Server State Management

Definition of the Problem

Portable Mechanisms

Conversational State

Best Practices

The Problem

HTTP is *stateless*

- Each request made by the client has no dependency, or context, based on previous requests made by that same client

Typically the programs we right are *stateful*

- The response we give to the user is based on previous interactions with the user and events in the application's environment

So, we have an impedance mismatch problem.

- The protocol expects each request-response is independent of any other, yet our programs want to rely on a dependence of prior info

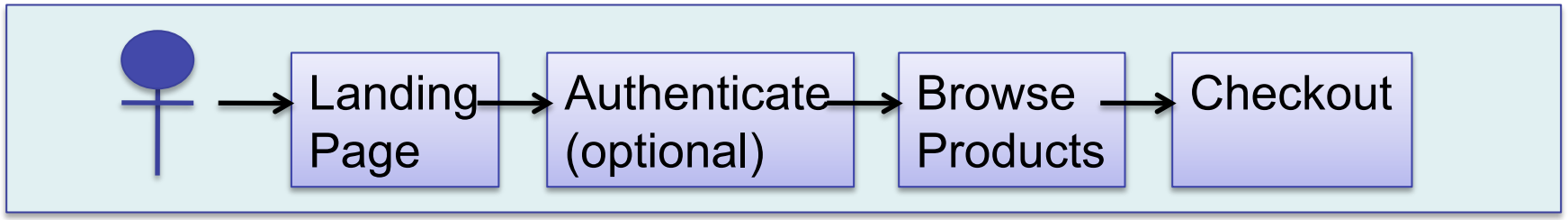
Types of state relevant to web development:

1. *Application state* – the lifecycle of your application process itself; inherently important to the component container model
2. *User Preferences state* – understanding who the user is, what they typically want to do and how they want to interaction with you
3. *World state* – information independent of a specific program
4. *Conversational state* – the topic of this talk.

Motivational Example

Example Use Case: The Shopping Cart

- User browses product inventory, selects items for purchase



- Problem 1: as the user browses and selects items and puts them her/his shopping cart, how does the application know which shopping cart to put them into and what the choices were? (conversational state)
- Problem 2: How do we remember what the user browsed for before? What her/his preferences are? (User preferences state)
- Problem 3: How do we access the inventory for product browsing? (world state)
- Problem 4: What happens if the application crashes in the middle of your interaction and you have to restart? What happens if a new version of the application comes out while you are using it! (app state)

Solution Architecture – Conversational State

Conversational State

- *Represents a bounded "conversation" or interaction between a client and a server*
- A web app may support the creation and deletion of several conversations during a user's interaction

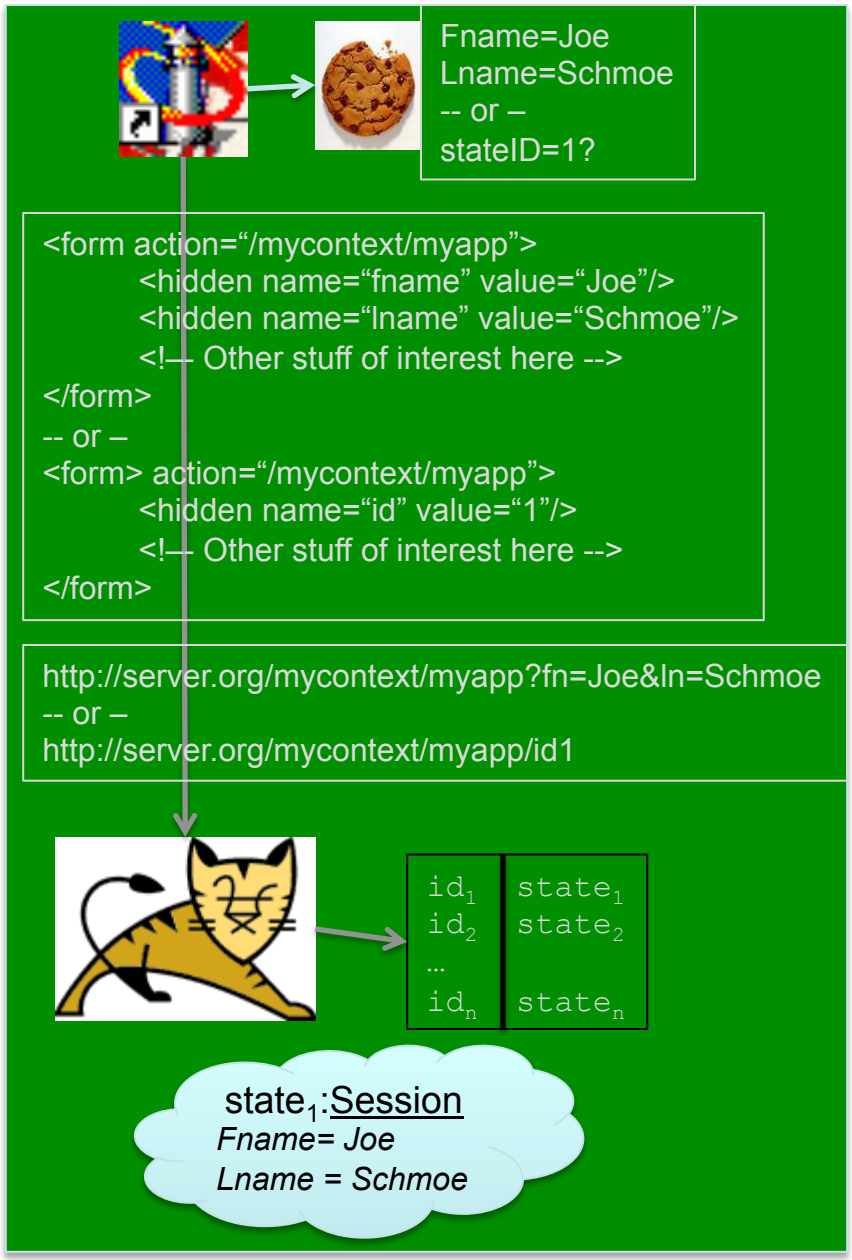
How do you solve the conversational state issue from an application architecture standpoint?

