

Objectives

- Servlets Review
- Handling Multiple Requests
- JSPs
- Web Application Code Organization
- Project

Servlets Review

- What happens when a web application starts?
 - What is the servlet life cycle?
- Init parameters
 - Why do we use init parameters?
 - Where are init parameters defined?
 - How do we access a servlet's init parameter?
- How can we save state across multiple requests from a user?
 - What are the pros and cons of each?
- How are *parameters* different from *attributes*?

Review: Servlet Life Cycle in Web Application Server (WAS)



1. Web application server creates **one** instance of servlet
 - Calls **init** method of servlet created
2. As requests come in, WAS calls **service** method of appropriate servlet
 - In turn, servlet calls appropriate **doMethod**
3. When web application server shuts down, calls **destroy** method of each servlet

Handling State across Multiple Requests

- Hidden parameters
 - Just hides info – client can access and change
 - Cookies
 - Passed back and forth between the clients and servers
 - Stored on the client side -- unreliable
 - Session state
 - Stored on the server, times out
- See yesterday's slides for fuller discussion

Review: Parameters vs Attributes

Parameters

- Represent data set in a request OR configuration parameters for a servlet
- Lookup by name
- Must be a String

Attributes

- Represent information stored on the server
- Look up by name
- Can be any data type

Advanced Topic

HANDLING MULTIPLE REQUESTS

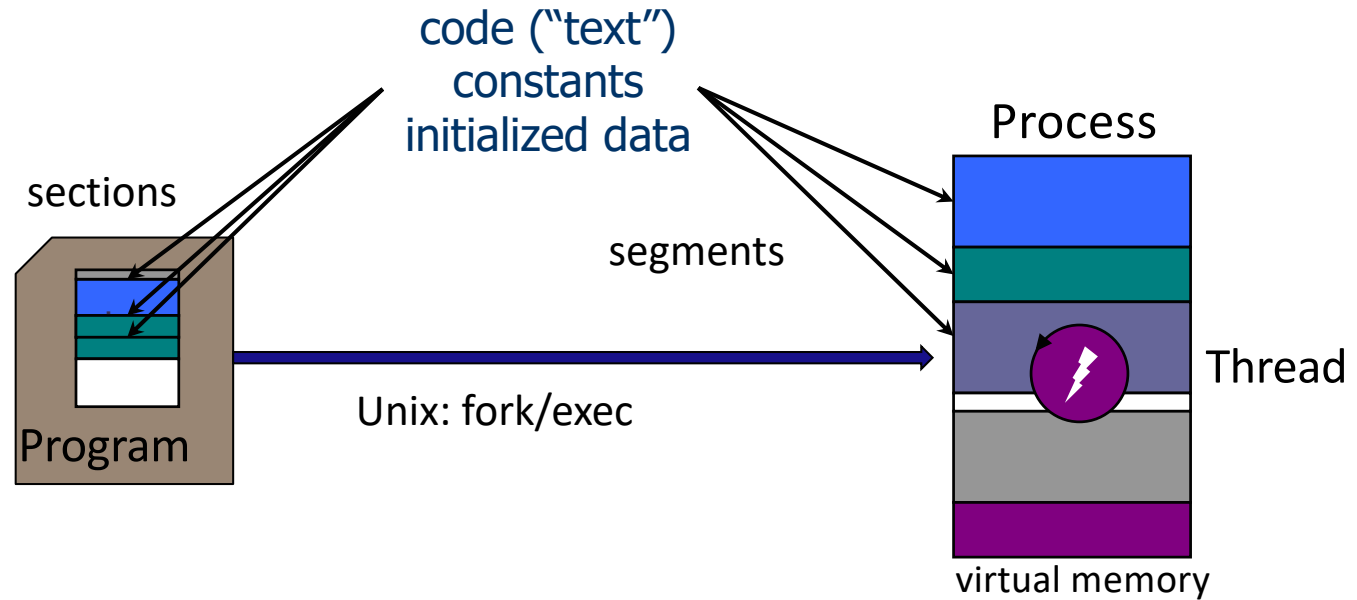
Handling Multiple Clients

- Web servers get lots of requests from users
- Web server handles multiple requests at a time by executing multiple *threads*
 - Approximately 1 thread/request



Web Application Server

OS Background: Processes



When a program launches, the Operating System creates a **process** to run it, with a main **thread** to execute the code and a **virtual memory** to store the running program's code and data.

Processes and Threads

Virtual Address Space (VAS)

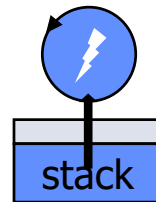


The address space is a private name space for a set of memory segments used by the process.

The kernel must initialize the process virtual memory for the program to run.

To process, looks like has access to all the memory.

