Introduction to Java Servlets

Why Build Web Pages Dynamically?

- The Web page is based on data submitted by the user
  - E.g., results page from search engines and order-confirmation pages at on-line stores
- The Web page is derived from data that changes frequently
  - E.g., a weather report or news headlines page
- The Web page uses information from databases or other server-side sources
  - E.g., an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale.

Server-Side Java

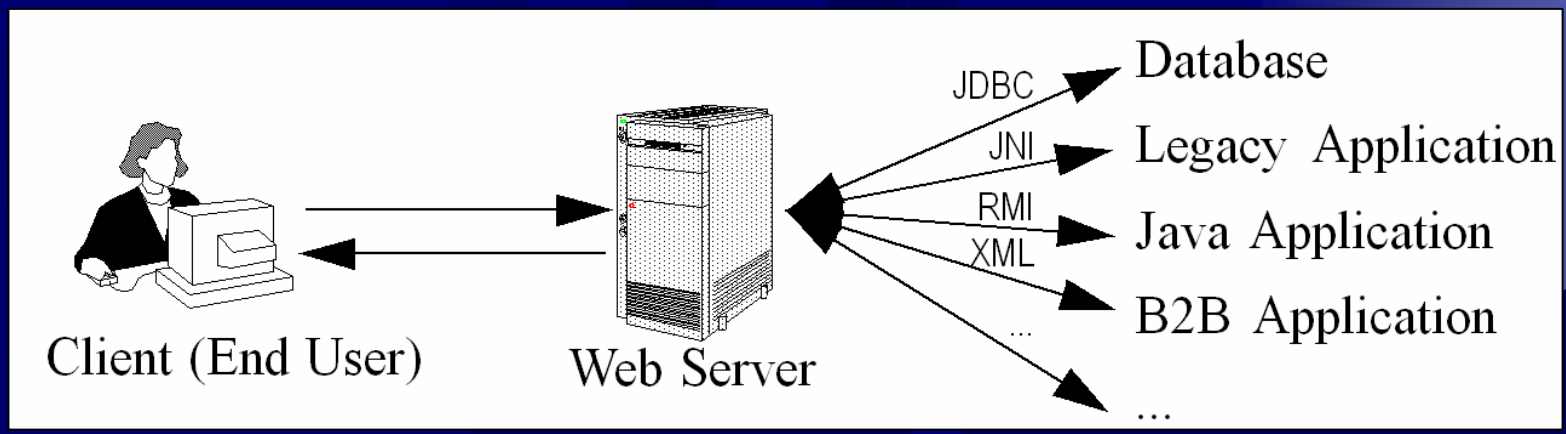
- Big applets require long download time
- Applets do not have access to all the system resources
- Server-side Java solves problems that applets face
  - Code executed on the server side and only the results sent to client
  - Servlets can access legacy applications and data sources

Java Servlet

- Servlets are generic extensions to Java-enabled servers
- Servlets are secure, portable, and easy to use replacement for CGI
- ***Servlet is a dynamically loaded module that services requests from a Web server***
- Servlets are executed within the Java Virtual Machine
- Because the servlet is running on the server side, it does not depend on browser compatibility

A Servlet's Job

- Read explicit data sent by client (form data)
- Read implicit data sent by client (request headers)
- Generate the results
- Send the explicit data back to client (HTML)
- Send the implicit data to client (status codes and response headers)



Execution of Java Servlet

- Applications of Java Servlets
  - Building e-commerce store fronts
    - Servlet builds an online catalog based on the contents of a database
    - Customer places an order, which is processed by another servlet
  - Servlets as wrappers for legacy systems
  - Servlets interacting with EJB applications

Java Servlet Alternatives

- CGI – Common Gateway Interface
  - New process for every cgi request
    - Slow response time
    - If cgi program terminates before responding to web server, the browser just waits for a response until it times out
- Proprietary APIs
  - NSAPI – Netscape Server API
  - ISAPI – IIS Server API
    - Dynamic link libraries
- Server-Side JavaScript
  - Embedding javascript into precompiled HTML pages – only few servers support it