- Well, you have to maintain state somewhere
- Your choices on the web are the client or the server

Mechanisms for Managing Conversational State

1. Hidden form fields
   - <hidden name="key" value="1234"/>
   - Pros: portable
   - Cons: not secure, all interactions via a form (button or script)
2. URL Rewriting: Append state or key to URL
   - Pros: portable
   - Cons: possibly not secure, all URLs need to be modified
3. Cookies: Set a cookie value for state or key
   - Pros: Default mechanism, embedded in response header
   - Cons: User may disable
4. SSL Sessions – SSL establishes a session
   - Pros: Session is encrypted, key exchange part of the process
   - Cons: Computationally more expensive, user experience
5. Session-oriented middleware – *stay tuned…*

Client or Server?



Options 1-3 could be used to allow the client to see/store state

- Hidden form fields
  - Forms can have any number of hidden fields
  - Requires your navigation to be in the form
- URL Rewriting
  - You could encode flow in the URL itself
  - Even with query string limitations, you can use a dynamic URL
  - Some consider this "RESTful"
- Cookies
  - Set a cookie for each name/value pair
  - Is OK for small "innocent" pieces of data
  - Is not tied to flow, so could get unwieldy

What about server-side state?

- Store/send a temporary "session" ID
- ID is a key to look up state on server
- Basically what Java and Node do
- What about our scalability issue?

Java example: Server-side Session Tracking

Session objects live on the server and are automatically associated with client via cookies or URL-rewriting

- Use `request.getSession` to get session
  - Behind the scenes, the system looks at cookie or URL extra info and sees if it matches the key to some previously stored session object. If so, it returns that object. If not, it creates a new one, assigns a cookie or URL info as its key, and returns that new session object.

Hashtable-like store allows arbitrary objects inside session

```
HttpSession session = request.getSession();
SomeClass value = (SomeClass)session.getAttribute("someID");
if (value == null) {
  value = new SomeClass(...);
  session.setAttribute("someID", value);
}
doSomethingWith(value);
```

- Do not need to call setAttribute again (after modifying value) if the modified value is the same object. But if value is immutable, modified value will be a new object ref, and you must call setAttribute again.

We will see something quite similar in Node/express soon!

Conversational State Management Best Practices

The Web dev community basically agrees on the following best practices for conversational state management:

1. The state management technique matters
    1. Avoid the <hidden> solution – too easy to break, and too easy to steal sessions
    2. URL rewriting is better but suffers from similar problems
    3. Cookies are better than URL rewriting, but it is still possible to steal, and cookies may persist on the client
    4. SSL sessions best but can be a pain from both the user experience and computational scalability standpoint
2. Use a session framework instead of “rolling your own”
3. Make sure session ids are not based on security or privacy info, and that they are temporal GUIDs

Summary

Conversational State is the name for a bounded interaction between web browser and server.

- Several techniques exist for maintaining conversational state, though some are legacy and all raise issues.
- A Session API is best used for conversational state
- On HTML5 clients we now have SessionStorage (is it the same?)

It is not the only form of state

- Universal, or world model state
  - Is not bounded; is in fact app-independent and outlives app processes
  - Databases (broadly defined) are best used for world model state
- User preferences state
  - State pertaining to a user that informs her/his interactions with your apps
  - May also be used by apps to track and share user interaction information
  - Cookies are best used for User Preferences state per client
  - Often the server-side will also store preferences in a profile datastore