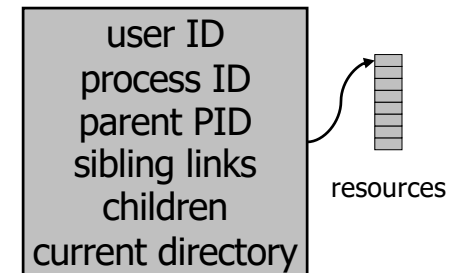
Thread(s)



+

+

Process Control Block (PCB)



Each process has **at least one** thread (the “main thread”) bound to the VAS. Each thread has a stack addressable in the VAS, i.e., has its own state (**where in program, state from executing program**).

The kernel can suspend/restart a thread wherever and whenever it wants.

The OS maintains some **kernel state** for each process in the kernel’s internal data structures: e.g., a file descriptor table, links to maintain the process tree, current directory, and a place to store the exit status.

Multiple Clients

- Web servers get lots of requests from users
- Web server handles multiple requests at a time by executing multiple *threads*
 - Approximately 1 thread/request



Web Application
Server

- Request from blue user is handled by blue thread
 - It's on line 68 right now
- Similarly, the green user's request is on line 50

Multiple Clients

- Web servers get lots of requests from users

- Web server handles multiple requests at a time by executing multiple *threads*



Web Application Server

- Approximately 1 thread/request

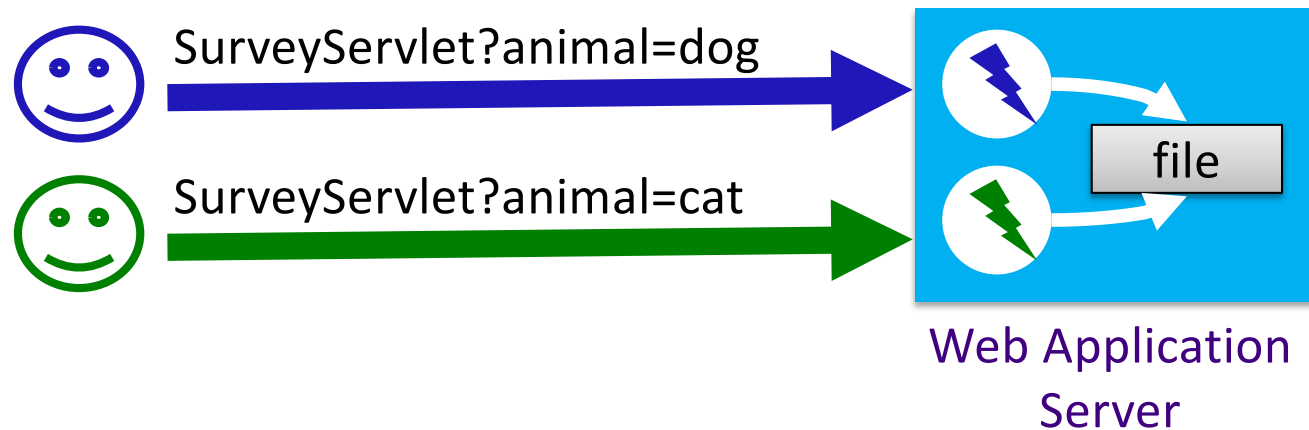
- Benefit: faster response times to users

⇒ Need to ensure sure that threads overlap in ways that do not break the application

Example Scenario

- **SurveyServlet** stores the results of the survey in a file on the server
- When >1 client connects to the server at one time, server handles both clients **concurrently**
 - >1 thread can execute **SurveyServlet**
 - >1 thread can read/modify file at one time
 - Can lead to inconsistent data!

SurveyServlet Implementation

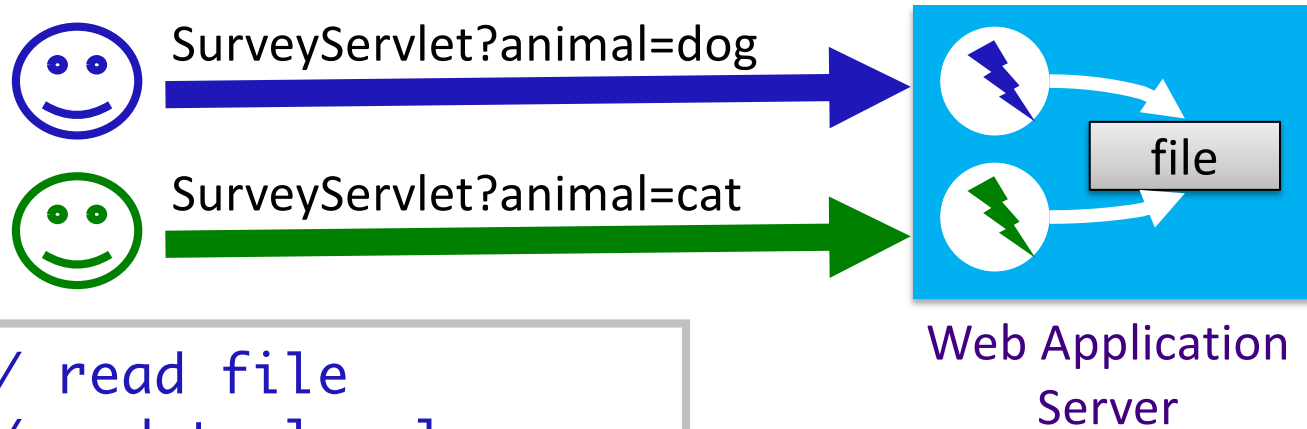


```
// read file  
// update local array  
// write file  
// print results
```

```
// read file  
// update local array  
// write file  
// print results
```