Advantages of Servlets

- Efficiency
  - More efficient – uses lightweight java threads as opposed to individual processes
- Persistency
  - Servlets remain in memory
  - Servlets can maintain state between requests
- Portability
  - Since servlets are written in Java, they are platform independent
- Robustness
  - Error handling, Garbage collector to prevent problems with memory leaks
  - Large class library – network, file, database, distributed object components, security, etc.

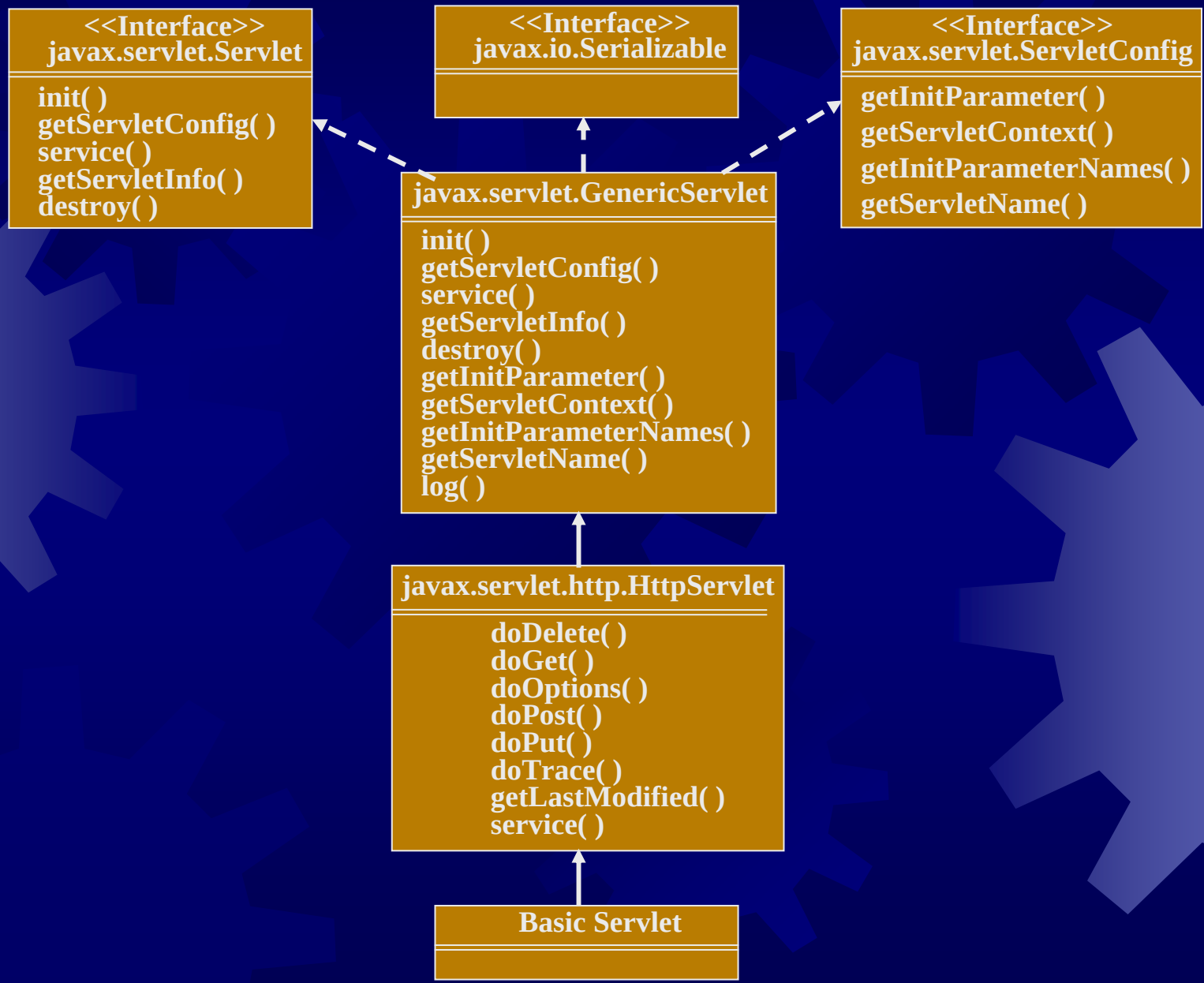
Advantages of Servlets

- Extensibility
  - Creating new subclasses that suite your needs
    - Inheritance, polymorphism, etc.
- Security
  - Security provided by the server as well as the Java Security Manager
  - Eliminates problems associated with executing cgi scripts using operating system “shells”
- Powerful
  - Servlets can directly talk to web server
  - Facilitates database connection pooling, session tracking etc.
- Convenient
  - Parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, etc.

Java Servlet Architecture

- Two packages make up the servlet architecture
  - *javax.servlet*
    - Contains generic interfaces and classes that are implemented and extended by all servlets
  - *javax.servlet.http*
    - Contains classes that are extended when creating HTTP-specific servlets
- The heart of servlet architecture is the interface class *javax.servlet.Servlet*
- It provides the framework for all servlets
- Defines five basic methods – init, service, destroy, getServletConfig and getServletInfo

Object model of Servlet Framework



GenericServlet & HttpServlet

- *HttpServlet* class is extended from *GenericServlet* class
- *GenericServlet.service()* method has been defined as an abstract method
- The two objects that the service() method receives are *ServletRequest* and *ServletResponse*
- ServletRequest Object
  - Holds information that is being sent to the servlet
- ServletResponse Object
  - Holds data that is being sent back to the client

GenericServlet & HttpServlet

- Unlike the *GenericServlet*, when extending *HttpServlet*, don't have to implement the *service()* method. It is already implemented for you
- When *HttpServlet.service( )* is invoked, it calls *doGet( )* or *doPost( ),* depending upon how data is sent from the client
