State Handling with Java Servlets

CS 4640
Programming Languages for Web Applications

[Based in part on SWE432 and SWE632 materials by Jeff Offutt]
[Robert W. Sebesta, “Programming the World Wide Web”]
Interactions in Web Apps

Each request/response is treated independently

1\textsuperscript{st} interaction $\rightarrow$ 1\textsuperscript{st} request/response

2\textsuperscript{nd} interaction $\rightarrow$ 2\textsuperscript{nd} request/response

“Connectionless” $\rightarrow$ “Stateless”
Session State Information

The initial versions of the web suffered from a lack of state.

If a web app needed multiple screens, there was no way for state to be accumulated or stored.

• This is due to the stateless property of HTTP.

In reality, we may want to keep track of the information (i.e., state).
Session Tracking

• Web sites that are service-oriented or e-commerce need to maintain user state

• This is called session tracking

• Session = a series of related interactions between a client and a web server

• Session tracking = keeping data between multiple HTTP requests

• The web brings in unique constraints:
  • HTTP is connectionless
  • Web apps are distributed
New Control Flow and State Handling

To support session handling (and other issues), Java EE introduced new features

- New control flow features
- New state management
- 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 Web 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

4. **Dynamic include** – Control passes to another component, then returns, no parameters
Web App Control Flow (2)

New Control Flow Mechanisms

5. **Forward** – Transfers control from one server component to another

6. **Redirect** – Ask client to send request elsewhere

7. **Asynchronous message passing** – Client to server, Ajax

8. **Operational transitions** – Controlled by users and browsers (back button, forward, URL rewriting, history, caching)

9. **Dynamic binding** – Reflection allows new components to be added and used dynamically, no return
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
New Control Flow and State Handling

To support session handling (and other issues), Java EE introduced new features

• New control flow features
• New state management
• New variable scopes
Handling State in Procedural Languages

- The C programming language has simple ways to handle state

```c
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**
State on the Web

• Two assumptions (traditional software):
  1. The software components share physical memory
  2. The program runs to completion with active memory

• These assumptions are violated in web apps due to
  1. Distributed software components
  2. Connectionless nature of HTTP

• Need ways to store and access variables and objects to keep state in web apps

Public access and parameter passing are not enough for web apps
State and Session Tracking

• Session tracking
  • Passing data from one HTTP request to another

• A web app
  • Composed of several web software components

• Web 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

• **URL rewriting** -- include data as extra parameters

• **Hidden form fields**

• **Cookies**

• **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
  
  \[\text{URL}\text{?}\text{P1}=v1\&\text{P2}=v2\&\text{P3}=v3\&\ldots\]  

• You can add value in the URL as parameters
  
  \[\text{href}="../\text{servletname}\text{?User=george}"\]  

• The above example is used as a key to find the saved information about the user george

• Drawback
  
  • Messy and clumsy
  
  • Long URLs
  
  • Information on URL is public
  
  • All HTML pages must be created dynamically
Non-servlet Methods:
(2) Hidden Form Fields

• Flows of control go through the client

• Store data to be passed from one software component to another 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
  • Small files or text strings
  • Created by the web browser
  • Arbitrary strings stored on the client
  • From the server’s (Java) perspective: val_name=value pairs
  • 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 – cont.

• Cookies
  • Useful and simple
  • Not visible as part of the HTML content
  • Convenient way to solve a real problem

• Cookies are scary!
  • It’s as if I stored my files at your house
  • Cookies go way beyond session tracking
  • Cookies allow behavior tracking
Servlet Methods: (4) Sessions

- 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 (Overview)

Server returns a new unique session ID when the request has none
Sessions (Overview)

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

Server recognizes all the requests as being from a different client.
## Servlet API for Session Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setAttribute(String name, Object attribute)</code></td>
<td>Adds an item (name) with its value (attribute) to the session</td>
</tr>
<tr>
<td><code>Object getAttribute(String name)</code></td>
<td>Returns the value stored for the given name</td>
</tr>
<tr>
<td><code>void removeAttribute(String name)</code></td>
<td>Removes an item from the session</td>
</tr>
<tr>
<td><code>Enumeration getAttributeNames()</code></td>
<td>Returns an enumeration of all the value names that are stored for this session</td>
</tr>
<tr>
<td><code>String getID()</code></td>
<td>Returns the session ID</td>
</tr>
<tr>
<td><code>void invalidate()</code></td>
<td>Removes the current session</td>
</tr>
</tbody>
</table>
Servlet API for 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
Using Session objects

• **Initialize** a session object:
  
  ```java
  HttpSession s = request.getSession(true);
  - true       // create if it does not exist
  - False      // return null if it 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** primitive values from session objects:
  
  ```java
  Integer ansobj = (Integer)s.getAttribute("answer");
  int ans = ansobj.intValue();
  ```

• **Delete** session:
  
  ```java
  s.invalidate(); // Information is thrown away
  ```
Session Definition

A session is defined by

1. The web server
   • Servlet container
   • Servlet context

2. 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
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
More on Maintaining State

Single-user session state

- Cookies and session object

Multi-user session state

- Servlet-context object

Why do we need them?

- Social network system – allow multiple users to interact
- Group working – online meeting
- Online bidding
- Reservation system, invitation system

Sometimes we want to share session data among multiple clients
Context Scope

Container Engine

Servlet S1

session object 1

JSP 1

JSP 2

context object

Servlet S2

session object 2

JSP 3

JSP 4

Session 1

Session 2

Context (application)
.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
  
  ```java
  servContext.getAttribute()
  servContext.setAttribute()
  servContext.removeAttribute()
  ```

• Information about servlet’s environments:
  • Server name

• Method to write to a log file (`log()`)

---

Spring 2018 – University of Virginia
© Praphamontripong

31
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
  • Books and tutorials describe syntax, but not concepts

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