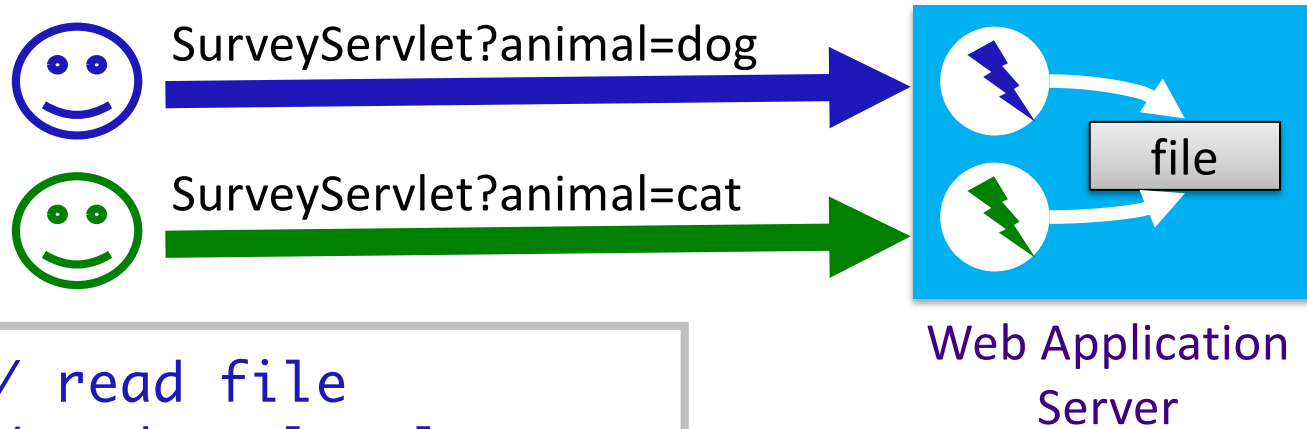
- Operations of each thread can overlap

Thread Interleaving (No Concurrency)



```
// read file
// update local array
// write file
// print results
// read file
// update local array
// write file
// print results
```

Bad Interleaving



```
// read file
// update local array
// read file
// update local array
// write file
// print results
// write file
// print results
```

What happens in this case?

Loses blue's vote

Critical Section

- Sections of code that should happen uninterrupted or **atomically**
 - Only one thread can execute at a time
- What is the critical section in this code?

```
// read file  
// update local array  
// write file  
// print results
```


Critical Section

- Sections of code that should happen uninterrupted or **atomically**
 - Only one thread can execute at a time
- What is the critical section in this code?
 - The shared file must be read and written atomically
- **Writes** to shared data cause trouble

```
// read file  
// update local array  
// write file  
// print results
```

210 in 335

- Even if only one Java statement in critical section, synchronize it!
- One high-level Programming Language statement often translates into multiple VM language statements
 - Prevent interruption at low level

High-level:

```
count++;
```

Virtual Machine level:

```
Retrieve count  
Add 1 to count  
Store count
```

Synchronization Mechanisms

- Synchronized classes
- Synchronized methods
- Synchronized statements
- Expense associated with each of these
 - But without it, get wrong or inconsistent answers!

Synchronized Methods

- When a thread calls a **synchronized** method of an object, that *object* becomes **locked**
 - Exactly 1 shared key for an object
 - Example: restroom key at a gas station
 - Thread must have key to enter an object's synchronized method
 - With key, unlock the door to synchronized method you want to enter
 - When another thread attempts to enter a synchronized method, it cannot get the key, so it *blocks*
 - Blocking thread waits for the key

Synchronized Statements

- Every Java **Object** has an implicit lock or **monitor**
 - **wait, notify, notifyAll** methods
- Synchronize a block of code on an **Object**

```
synchronized (this) {  
    ...  
}
```

Finer granularity than methods

```
synchronized (object) {  
    ...  
}
```

If **this** doesn't need to be synchronized because **object** is independent of other data in **this**

General Rules

- Need to synchronize access to *shared resources*
 - Instance variables, Files
 - Sessions
 - Don't need to worry about local variables
- Want to limit size of critical section
 - Larger section reduces amount of concurrency
- Programmer must be very careful not to write programs in which deadlock can occur
 - Careful synchronization: keep it simple

Synchronizing SurveyServlet

- Identify the shared variables
- Identify when shared variables are written, when they are read
- Identify the critical section
- How would you synchronize?

