Web Development in Java

Perdita Stevens, University of Edinburgh

August 2010

Agenda

Not necessarily quite in this order:

- From applets to server-side technology (servlets, JSP, XML; XML basics and object serialisation)
- A basic MVC architecture web application with JSP and Struts
- What Java EE 5 adds, compared with Java SE: JDBC, RMI, EJB, JMS, JTA, XML, JCA, JSP, JSTL, JSF, JPA etc., all very briefly!
- The role of a Java EE application server
- EJBs, Spring and Hibernate

... plus all the other TAFLAs I found I had to explain in order to explain that lot...

Warning

I will attempt to give an overview of what technologies are out there and what they are useful for, with pointers to more information.

BUT:

- it's a kilometre wide and a millimetre thick;
- this is not stuff I have experience of using for real.

If you think something I say is misleading, you could well be correct – it's important that you say so!

More information

The single most useful source of information I've found is Oracle's Java EE Tutorial,

`http://java.sun.com/javaee/5/docs/tutorial/doc/`

– including for the basic stuff that's already in Java SE.

For individual technologies Google finds documentation and tutorials quite well; but beware,

- most have gone through multiple versions and there is a lot of outdated info out there;
- the Wikipedia articles are often impenetrable (I've improved a few: do you too!)

HTTP basic

Client sends HTTP Request over network to web server...

... which responds with HTTP Response.

- Not OO
- Stateless: when sessions are needed, can implement these using
  - cookies, or
  - URI rewriting.

Layers

As you consider increasingly complex Java-based web applications you may be concerned with:

- client-side only: web browser displaying (X)HTML web pages received from the “dumb” web server; applets
- client-server: involving only things running on the client machine and things running on the web server’s machine, e.g., to generate dynamic web pages: typically using Java SE
- multi-tier: involving client, web server and other server(s) e.g. database, other systems..., to make arbitrary functionality available via the web: typically using Java EE.

Terminology that’s not specific to Java

EIS: Enterprise Information System - a polite way to say “legacy system”? Sort of...

EAI: Enterprise Application Integration - sticking your legacy systems together

Web application: any application accessed over the web, in any way, e.g. by a web-based GUI

Web services: making your legacy systems available over the web via individual service requests in XML.

HTML and XHTML

Your most basic web page is written in HTML, HyperText Markup Language.

Aberration: HTML is an ad-hoc ill-structured language, hard to work with. So instead often use

XHTML: HTML done properly as an XML language.

BEGIN quick digression on XML:

XML

Tree-structured documents often using IDs to represent more general graphs

Textual, structured, easy to parse.

elements, attributes

Specified using schemas or DTDs.

Object serialization and XML

Recall we discussed serializing and deserializing objects to objectstreams in Java.

Problem: that representation wasn't much use for anything except deserializing later.

If you store object state as XML instead, then other applications can also read it, it can be used to generate human-readable representations, etc.

Downside: verbose, so representations can get large.

JAXP

Java API for XML Processing

provide functionality for reading/writing/manipulating XML data using either:

- DOM, Document Object Model
- SAX, Simple API for XML

plus XSLT.

JAXB

Java Architecture for XML Binding

i.e. binding

- an XML schema (a description of a family of trees of text pieces) – plus some extra information – to
- a collection of Java classes describing a family of trees of objects – suitably annotated.

Both ways round: given the classes, generate the schema, or vice versa.

Supports marshalling/unmarshalling with validation.

END quick digression on XML!

Applets: simplest possible web application

Recall: a Java Applet is a program whose bytecode is downloaded over the web. It runs in a Java Virtual Machine in the user's browser. It runs in a sandbox, pretty much independent of the outside world.

What if that's not enough?

CGI: simplest possible server-side processing

Common Gateway Interface

Typical scenario:

- user fills in some fields in a web form, clicks a button.
- This invokes a program, sending it the user's data.
- The program generates a new HTML (usually) page, which is displayed to the user.

The program would typically be in Perl, but could be in Java or some other language (invoked from a batch/shell/Perl script), using a CGI package.

Uses and limitations of CGI

The CGI program *can* do anything you like, including access databases, etc.

You *can* also use an applet, rather than a simple web form, on the client side to invoke the CGI program when you want to, and in this way do complicated stuff.

But every HTTP request is handled independently: new process, new copy of the CGI program – doesn't scale.

Once you move beyond simple form processing, there is almost certainly a better way....