- *HttpServletRequest* and *HttpServletResponse* classes are just extensions of *ServletRequest* and *ServletResponse* with HTTP-specific information stored in them

Life Cycle of a Servlet

- Applet life cycle methods: *init( ), start( ), paint( ), stop( ), and destroy( )* – appropriate methods called based on user action
- Similarly, servlets operate in the context of a request and response model managed by a servlet engine
- The engine does the following
  - Loads the servlet when it is first requested
  - Calls the servlet's ***init( )*** method
  - Handles any number of requests by calling the servlet's ***service( )*** method
  - When shutting down, calls each servlet's ***destroy( )*** method

Life Cycle – *init( )* method

- Request for a servlet received by the servlet engine
- Checks to see if the servlet is already loaded
- If not, uses a class loader to get the required servlet class and instantiates it by calling the constructor method
- After the servlet is loaded, but before it services any requests, the *init ( )* method is called
- Inside *init( ),* the resources used by the servlet are initialized. E.g: establishing database connection
- This method is called only once just before the servlet is placed into service
- The init( ) method takes a *ServletConfig* object as a parameter
- Most common way of doing this is to have it call the *super.init( )* passing it the *ServletConfig* object

Life Cycle – *service( )* method

- **The *service( )* method handles all requests sent by a client**
- **It cannot start servicing requests until the *init( )* method has been executed**
- **Only a single instance of the servlet is created and the servlet engine dispatches each request in a single thread**
- **The *service( )* method is used only when extending *GenericServlet* class**
- **Since servlets are designed to operate in the HTTP environment, the *HttpServlet* class is extended**
- **The *service(HttpServletRequest, HttpServletResponse)* method examines the request and calls the appropriate *doGet()* or *doPost()* method.**
- **A typical Http servlet includes overrides to one or more of these subsidiary methods rather than an override to *service()***