Aside: SurveyServlet

- Should periodically write the survey results but not hold up requests
 - Separate thread to write results

Synchronization Mechanisms

- In code for in-memory state
 - Synchronized classes
 - Synchronized methods
 - Synchronized statements
- Alternative: database (later!)
 - Handles synchronization on *persistent* data
- Expense associated with each of these
 - But without it, get wrong or inconsistent answers!

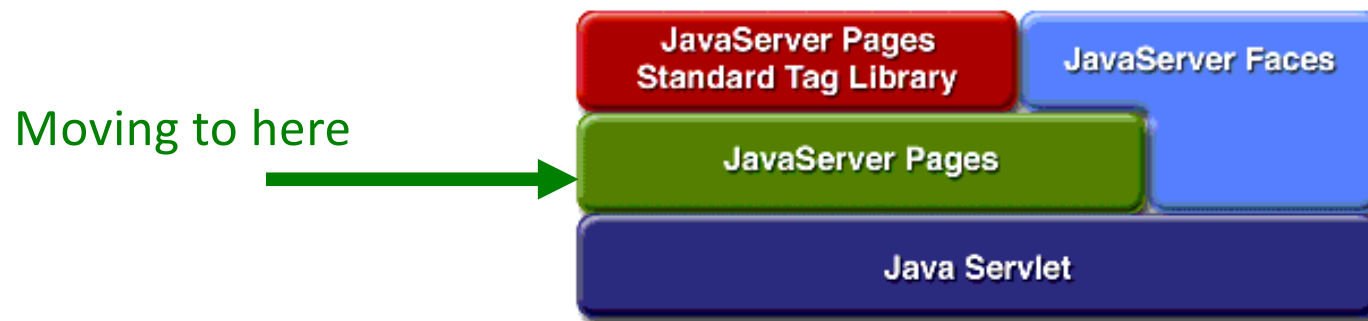
JAVASERVER PAGES (JSPS)

Discussion

What made writing servlets difficult?

Motivation: JavaServer Pages (JSPs)

- Simplify web application development
- Separate UI from backend code
 - Separate presentation layer
- Difficult to write HTML in print statements



JavaServer Pages (JSPs)

- Merge HTML and Java
 - Separate static HTML from dynamic
 - Make HTML templates, fill in dynamic content
 - Encourages separation of tasks
- Web application server compiles JSPs into servlet code
 - Clean and efficient
- Easier to develop, deploy, modify scripted pages
 - How much trouble did you have with HTML in Strings?

JSP Syntax: Expression

- Enclosed code in `<%= %>`
- Are evaluated and turned into a String

```
<html>
<body>
<p>
Hello! The time is now <%= new java.util.Date() %>
</p>
</body>
</html>
```

Expression

Evaluated, turned into a String

JSP Syntax: Scriptlet

```
<html>
<body>
<%
    // This is a scriptlet.  The "date" variable
    // we declare here is available in the
    // embedded expression later on.
    java.util.Date date = new java.util.Date();
%>
<p>
Hello!  The time is now <%= date %>
</p>
</body>
</html>
```

What is the syntactic difference between a scriptlet and an expression?


Example: SurveyServlet Output as a JSP

```
<%  
for (int i = 0; i < animalNames.length; i++) {  
%>  
  <tr>  
    <td><%=animalNames[i]%></td>  
    <td><%=votes[i]%></td>  
    <td><%=formattedPercentages[i]%></td>  
    <%  
      totalVotes += votes[i];  
    %>  
  </tr>  
  <%  
    }  
%>
```

Use the JSP terminology to describe what this code does

Example: SurveyServlet Output as a JSP

```
<%  
for (int i = 0; i < animalNames.length; i++) {  
%>  
  <tr>  
    <td><%=animalNames[i]%></td>  
    <td><%=votes[i]%></td>  
    <td><%=formattedPercentages[i]%></td>  
    <%  
      totalVotes += votes[i];  
    %>  
  </tr>  
  <%  
    }  
%>
```

 To be displayed at end

JSP Directives

- Page Directive

- Java files to import (like `import` statement in Java)

```
<%@ page import="java.util.*, java.text.*" %>
```

```
<%@ page import="ourcode.MyClass"%>
```

- Include Directive

- Include contents of another file: JSP, HTML, or text
- Example: include site's common headers or footers

```
<%@ include file="header.jsp" %>
```

JSP Variables

- By default, JSPs have some variables
 - **Not explicitly declared in the file**
 - **HttpServletRequest request**
 - **HttpServletResponse response**
 - **HttpSession session**
- JSPs can access request parameters, session data

