CICS 515 a
Internet Programming
Week 6

Mike Feeley

Java Server Pages

Java in the Web Server

- tomcat
  - written in Java
  - integrated with Apache (or with server embedded in it)
  - implements Java Server Pages and Java Servlets
- separate configuration
  - application directory structure under webapp subdirectory
  - xml configuration files (context.xml and web.xml)
- we'll access it on port 9123 or something else ...
  - configure when you start tomcat

Java Server Pages

- PHP-like server-side scripting in real Java
- each JSP document is a class with
  - standard, unmodified HTML, as in PHP
  - Java directives <%@ ... %>
  - Java declarations <%! ... %>
  - Java code <% ... %>
  - Java expressions <%= ... %>
  - Java-Expression-Language expressions ${ ... }
- outputting data to response document
  - out.println ()

JSP example

- http://people.cs.ubc.ca:9123/nodb/foo.jsp

```
<html>
<body>
</body>
<%@ page import="java.io.PrintWriter" %>
<%
  PrintWriter outf = new PrintWriter(out);
  out.println ("<table border=2>");
  for (int i=0; i<2; i++) {
    out.println ("<tr>");
    outf.printf ("<td>%d</td><td>%d</td>\n",i,i*10);
    out.println ("</tr>");
  }
  out.println ("</table>");
%>
</html>
```

- in .../apache-tomcat-6.0.13/webapps/nodb

Accessing the database

in Java using JDBC

- java's interface to the standard database access protocol (ODBC)

```
<%@ page import="java.net.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
```

```
Class.forName ("com.mysql.jdbc.Driver").newInstance ();
Connection conn = DriverManager.
  getConnection ("jdbc:mysql://"+dbHost+"/"+dbName, dbUser, dbPass);
Statement stmt = conn.createStatement ();
ResultSet rows = stmt.executeQuery ("SELECT * FROM student ORDER BY sid");
```

JSP Database Example 1

- http://people.cs.ubc.ca:9123/student/listStudents.jsp

```
<html>
<body>
<%@ page import="java.net.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
<%!
  final static String dbHost = "localhost";
  final static String dbUser = "feeley";
  final static String dbPass = "feeley";
  final static String dbName = "feeley_database";
%>
<%
try {
  Class.forName ("com.mysql.jdbc.Driver").newInstance ();
  Connection conn = DriverManager.
    getConnection ("jdbc:mysql://"+dbHost+"/"+dbName, dbUser, dbPass);
  Statement stmt = conn.createStatement ();
  ResultSet rows = stmt.executeQuery ("SELECT * FROM student ORDER BY sid");
  out.println ("<table border=2>");
  out.println ("<tr><td><b>SID</b></td><td><b>Name</b></td></tr>");
  while (rows.next ()) {
    out.println ("<tr><td>"+rows.getString("sid")+"</
    td><td>"+rows.getString("name")+"</td></tr>\n");
  }
  conn.close ();
} catch (Exception e) {
  out.println ("error<br>");
  e.printStackTrace ();
  throw e;
}
%>
</body>
</html>
```

JSP Database Example 2

- http://people.cs.ubc.ca:9123/student/listStudents1.jsp

```
ResultSetMetaData rowsMeta = rows.getMetaData ();
out.println ("<tr>");
for (int i=1; i<=rowsMeta.getColumnCount (); i++)
  out.println ("<td><b>"+rowsMeta.getColumnName (i)+"</b></td>");
out.println ("</tr>");

while (rows.next ()) {
  out.println ("<tr>");
  for (int i=1; i<=rowsMeta.getColumnCount (); i++)
    out.println ("<td>"+rows.getString (i)+"</td>");
  out.println ("</tr>");
```

Java Beans

- a standard template for writing Java classes
  - designed to make it easy for bean classes to be re-used
  - for example, Ecplise's extension mechanism is build on beans
  - and, JSP (and later Java Server Faces) use beans
- standard beans and enterprise beans
  - the bean standard has different levels of requirement and applicability
  - we'll just look at very simple, standard beans
- the standard bean template is just setters and getters
  - beans have properties
  - every property has a getPropertyName and setPropertyName method
  - JSP and tools access the bean via a property view instead of calling methods
    - provides a simple way for JSP page to invoke methods in java class (the bean)
    - often used for forms