Life Cycle – *destroy( )* method

- This method signifies the end of a servlet's life
- The resources allocated during init( ) are released
- Save persistent information that will be used the next time the servlet is loaded
- The servlet engine unloads the servlet
- Calling destroy( ) yourself will not acutally unload the servlet. Only the servlet engine can do this

Extending the Power of Servlets: JavaServer Pages (JSP)

- Idea:
  - Use regular HTML for most of page
  - Mark dynamic content with special tags
  - Details in second half of course

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
<H1>Welcome to Our Store</H1>
<SMALL>Welcome,
<!-- User name is "New User" for first-time visitors -->
<%= Utils.getUserNameFromCookie(request) %>
To access your account settings, click
<A HREF="Account-Settings.html">here.</A></SMALL>
<P>
Regular HTML for  rest of on-line store's Web page
</BODY></HTML>
```

Compiling and Invoking Servlets

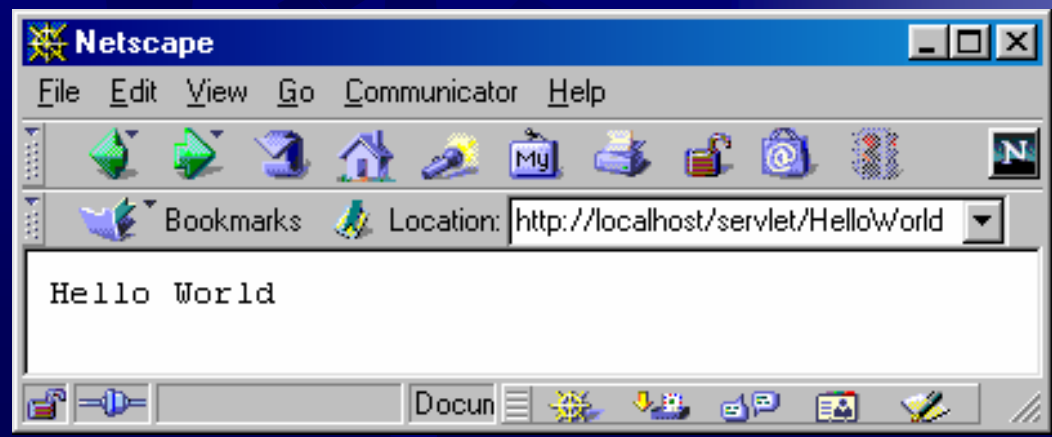
- Set your CLASSPATH
  - Servlet JAR file (e.g., *install_dir*\lib\common\servlet.jar).
  - Top of your package hierarchy
- Put your servlet classes in proper location
  - Locations vary from server to server. E.g.,
    - *tomcat_install_dir*\webapps\examples\WEB-INF\classes
    - *jrun_install_dir*\servers\default\default-app\WEB-INF\classes
  - Invoke your servlets
  - http://*host*/servlet/*ServletName*
  - Custom URL-to-servlet mapping (via web.xml)

Simple Servlet Template

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

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (query data)

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

A Simple Servlet That Generates Plain Text

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

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```



Generating HTML

- Set the Content-Type header
  - Use response.setContentType
- Output HTML
  - Be sure to include the DOCTYPE
- Use an HTML validation service
  - http://validator.w3.org/
  - http://www.htmlhelp.com/tools/validator/
  - If your servlets are behind a firewall, you can run them, save the HTML output, and use a file upload form to validate.

A Servlet That Generates HTML

```
public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello
   WWW</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Packaging Servlets

- Move the files to a subdirectory that matches the intended package name
  - For example, the author uses the coreservlets package for most of the rest of the servlets. So, the class files need to go in a subdirectory called coreservlets.
- Insert a package statement in the class file
  - E.g., top of HelloWWW2.java:
    package coreservlets;
- Set CLASSPATH to top-level directory
  - E.g., C:\Servlets+JSP.
- Include package name in URL
  - http://localhost/servlet/coreservlets.HelloWWW2