These variable names
must be used

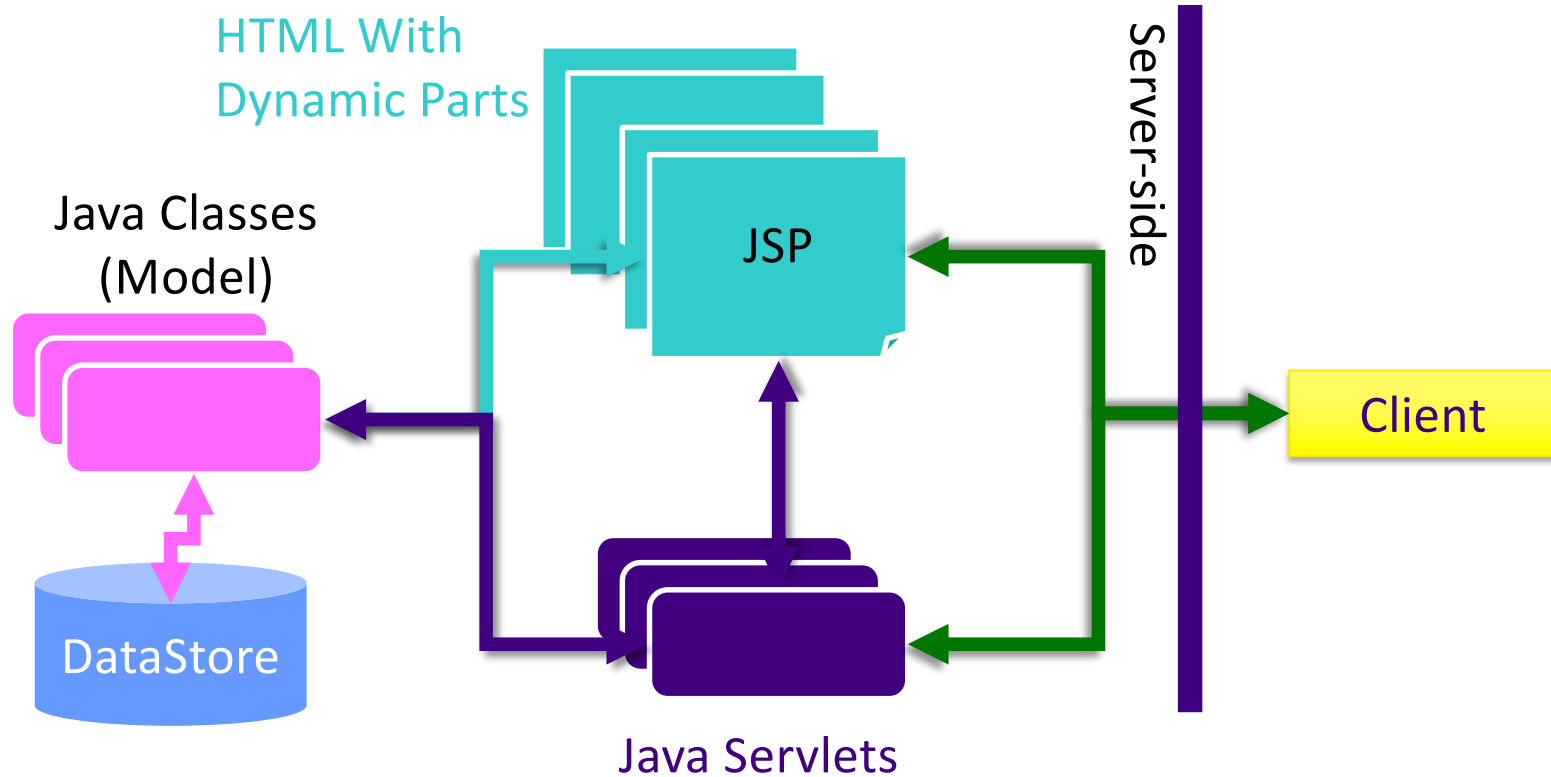
JSP Declarations

- For instance variables and methods

```
<%!  
    private ArrayList users;  
  
    public void jspInit() {  
        // on start up: set up  
    }  
    public void jspDestroy() {  
        // on shut down: clean up  
    }  
%>
```

