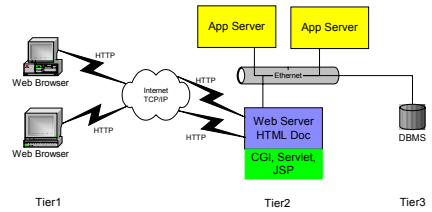


# Introduction to Internet Computing



## Typical Web Application



Presentation layer: CGI, Servlet, and JSP

Business logic layer: EJB, CORBA, COM+

## Focus of This Course

- ◆ June: Presentation layer techniques
  - Java servlets, Java Server Pages (JSP)
- ◆ July: Business layer
  - Enterprise JavaBeans
- ◆ Projects: Design and develop Web applications, like online stores, using servlets, JSP, and EJB

## Hypertext Transaction Protocol

- ◆ HTTP is an application protocol that allows Web clients and servers to communicate
- ◆ Main function of HTTP is to download HTTP documents specified by URLs
- ◆ HTTP is built on top of TCP/IP
- ◆ HTTP is stateless
- ◆ HTTP is Web's version of RPC
- ◆ Current dominant version is HTTP 1.0

## HTTP Request

- ◆ A Web client sends an HTTP request to a Web server specified by an URL
- ◆ An HTTP request has a request line, a few request header fields, and an entity body

```
<method><resource ID><HTTP version>
[<header> : <value>]
.....
[<header> : <value>]
(blank line)
[Entity body]
```

## HTTP Request Example

```
GET /path/file.html HTTP/1.0
Accept: text/html
Accept: audio/x
User-agent: MacWeb
```

## HTTP Response

- ◆ Web server processes the HTTP request, and sends back an HTTP response
- ◆ HTTP response has a response header line, a few response header fields, and an entity body

```
<HTTP Version><result code>[<explanation>]
[<header> : <value>]
.....
[<header> : <value>]
      (blank line)
[Entity body]
```

## HTTP Response Example

```
HTTP/1.0 200 OK
Server: NCSA/1.3
Mime_version: 1.0
Content_type: text/html
Content_length: 2000
```

```
<HTML>
.....
</HTML>
```

## Multipurpose Internet Mail Extensions

- ◆ MIME was designed to support extensible mechanisms for transmitting multimedia email
- ◆ HTTP uses MIME to specify its contents type
- ◆ Common HTTP content-types
  - Text/html
  - Text/plain
  - Image/gif
  - Image/jpeg
  - Video/mpeg

## Composition of a URL

- ◆ A sample URL may look like  
`http://www.server/cgi-bin/myScript.cgi?name=L&oid=21`
- ◆ `http://www.server` identifies a Web server
- ◆ `/cgi-bin/myScript.cgi` identifies a resource on the Web server
- ◆ The substring after the ? mark is the query string. It was mainly designed to support database queries. But it is also used to pass name/value pairs to the Web server

## GET

- ◆ GET is mainly for retrieving a file from a Web server
- ◆ When you enter a URL into the address field of a Web browser, or when you click on a hyperlink, the Web browser issues a GET HTTP request to the target HTTP server
- ◆ The URL for a GET may contain a query-string, which may be used to pass arguments
- ◆ The documents retrieved by GETs are usually cached

## GET Example

- ◆ Submit data from a form

```
<html>
<body>
<form method=get action="http://server/cgi-bin/script">
  Enter your name: <input type="text" name="user">
  <input type="submit" name="submit" value="Submit">
</form>
</body>   Enter your name:  
</html>
```

- ◆ After entering "Ada" and clicking on Submit, URL becomes  
`http://server/cgi-bin/script?user=Ada`

## Drawbacks of GET

- ◆ Even though GET is the default HTTP method, it should not be used to submit form data for two reasons
  - All user data goes to URL query string
  - It is hard to control how much data a user will submit
    - All user data will be submitted through a `query_string` environment variable, which has limited memory capacity
    - If user data is larger than environment memory capacity, the data may be truncated, or the server may crash