Some Simple HTML-Building Utilities

```
public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";

  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }
  ...
}
```

- **Don’t go overboard**
  - **Complete HTML generation packages usually work poorly**
  - **The JSP framework is a better solution**

HelloWWW with ServletUtilities

```
package coreservlets;

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

public class HelloWWW3 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(ServletUtilities.headWithTitle("Hello WWW") +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

HelloWWW Result

Servlet Life Cycle Summary

- init
  - Executed once when the servlet is first loaded.
    Not called for each request.
- service
  - Called in a new thread by server for each request.
    Dispatches to doGet, doPost, etc.
    Do not override this method!
- doGet, doPost, doXxx
  - Handles GET, POST, etc. requests.
  - Override these to provide desired behavior.
- destroy
  - Called when server deletes servlet instance.
    Not called after each request.

Why You Should *Not* Override service

- You can add support for other services later by adding doPut, doTrace, etc.
- You can add support for modification dates by adding a getLastModified method
- The service method gives you automatic support for:
  - HEAD requests
  - OPTIONS requests
  - TRACE requests
- Alternative: have doPost call doGet

Initializing Servlets

- Common in real-life servlets
  - E.g., initializing database connection pools.
- Use ServletConfig.getInitParameter to read initialization parameters
- Set init parameters in web.xml (ver 2.2/2.3)
  - …/WEB-INF/web.xml
  - Many servers have custom interfaces to create web.xml
- It is common to use init even when you don’t read init parameters
  - See modification date example in *Core Servlets and JavaServer Pages* Chapter 2

A Servlet That Uses Initialization Parameters

```
public class ShowMessage extends HttpServlet {
  private String message;
  private String defaultMessage = "No message.";
  private int repeats = 1;