- We won't do too much of this
 - Let servlets do the work

Web Application Architecture

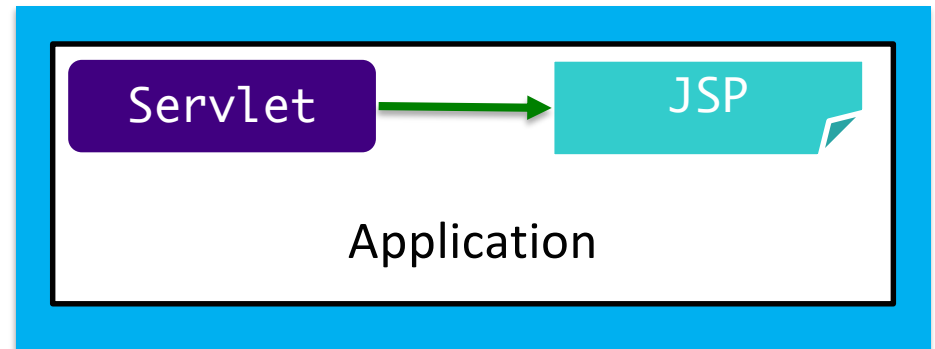


- Heavy lifting [code] of requests
- Forward to JSPs

Communicating Between JSPs and Servlets: Attributes

- **Attributes**

- Name/Value pairs
- Values are **Objects**



Web Application Server

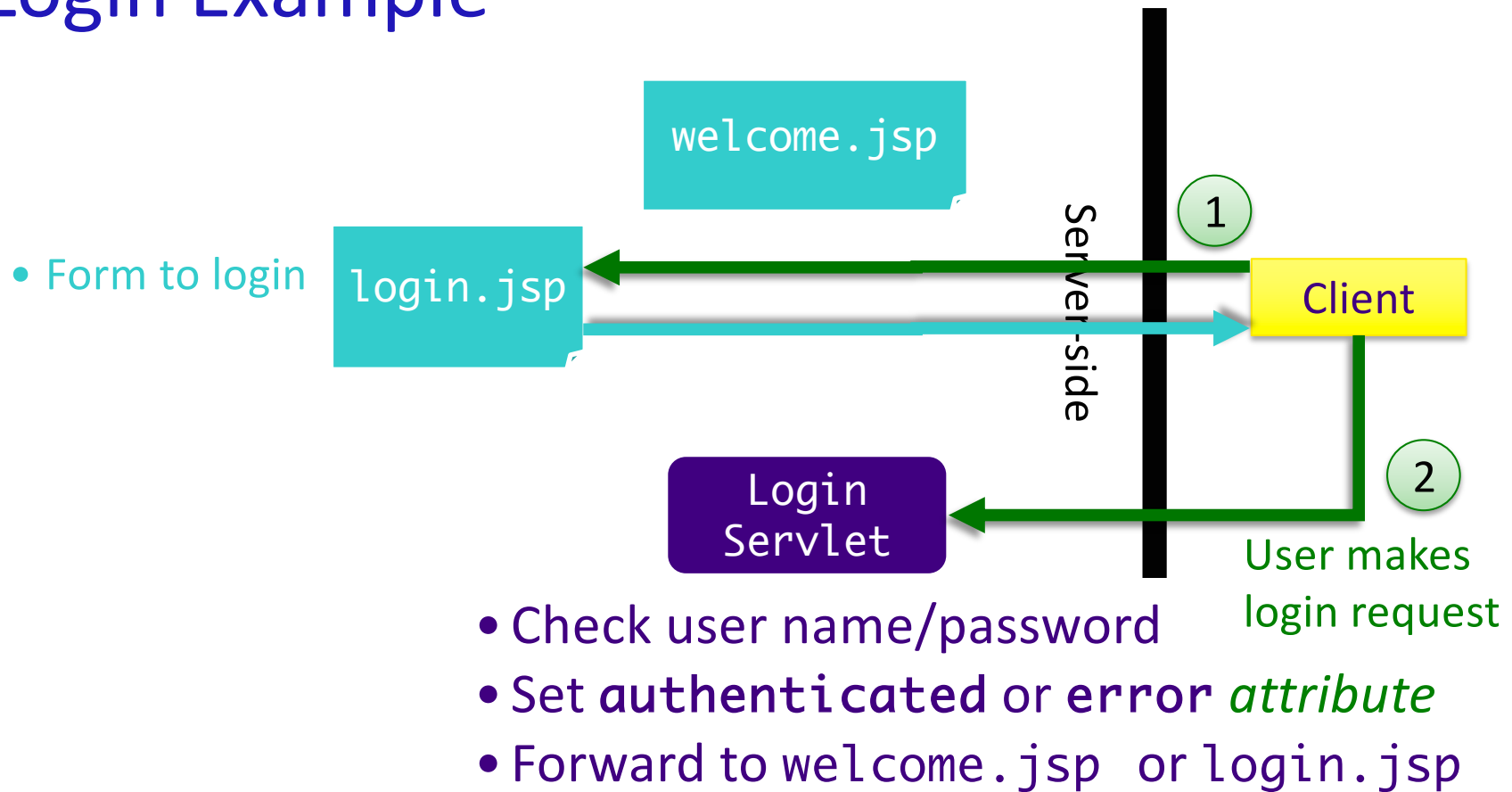
- 3 types of attributes

- Differ in where they are stored/their context/their lifetimes
 - Request
 - Session
 - Application – for the whole application