## POST

- ◆ POST is similar to GET, but user data is transmitted to the Web server in the entity body. The processor (CGI) can read these data through `System.in`
- ◆ It is safer to use POST to submit data to a Web server
- ◆ If POST is used, environment variable `query_string` will not be set on the server

## How to Support a Session?

- ◆ Typical e-commerce applications need to maintain user data for a user session
- ◆ HTTP is stateless: successive GET/POST calls don't share information through HTTP: a socket is opened to connect to the server, submit data, process data, and return data. Then socket connection terminates
- ◆ Two basic patches to solve the problem
  - Hidden form fields
  - Cookie

## Hidden Form Fields

- ◆ Inside an HTTP form, some data fields are declared and made invisible
- ◆ Session data is stored in hidden fields
- ◆ Web browsers submit form data to Web server, the server set value of some hidden fields of the following form, and send the new form back to the Web browser
- ◆ Main problem: insecure. Hackers can use View Source to read the values of the hidden fields, and modify them

## Cookies

- ◆ A cookie is a small piece of session data to be stored on client's machine
- ◆ After a Web browser submits data to a Web server, the server may create a sequence of cookies in form of a sequence of name=value pairs. Server may set the max age of the cookies
- ◆ The cookies are returned to the Web browser as part of the response, and saved in files on the client machine

## Cookies ...

- ◆ When the client makes the next request to the same Web server, all previous cookies sent over by this server in this session will be automatically sent back to the server
- ◆ When the client shuts down the Web browser, or if a cookie's life has come to the end, the cookie will be deleted from the client machine
- ◆ Main problems
  - Insecure
  - Inefficient

## Web Server Extensions

- ◆ Web server by itself is only responsible to serve file requests
- ◆ CGI and Servlet are simple extensions to Web servers
- ◆ If a request needs data processing, the Web server will forward the request to a CGI or servlet
- ◆ CGI or servlet processes the input data, and dynamically generates an HTML file to let the Web server send back to the client Web browser
- ◆ CGI and Servlet are two standard interfaces for Web server to communicate with Web extensions

## Common Gateway Interface (CGI)

- ◆ Each Web server has its own way of identifying requests for CGI applications
  - URL contains "/cgi-bin"
  - The target application has extension ".cgi"
- ◆ For each CGI request, the Web server sets values for a standard set of environment variables including `query_string`, `server_port`, `request_method`, `content_length`, and `remote_host`

## Common Gateway Interface ...

- ◆ A CGI application is a script or executable that reads request entity body through standard system input stream, reads value of environment variables, composes an HTML file, and sends the file through standard system output stream
- ◆ Main problem: Each CGI request starts a new server process, slow and not scalable

## Servlets

- ◆ Servlets
  - Like applets running in a container, but
    - Execute on server's machine, supported by most Web servers
  - Communication with client via HTTP protocol
    - Client sends HTTP request
    - Server receives request, servlets process it
    - Results returned (HTML document, images, binary data)

## The Servlet API

- ◆ **Servlet** interface
  - Implemented by all servlets
  - Many methods invoked automatically by servlet container
    - Similar to applets (`paint`, `init`, `start`, etc.)
  - **abstract** classes that implement **Servlet**
    - `GenericServlet` (`javax.servlet`)
    - `HTTPServlet` (`javax.servlet.http`)
  - My example extends `HTTPServlet`
- ◆ **Methods**
  - `void init( ServletConfig config )`
    - Automatically called, argument provided

## The Servlet API ...

- ◆ **Methods**
  - `ServletConfig getServletConfig()`
    - Returns reference to config object, gives access to config info
  - `void service ( ServletRequest request, ServletResponse response )`
    - Key method in all servlets
    - Provide access to input and output streams
      - Read from and send to client
  - `void destroy()`
    - Cleanup method, called when servlet exiting

## HttpServlet Class

### ◆ HttpServlet

- Base class for Web-based servlets
- Overrides method **service**
  - Request methods:
    - **GET** - retrieve HTML documents or image
    - **POST** - send server data from HTML form
- Methods **doGet** and **doPost** respond to **GET** and **POST**
  - Called by **service**
  - Receive **HttpServletRequest** and **HttpServletResponse** objects