Defining a Java Bean class

- Bean.java

```
package Lecture6;

public class Bean {
  String string;
  public Bean()                          { string = "default value"; }
  public void setStr (String aString)    { string = aString; }
  public String getStr ()                { return string; }
}
```

- put class in webapps/<appdir>/classes/<package path>
  - e.g., /usr/local/apache-tomcat-6.0.13/webapps/nodb/classes/Lecture6/Bean.class
- bean is accessible from JSP stored in its parent app dir
  - in this case its .../webapps/nodb
  - subdirectories are okay too
  - but, for security reasons JSP usually prohibits access to beans outside JSP's directory tree

Accessing Bean from Java Server Page

- importing the class

```
<%@ page import="Lecture6.Bean" %>
```

- creating an instance of the class (an object)

```
<jsp:useBean id='bean0' class='Lecture6.Bean' scope='session'/>
```

- getting and setting properties

```
<jsp:setProperty name='bean0' property='str' value='Hello'/>
<jsp:getProperty name='bean0' property='str'/>
```

```
<%
out.println ("bean0.getStr() = " + bean0.getStr ());
%>
```

in a form

- jsp:setProperty property='*' sets all properties who's name match form inputs
  - its sort of like what happens with GET or POST in regular HTML
  - but now input values are placed in bean, not in the PHP $_GET or $_POST arrays

```
<jsp:useBean id='bean1' class='Lecture6.Bean' scope='request'/>
<form method='post'>
<input name='str' type='text'/ value='<jsp:getProperty name="bean1" property="str"/>'>
<input name='action' type='hidden' value='submitted'>
</form>
<jsp:setProperty name='bean1' property='*'/>
```

form handling is as in JSP

- use hidden input to see if form has been submitted
- use embedded Java to process form inputs

```
<%
if ("submitted".equals(request.getParameter ("action"))) {
  out.println ("bean1.getStr() = " + bean1.getStr ());
}
%>
```

http://people.cs.ubc.ca:9123/nodb/Bean.jsp

Java Servlets

Java Servlets

- Java classes that execute in web server
- Better than PHP
  - same interface for calls within server and calls from client
  - client calls to servlet are method invocations, with PHP its a new subprocess each time
  - reduces complexity associated with language heterogenaity
- In Java Enterprise Edition (J2EE): `javax.servlet.*`
  - J2SE includes the standard packages and J2EE includes stuff for *enterprise* computing
  - to build a sevlet, extend `javax.servlet.http.HttpServlet` class to implement
    - `init()`
    - `destroy()`
    - `doGet (HttpServletRequest request, HttpServletResponse response)`
    - `doPost (HttpServletRequest request, HttpServletResponse response)`

Java Servlet Example

- http://people.cs.ubc.ca:9123/student/ListStudents

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

public class ListStudentsServlet extends HttpServlet {

  public void
  doGet (HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
  {
    HttpSession session = request.getSession (true);
    PrintWriter out = response.getWriter ();
    try {
      Student student = (Student) session.getAttribute ("Student");
      if (student==null) {
        student = new Student ();
        session.setAttribute ("Student", student);
      }
      response.setContentType ("text/html");
      student.listStudents (out);
      out.close ();
    } catch (Exception e) {
      out.println ("error<br>");
      e.printStackTrace ();
      throw new ServletException ();
    }
    out.close ();
  }
}
```

```
import java.io.*;
import java.sql.*;

public class Student {
  final static String dbHost = "localhost";
  final static String dbUser = "feeley";
  final static String dbPass = "feeley";
  final static String dbName = "feeley_database";

  Connection conn;

  Student () throws java.lang.ClassNotFoundException, java.lang.InstantiationException,
    java.lang.IllegalAccessException, java.sql.SQLException
  {
    Class.forName ("com.mysql.jdbc.Driver").newInstance ();
    conn = DriverManager.
      getConnection ("jdbc:mysql://"+dbHost+"/"+dbName, dbUser, dbPass);
  }

  public void listStudents (PrintWriter out) throws java.sql.SQLException
  {
    Statement stmt = conn.createStatement ();
    ResultSet rows = stmt.executeQuery ("SELECT * FROM student ORDER BY sid");
    out.println ("<table border=2>");

