  - Chat rooms: Allow multiple users to interact
  - Group collaboration: Online meeting
  - Online bidding
  - Reservation systems
- J2EE also defines a **context scope**

This allows us to share data among multiple users
**Context Scope**

*Container Engine*

Servlet S1

JSP 1

JSP 2

Servlet S2

JSP 3

**Servlet Context Object**

The servlet context object supports resources that can be shared by groups of users:

- **Get a servlet context object**
  
  ```java
  ServletContext servContext = getServletContext();
  ```

- **Share information through context attributes**
  
  1. `servContext.getAttribute()`
  2. `servContext.setAttribute()`
  3. `servContext.removeAttribute()`

- **Information about servlet’s environment**:
  
  - Server name
  - MIME type

- **Method to write to a log file** (`log()`)
Session and Context Scope Examples

Compare attributeServlet and servletContext examples

http://cs.gmu.edu:8080/offutt/servlet/attributeServlet
http://cs.gmu.edu:8080/offutt/servlet/servletContext

Try them in different browsers
Compare the differences

Summary

• Managing state is fundamental to any program
• Managing state is the most unique aspect of designing and programming web applications
• Software vendors are creating new frameworks all the time
  – Most of them introduce additional state handling techniques
• Many professional developers make fundamental mistakes with managing state

State management is the most common source of software faults in web applications