  public void init() throws ServletException {
    ServletConfig config = getServletConfig();
    message = config.getInitParameter("message");
    if (message == null) {
      message = defaultMessage;
    }
    try {
      String repeatString =
        config.getInitParameter("repeats");
      repeats = Integer.parseInt(repeatString);
    } catch(NumberFormatException nfe) {}
  }
```

ShowMessage Servlet (Continued)

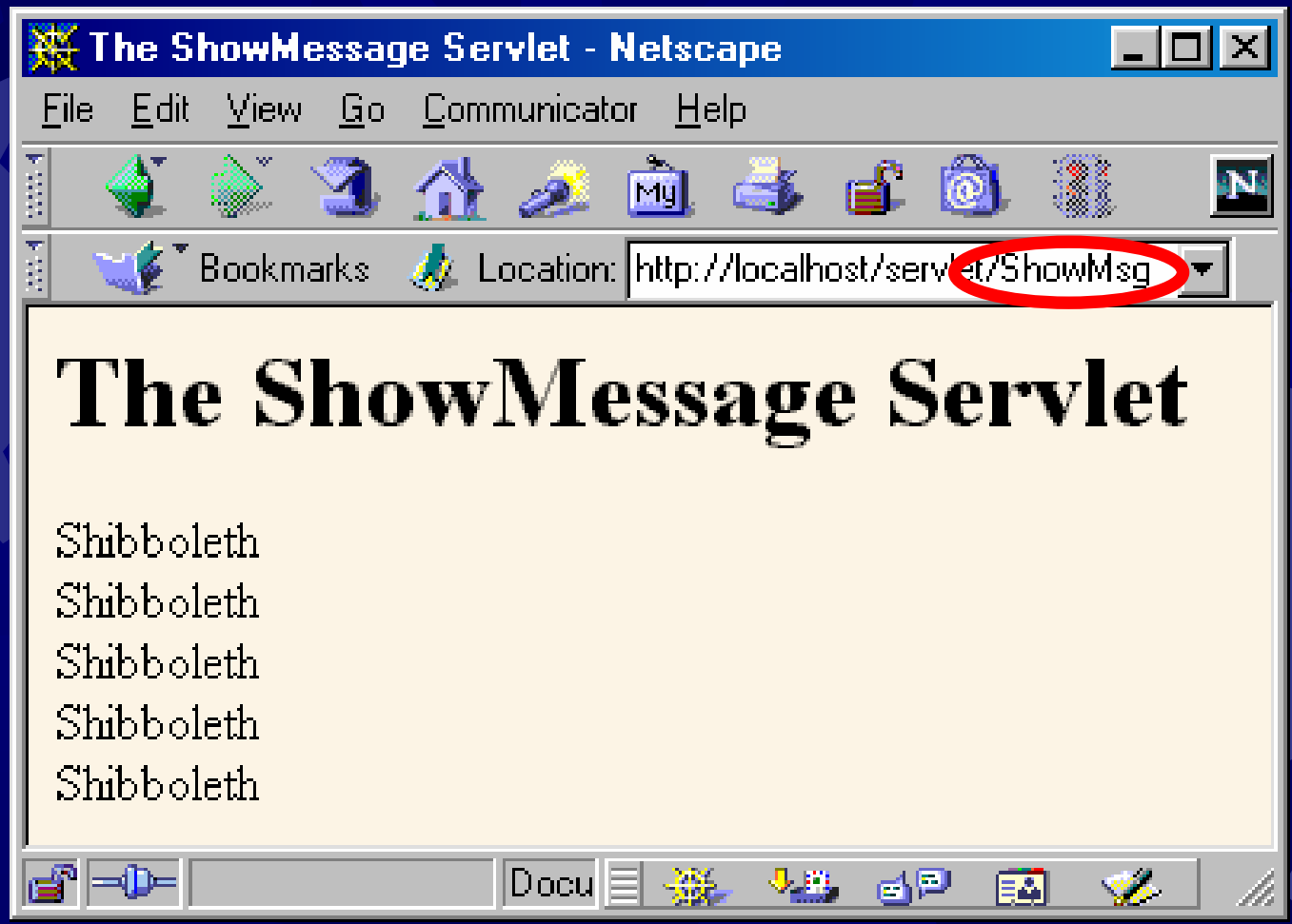
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String title = "The ShowMessage Servlet";
  out.println(ServletUtilities.headWithTitle(title)+
              "<BODY BGCOLOR=\"#FDF5E6\">\n" +
              "<H1 ALIGN=CENTER>" + title + "</H1>");
  for(int i=0; i<repeats; i++) {
    out.println(message + "<BR>");
  }
  out.println("</BODY></HTML>");
}
}
```

Setting Init Parameters (Servlets 2.2/2.3)

- ...\WEB-INF\web.xml
  - *tomcat_install_dir*\webapps\examples\WEB-INF\web.xml
  - See *More Servlets & JSP* (www.moreservlets.com) for details on web.xml

```
<web-app>
  <servlet>
    <servlet-name>ShowMsg</servlet-name>
    <servlet-class>coreservlets.ShowMessage</servlet-class>
    <init-param>
      <param-name>message</param-name>
      <param-value>Shibboleth</param-value>
    </init-param>
    <init-param>
      <param-name>repeats</param-name>
      <param-value>5</param-value>
    </init-param>
  </servlet>
</web-app>
```

ShowMessage Result



Debugging Servlets

- Use print statements; run server on desktop
- Integrated debugger in IDE
- Look at the HTML source
- Return error pages to the client
  - Plan ahead for missing or malformed data
- Use the log file
  - log("message") or log("message", Throwable)
- Look at the request data separately.
  - See EchoServer at www.coreservlets.com
- Look at the response data separately
  - See WebClient at www.coreservlets.com
- Stop and restart the server