    ResultSetMetaData rowsMeta = rows.getMetaData ();
    out.println ("<tr>");
    for (int i=1; i<rowsMeta.getColumnCount (); i++)
      out.println ("<td><b>"+rowsMeta.getColumnName (i)+"</b></td>");
    out.println ("</tr>");

    while (rows.next ()) {
      out.println ("<tr>");
      for (int i=1; i<rowsMeta.getColumnCount (); i++)
        out.println ("<td>"+rows.getString (i)+"</td>");
      out.println ("</tr>");
    }
  }
}
```

Servlet deployment

- create subdirectory in tomcat's webapps directory
- put jsp and html etc in this directory
- create another subdirectory called WEB-INF
  - web.xml in this WEB-INF describes servlet

```
<servlet>
  <servlet-name>ListStudents</servlet-name>
  <servlet-class>ca.ubc.cics.cics515.Summer2007.Lecture6.ListStudentsServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ListStudents</servlet-name>
  <url-pattern>/ListStudents</url-pattern>
</servlet-mapping>
```

  - classes subdirectory of WEB-INF stores servlet classes
- configuration details in the wiki (last year's)
  - please add for windows if you like

AJAX and Servlets

accessing servlet from JavaScript

- just change the URL

```
xmlHttp.call ('DBSelect',
  { table: 'blogEntry',
    query: 'SELECT id, author, date, subject, text, replyTo FROM blogEntry ORDER BY id',
    nestKey: 'id', nestOn: 'replyTo' },
  createBlogTable);
```

```
  xmlHttp.call = function (url, args, whenComplete) {
    var xmlDoc = newXMLDoc ();
    var appendElement = function (toElement,tag) {
      var element = xmlDoc.createElement (tag);
      toElement.appendChild (element);
      return element;
    }
    var xmlRoot = appendElement (xmlDoc, 'rpcArgs');
    for (arg in args)
    {
      alert(arg+": "+args[arg]);
      appendElement(xmlRoot, arg).appendChild (xmlDoc.createTextNode(args[arg]));
    }
    this.whenComplete       = whenComplete;
    this.whenError          = xmlCallError;
    this.onreadystatechange = this.whenStateChanges;
    this.open ('POST', url, true);
    this.setRequestHeader ('Content-type', 'text/xml');
    this.send (saveXML(xmlDoc));
  }
  return xmlHttp;
}
```

in Java Servlet

- parse XML input into DOM datastructure
- process input as DOM
- produce output as DOM
- convert DOM into XML and send back in response

structuring with classes

- AjaxServlet
- AjaxServletRequest
- AjaxServletResponse
- AjaxDB
- AjaxDBConnection

then two servlets

- DBSelectServlet
- DBUpdateServlet

Parsing XML in Java

DocumentBuilder

- creates DOM documents and parses XML into them
- create one in init and call ajaxInit so that child classes get to init too

AjaxServlet:

```
final public void init () throws ServletException {
  try {
    docBuilder = (DocumentBuilderFactory.newInstance()).newDocumentBuilder ();
    ajaxInit ();
  } catch (javax.xml.parsers.ParserConfigurationException pce) {
    pce.printStackTrace();
    throw new ServletException ();
  }
}
```

ajax processing for doPost (and doGET)

- create ajax request and reply
- call application doPost (or doGet)
- translate result into XML for httpServletResponse

AjaxServlet:

```
final public void doPost (HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
{
  AjaxServletRequest ajaxRequest   = new AjaxServletRequest (request, docBuilder);
  AjaxServletResponse ajaxResponse = new AjaxServletResponse (response, docBuilder);

  doPost (ajaxRequest, ajaxResponse);

  ajaxResponse.saveXML ();
}
```

docBuilder.parse: XML => DOM

AjaxServletRequest:

```
AjaxServletRequest (HttpServletRequest anHttpServletRequest, DocumentBuilder
 docBuilder)
  throws ServletException, IOException
{
  httpServletRequest = anHttpServletRequest;
  try {
    reqDoc = docBuilder.parse (httpServletRequest.getInputStream());
  } catch (org.xml.sax.SAXException se) {
    throw new ServletException (se.getException());
  }
}
```

child class can get arguments from the reqDoc

AjaxServletRequest:

```
String getParameter (String name) {
  Node paramNode = reqDoc.getElementsByTagName(name).item(0);
  return paramNode!=null? paramNode.getFirstChild().getNodeValue() : null;
}
```

application servlet then looks like this

DBSelectServlet:

```
public void
doPost (AjaxServletRequest request, AjaxServletResponse response)
  throws ServletException, IOException
{
  String table   = request.getParameter ("table");
  String query   = request.getParameter ("query");
  String nestKey = request.getParameter ("nestKey");
  String nestOn  = request.getParameter ("nestOn");
  assert (table!=null && query!=null);
```

Access MySQL database server

- application creates instance of AjaxDB in its init

DBSelectServlet:

```
AjaxDB db = null;

public void ajaxInit () throws ServletException {
  db = new AjaxDB ("localhost","feeley","feeley","feeley_database");
}
```

- servlet post/get uses AjaxDB to allocate a connection

DBSelectServlet.doPost():

```
AjaxDBConnection conn =
  db.getConnection (request.getHttpServletRequest().getSession(true));
```

AjaxDB manages per-session DB connections

AjaxDB:

```
public AjaxDB (String aDbHost, String aDbUser, String aDbPass, String aDbName)
  throws ServletException
{
  dbHost = aDbHost; dbUser = aDbUser; dbPass = aDbPass; dbName = aDbName;
  try {
    Class.forName ("com.mysql.jdbc.Driver").newInstance ();
  } catch (Exception e) {
    throw new ServletException (e);
  }
}
```

AjaxDB:

```
public AjaxDBConnection getConnection (HttpSession session) throws ServletException {
  AjaxDBConnection conn = (AjaxDBConnection) session.getAttribute ("dbConnection");
  if (conn==null) {
    conn = new AjaxDBConnection (dbHost, dbUser, dbPass, dbName);
    session.setAttribute ("dbConnection",conn);
  }
  return conn;
}
```

AjaxDBConnection connects to DB and translates to DOM

AjaxDBConnection:

```
AjaxDBConnection (String dbHost, String dbUser, String dbPass, String dbName)
  throws ServletException
{
  try {
    conn = DriverManager.
      getConnection ("jdbc:mysql://"+dbHost+"/"+dbName, dbUser, dbPass);
  } catch (Exception e) {
    throw new ServletException (e) ;
  }
}
```

```
public Node executeSelect
    (Document doc, String table, String query, String nestKey, String nestOn)
    throws ServletException
  { ... }


public Node executeUpdate
    (Document doc, String table, String update, String key, String replyFields)
    throws ServletException
  { .... }
```

Creating the DOM reply

- create new DOM document for response

AjaxServletResponse:

```
AjaxServletResponse
  (HttpServletResponse anHttpServletResponse, DocumentBuilder docBuilder)
{
  httpServletResponse = anHttpServletResponse;
  doc = docBuilder.newDocument ();
}
```

- application call to db connection to execute query

DBSelectServlet.doPost():

```
Document rDoc = response.getDocument ();
rDoc.appendChild (conn.executeSelect (rDoc, table, query, nestKey, nestOn));
```

perform DB query and get info from Result Set

AjaxDBConnection:

```
public Node executeSelect
  (Document doc, String table, String query, String nestKey, String nestOn)
  throws ServletException
{
  try {

    ResultSet rs = conn.createStatement().executeQuery (query);

    ResultSetMetaData rsMd = rs.getMetaData ();
    int rsColumnCount = rsMd.getColumnCount ();
    String rsColumnName [] = new String [rsColumnCount+1];
    int nestKeyCol=0, nestOnCol=0;
    for (int i=1; i<=rsColumnCount; i++) {
      rsColumnName [i] = rsMd.getColumnName (i);
      if (rsColumnName[i].equals(nestKey))
        nestKeyCol = i;
      if (rsColumnName[i].equals(nestOn))
        nestOnCol = i;
    }

  ...
```

build DOM reply (without nesting)

AjaxDBConnection.executeSelect():

```
...

Node rootElmt = doc.createElement (table+"s");
while (rs.next()) {
  Node rowElmt = doc.createElement (table);
  for (int i=1; i<=rsColumnCount; i++) {
    Node colElmt = rowElmt.appendChild (doc.createElement(rsColumnName[i]));
    colElmt.appendChild (doc.createTextNode(rs.getString(i)));
  }
  rootElmt.appendChild (rowElmt);
}

return rootElmt;

...
```

build DOM reply with nesting

AjaxDBConnection.executeSelect():

```
...