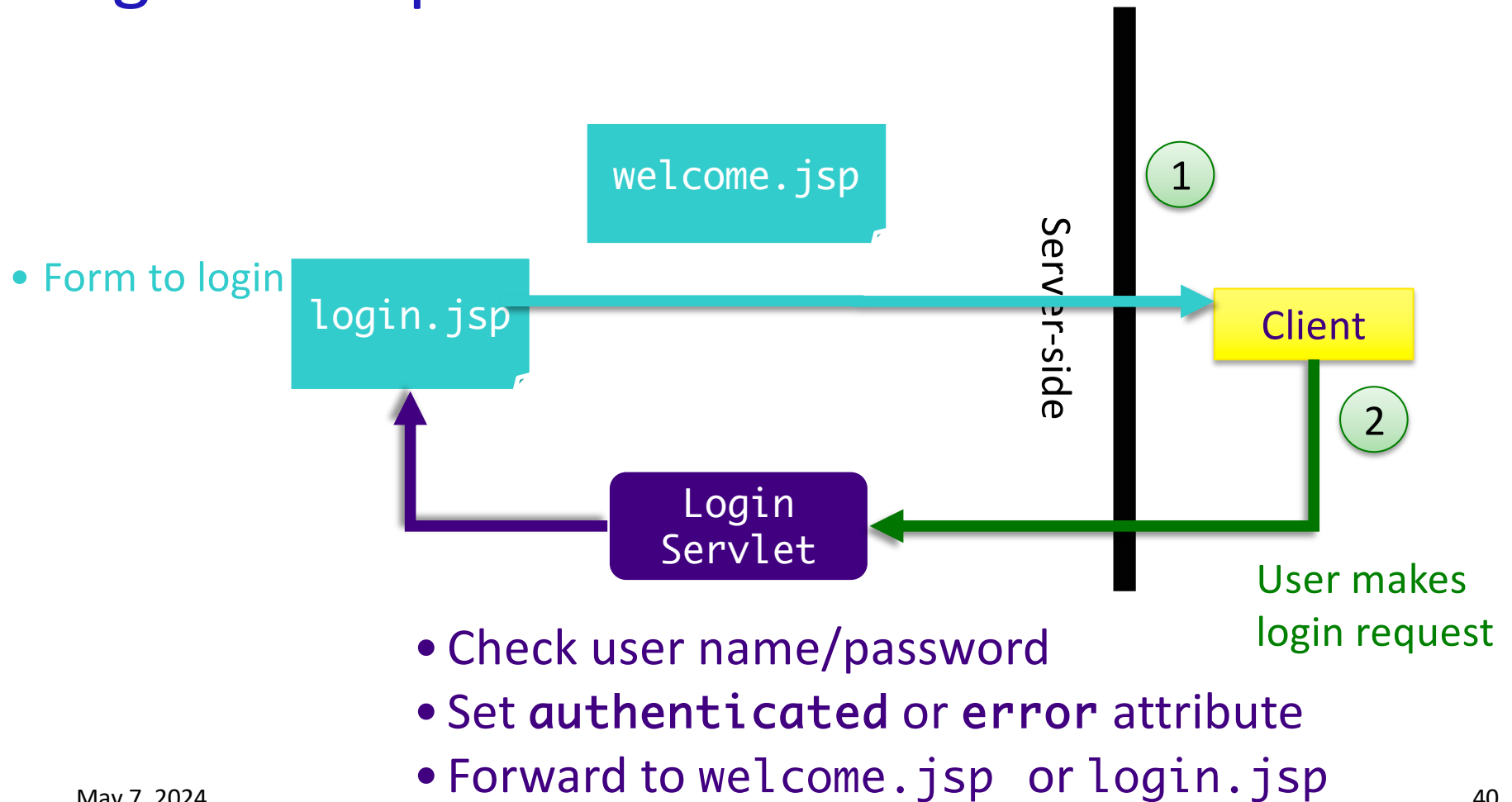
- Typical use:

- Set attribute in Servlet
- Get attribute in JSP

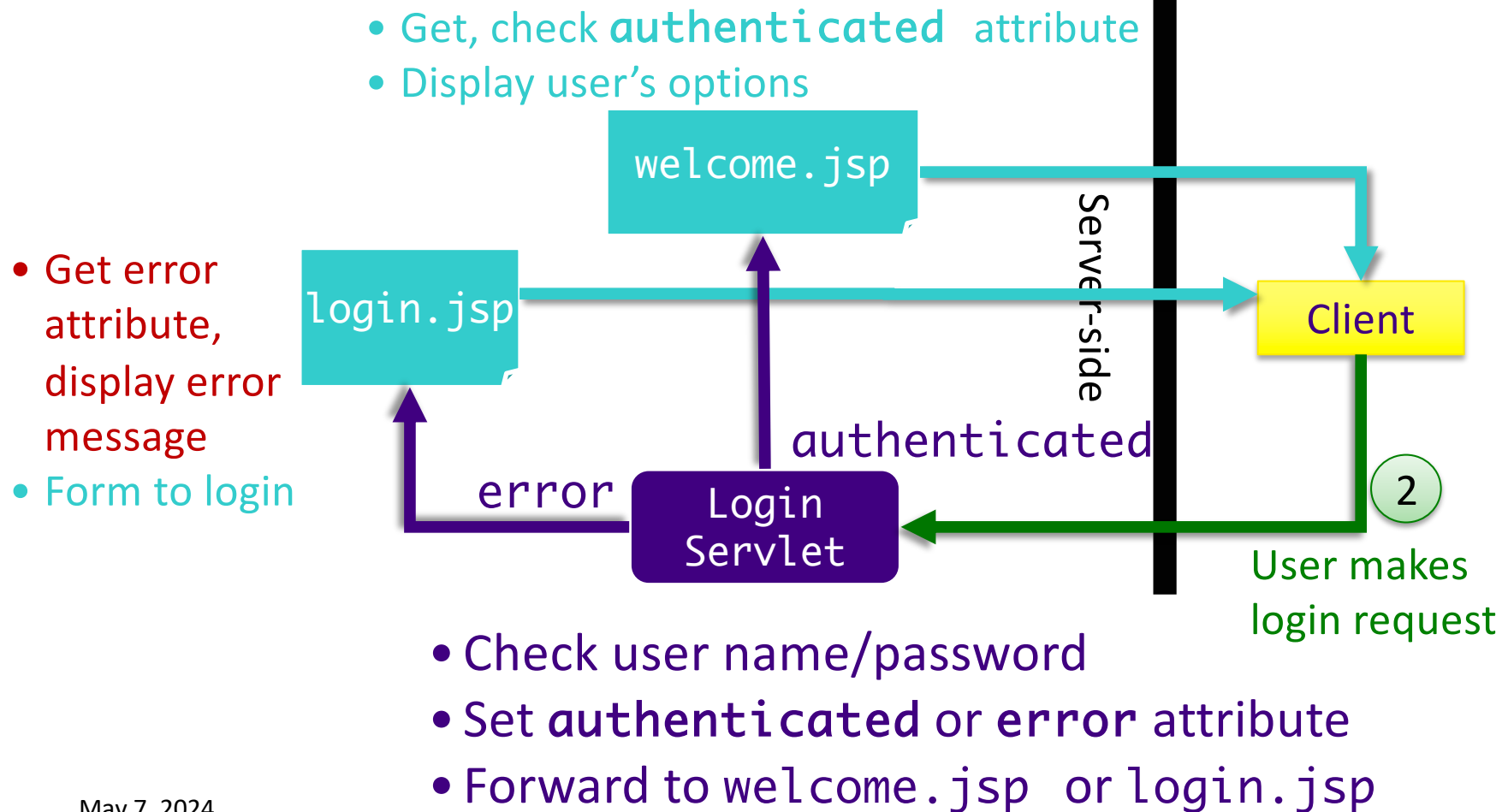
Communicating Between Servlets and JSPs: Login Example



Communicating Between Servlets and JSPs: Login Example



Communicating Between Servlets and JSPs: Login Example



Forwarding Requests from Servlet

- `HttpServletRequest`'s `getRequestDispatcher` method
 - Returns a `RequestDispatcher` object

The name of the resource to forward to



```
request.getRequestDispatcher("welcome.jsp").  
    forward(request, response);
```

- Can use `RequestDispatcher`'s `include` method similarly

Protecting JSPs

- If there are JSPs that you don't want users to be able to access directly by typing in the URL, put them in the `WEB-INF` directory
 - Web application server blocks access to the JSP
 - Don't need code to check authorization again
 - Only get to JSP through a servlet that checks authorization
- Forward requests from a servlet to the JSP by including `WEB-INF` in the URI

Using the WEB-INF Directory

- Example: User shouldn't be able to access petResponse.jsp directly

```
request.getRequestDispatcher("WEB-INF/petResponse.jsp").  
    forward(request, response);
```

Adding a JSP to SurveyServlet

- Separate heavy lifting from the HTML
- Think of JSP as a template
 - What is static about the response page?
 - What is dynamic?
- Servlet will handle most of the code

[Look at code](#)

Trick: Ternary Operator

- Alternative `if-then-else` syntax
- Returns a value
- Example:

Condition

```
minVal = (a < b ? a : b);
```

- Assign `minVal` value `a` if condition is `true`, `b` if condition is `false`

Ternary Operator in JSP

- Allows for more concise code

```
<input type=text name="username"  
value="<%= userName != null ? userName : ""%>">
```

Condition

Returned if true

Returned if false

HttpServletRequest

- `getContextPath()`

- Returns the portion of the request URI that indicates the context of the request.

- Example with various Request methods

`http://example.com:8080/app/dirpath/index.jsp?cat=2&cat=5`

```
getScheme() → "http"  
getServerName() → "example.com"  
getServerPort() → 8080
```

```
getContextPath() → "/app"  
getServletPath() → "/dirpath"  
getPathInfo() → "/index.jsp"  
getParameter("cat") → "2"  
getParameterValues("cat") → {"2", "5"}
```


Use in JSP

```
<a href="<%=request.getContextPath()%>">  
Main Page</a>
```

Synthesis

- Why JSPs?
- How should you organize your code?
 - What code should be in servlets?
 - What code should be in JSPs?
 - How do you communicate between them?

TODO

- Lab 6 - JSPs
- Web Quality Attributes – Reading
- Exploring Ancient Graffiti Project