## HttpServletRequest Interface

### ◆ HttpServletRequest interface

- Object passed to **doGet** and **doPost**
- Extends **ServletRequest**

### ◆ Methods

- **String** **getParameter( String name )**
  - Returns value of parameter **name**
- **Enumeration** **getParameterNames( )**
  - Returns names of parameters
- **String[]** **getParameterValues( String name )**
  - Returns array of strings containing values of a parameter
- **Cookie[]** **getCookies( )**
  - Returns array of **Cookie** objects, can identify client

## HttpServletResponse Interface

### ◆ HttpServletResponse

- Object passed to **doGet** and **doPost**
- Extends **ServletResponse**

### ◆ Methods

- **void** **addCookie( Cookie cookie )**
  - Add **Cookie** to header of response to client
- **ServletOutputStream** **getOutputStream( )**
  - Gets byte-based output stream, send binary data to client
- **PrintWriter** **getWriter( )**
  - Gets character-based output stream, send text to client
- **void** **setContentType( String type )**
  - Specify MIME type of the response
  - Helps display data

## Downloading the Java Servlet Development Kit

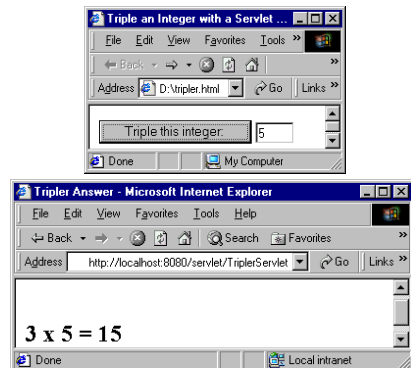
### ◆ Programming with servlets

- Download Java Servlet Development Kit (JSDK)  
<http://java.sun.com/products/servlet/index.htm>  
1
- After downloading, install and read **README.txt**
  - Setup of JSDK and how to start server to test servlets
- Copy **javax.servlet.jar** (has JSDK class files) from install directory to JDK extensions directory
  - `c:\jdk1.3\jre\lib\ext`

## A Servlet for Tripling Integers

- ◆ The client enters an integer in a text field, and then clicks on button "Triple this integer"
- ◆ The Web browser sends a POST request to the specified Web server with the integer
- ◆ Web server invokes the servlet to triple the integer, and passes back the response from the servlet

## Running Triple Servlet



## Triple.html

```
<html>
<head>
<title>Triple an Integer with a Servlet</title>
</head>
<body>
<form method="post"
  action="http://localhost:8080/servlet/TriplerServlet">
  <input type="submit" value="Triple this integer: ">
  <input type="text" name="number" value="0" size="4">
</form>
</body>
</html>
```

## TriplerServlet.java

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

public class TriplerServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    String value = request.getParameter("number");
    int n = java.lang.Integer.valueOf(value).intValue();
    response.setContentType("text/html");
    PrintWriter output = response.getWriter();
```

## TriplerServlet.java ...