Hashtable<String,Node> nestHashtable = new Hashtable<String,Node> ();
Node rootElmt = doc.createElement (table+"s"), parentElmt = rootElmt;
while (rs.next()) {
  Node rowElmt = doc.createElement (table);
  for (int i=1; i<=rsColumnCount; i++) {
    Node colElmt = rowElmt.appendChild (doc.createElement(rsColumnName[i]));
    colElmt.appendChild (doc.createTextNode(rs.getString(i)));
  }
  if (nestKeyCol!=0 && nestOnCol!=0) {
    parentElmt = nestHashtable.get (rs.getString(nestOnCol));
    if (parentElmt==null)
      parentElmt = rootElmt;
    nestHashtable.put (rs.getString(nestKeyCol),rowElmt);
  }
  parentElmt.appendChild (rowElmt);
}

return rootElmt;

...
```

Sending result DOM as XML in resposne

- client has appended result DOM response document
  - from earlier slide

DBSelectServlet.doPost():

```
Document rDoc = response.getDocument ();
rDoc.appendChild (conn.executeSelect (rDoc, table, query, nestKey, nestOn));
```

- when application doPost returns to AjaxServlet.doPost

AjaxServlet.doPost():

```
final public void doPost (...) ...
{ ...
  doPost (ajaxRequest, ajaxResponse);
  ajaxResponse.saveXML ();
}
```

translate DOM to XML for httpServletResponse

AjaxServletResponse:

```
void saveXML () throws ServletException {
  try {
    TransformerFactory.newInstance().newTransformer().
      transform (new DOMSource(doc), new
     StreamResult(httpServletResponse.getWriter()));
  } catch (Exception e) {
    throw new ServletException (e);
  }
}
```

Updating the database is similar

- to insert, update or delete
  - use executeUpdate() instead of executeQuery()
- to get generated keys following insert
  - use stmt.getGeneratedKeys()

AjaxDBConnection:

```
public Node executeUpdate
  (Document doc, String table, String update, String key, String replyFields)
  throws ServletException
 {
  try {
    Statement stmt = conn.createStatement ();
    stmt.executeUpdate (update);
    if (key!=null && replyFields!=null) {
      ResultSet rs = stmt.getGeneratedKeys();
      rs.next();
      String query = "SELECT "+replyFields+" FROM "+table+
                     " WHERE "+key+"="+rs.getString(1);
      return executeSelect (doc, table, query, null, null);
    } else
      return null;
  } catch (Exception e) {
    throw new ServletException (e);
  }
}
```

Putting it all together

- http://people.cs.ubc.ca:9123/ajax/blog.html

Model - View - Controller

- Design pattern for GUI applications
- Model
  - is part of system that owns the data
  - its the database
  - its all the rules about what is legal in the database (the business rules)
  - its all of the code for read, validating and updating data
  - implemented to be independent from the View or Controller
- View
  - is the part of the system that displays the data for users
  - its all of the display formatting
  - view depends on Controller
- Contoller
  - controller controls the view and is based by the database
  - controller depends on the View

Java Server Faces

- Java support for *model-view-controller (MVC)* design framework
  - an extension of JSP, sort of
- view is the GUI
  - implemented by JSP, described using something like *struts* (sort of a swing alternative)
  - <h:form> instead of <form> <h:inputText> instead of <input>
- model is the backend
  - supplies data and defines constraints on data
  - database information wrapped in *backing* or *entity* Java Beans
- controller connects view and model
  - handles input events and converts them to actions on model
  - input validation, error display, etc.
- Book describes Rails
  - which, similarly implements MVC framework, but in Ruby, not in Java

Javascript toolkits

- Library of javascript code
  - extensive support of GUI widgets
  - many example applications
- Dojo
  - open source toolkit
  - http://www.dojotoolkit.org/

Google Web Toolkit

- AJAX framework
  - most stuff runs in browser
- application written entirely in Java
  - unlike JSP or JSF, which retain the tag-based part of web programming
  - integrated programming environment a big plus
- compiles to Java into JavaScript, HTML and servlets
  - you deal with Java
  - it creates JavaScript and supporting Servlet architecture
  - it builds RPC connection between client JavaScript and server Java
- available at http://code.google.com/webtoolkit/

Summary

- brower programming
  - HTML
  - JavaScript
  - AJAX
  - Java Applets
  - Flash
- server programming
  - PHP
  - Java Server Pages (JSP)
  - Java Servlets
  - Java Server Faces
- trends
  - toward more client-side computation and less at server side
  - toward integrated programming environments