```
StringBuffer buf = new StringBuffer();
buf.append("<html>\n");
buf.append("<title>Tripler Answer</title>\n");
buf.append("<br><br><h2>");
buf.append("3 x " + n + " = " + 3*n);
buf.append("</h2>\n</html>");
output.println(buf.toString());
output.close();
}
```

## Running Servlets

- ◆ Must be running on a server
  - Check documentation for how to install servlets
- ◆ JSDK Webserver
  - Test servlets
  - Assumes `.class` files for servlets in subdirectory
    - `webpages\WEB-INF\servlets`
  - `startserver.bat` (`startserver` shell script)

## HttpSession

- ◆ In addition to using cookies to maintain session data on client machines, servlets also support HttpSession
- ◆ Each client session can be associated with a unique HttpSession on the Web server, which is basically a hash table of name/value pairs

## Observations

- ◆ CGI and servlet are both simple extensions to Web servers to provide dynamic Web pages
- ◆ CGI and servlet are not suitable to build peer-to-peer distributed systems
- ◆ Since CGI and servlet invocations go through Web server, efficiency is low

## Remote Method Invocations (RMI)

- ◆ Remote Method Invocation (RMI)
  - Allows remote method calls
    - Objects in different programs can communicate
    - Method calls appear as normal
  - Based on Remote Procedure Calls (RPC)
    - Developed in 1980's
    - Allows procedural program (like C) to call function on another computer
    - Performs networking and marshalling of data (packaging arguments and return values)
    - Not compatible with objects
    - Interface Definition Language required - describe functions
  - RMI is Java's implementation of RPC

## RMI ...

- ◆ RMI
  - Register objects as remotely accessible
    - Client can look up objects and receive a reference
    - Use reference to call method
    - Syntax same as a normal method call
  - Marshalling of data
    - Can transfer objects as well
    - Class `ObjectOutputStream` converts `Serializable` object into stream of bytes
      - Transmit across network
    - Class `ObjectInputStream` reconstructs object
  - No Interface Definition Language needed
    - Use Java's own interface

## Creating a Distributed System with RMI

- ◆ Four major steps
  - Define remote interface
    - Describes client/server communication
  - Define server application to implement remote interface
    - Same name as remote interface, ends with `Impl`
  - Define client application that uses remote interface reference
    - Interacts with server implementation
  - Compile and execute server and client

## Triple Integers

- ◆ Our example server has one public method  
`int triple(int);`  
to triple the argument and return the result

## Defining the Remote Interface

- ◆ First step
  - Define remote interface that describes remote methods
    - Client calls remote methods, server implements them
- ◆ To create a remote interface
  - Define interface that extends interface `Remote` (`java.rmi`)
    - Tagging interface - no methods to define
    - Objects of `Remote` classes can be exported
  - Each method in `Remote` interface must **throw** `RemoteException`
    - Potential network errors

## Tripler.java

```
import java.rmi.*;

public interface Tripler extends Remote {
    public int triple(int v) throws RemoteException;
}
```

## TriplerImpl.java

```
import java.rmi.*;
import java.rmi.server.*;

public class TriplerImpl extends UnicastRemoteObject
    implements Tripler {
    public TriplerImpl() throws RemoteException {
        super();
    }

    public int triple(int v) {
        System.err.println("Triple " + v);
        return 3*v;
    }
}
```

## TriplerImpl.java ...

```
public static void main(String[] args) throws Exception {
    System.err.println("Initializing server, ...");
    TriplerImpl server = new TriplerImpl();
    Naming.rebind("//localhost/Tripler", server);
    System.err.println("Tripler server is up and running.");
}
}
```

## Client.java

```
import java.rmi.*;

public class Client {
    public static void main(String[] args) throws Exception {
        String host = "localhost";
        if (args.length > 1)
            host = args[1];
        String objectName = "/" + host + "/Tripler";
        Tripler server = (Tripler)Naming.lookup(objectName);
    }
}
```

## Client.java ...

```
int triplee;
if (args.length > 0)
    triplee = java.lang.Integer.valueOf(args[0]).intValue();
else {
    triplee = 12;
    System.out.println("Using default value of " + triplee);
}
int answer = server.triple(triplee);
System.out.println(triplee + " * 3 = " + answer);
}
}
```

## Compile and Execute the Server and the Client

- ◆ Build and execute application
  - Compile classes with **javac**
  - Remote server class (**TriplerImpl**) compiled with **rmic** compiler
    - Makes a stub class - allows client to access remote methods
    - Gets remote method calls, passes to RMI system, which performs networking
    - **rmic -v1.2 TriplerImpl**
      - Older versions create now unnecessary skeleton class

## Compile and Execute the Server and the Client ..

- ◆ Start **rmiregistry**
  - Type **rmiregistry** at command window
    - No text in response
- ◆ Must bind remote server object
  - Run TriplerImpl application
    - java TriplerImpl**
    - Superclass **UnicastRemoteObject**
      - Constructor exports remote object
      - **main** binds object to **rmiregistry**
      - **rmiregistry** provides host and port number to clients