`http://www.apl.jhu.edu/~hall/java/CGI-with-Java.html` - but old

Servlets

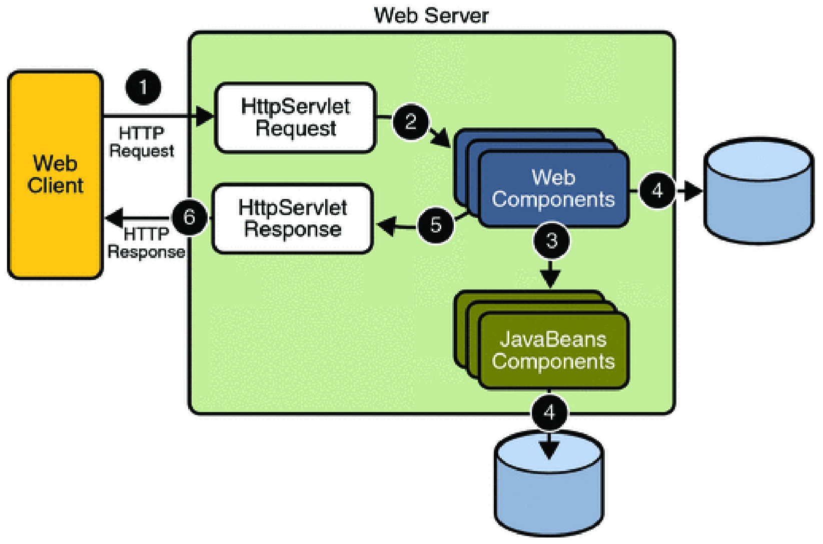
As the name suggests, a servlet is rather like an applet but it runs on the server side. It

- runs inside a JVM (it's a Java object)
- can handle multiple requests (i.e. is provided with easy way to use sessions)
- can communicate with other servlets
- is given (by the servlet container) an OO-wrapped view of the HTTP request/response cycle, e.g., receives a request object, populates a response object.

Easy to make more efficient than CGI – but don't go mad with session use, NB memory implications.

Often combined with JSP...

Basic server-side processing



picture from Java EE tutorial, Ch3

Servlets 2

Concretely a servlet must implement:

- public void init(ServletConfig config)
- public void doGet(HttpServletRequest request, HttpServletResponse response) - process request which was received via the HTTP GET protocol, building response.
- public void doPost(HttpServletRequest request, HttpServletResponse response) - process request which was received via the HTTP POST protocol, building response.
- public void destroy()

A servlet is usually a subclass of javax.servlet.http.HttpServlet – although many frameworks provide more specialised subclasses.

A tiny servlet using resource injection

```
private @Resource String welcomeMessage;

public class HelloWorld extends HttpServlet {

     public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
     throws ServletException, IOException {
         PrintWriter out = response.getWriter();
         out.println(welcomeMessage);
     }
}

<env-entry>
    <env-entry-name>welcomeMessage</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-name>
    <env-entry-value>Hello World from env-entry!</env-entry-value>
</env-entry>
```

Source: `http://java.sun.com/developer/technicalArticles/J2EE/injection/`

JSP

Java Server Pages

file.jsp – contains HTML and Java, compiled into a servlet and then treated like any other.

Can define “tags” – syntactic sugar that save writing the same Java repeatedly.

The JSTL, Java Standard Tag Library, contains many, e.g., for database access.

`http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/`

Using components in servlets/JSPs

Fundamental problem: using a “flat” servlet or JSP to implement a web application page quickly gets unmaintainable.

You need some way to hive off functionality in sensible ways that allow understanding, maintenance and reuse. (NB these desiderata can be in conflict – recall the dependency injection discussion!)

Various solutions, from “roll your own” use of Java classes that you then access using tags, to use of a framework such as Struts.

Let’s look briefly at some options, introducing terminology as we go.

POJOs

Plain Old Java Objects

“We wondered why people were so against using regular objects in their systems and concluded that it was because simple objects lacked a fancy name. So we gave them one, and it’s caught on very nicely.”

Martin Fowler, Rebecca Parsons and Josh MacKenzie

(cf POTS, Plain Old Telephony Service, which acronym long pre-dates Java)

I notice something of a trend for frameworks (e.g. Struts2) to advertise that they work with POJOs, i.e. don’t require developers to write classes that inherit from framework classes.

JavaBeans

A JavaBean is (just) a Java class that obeys certain conventions, allowing it to be used by applications relying on those conventions, e.g., as a component in something more complex. It must:

- have a public no-argument constructor
- have getters and setters following naming conventions (property `name`, methods `getName()`, `void setName(String s)`; Boolean property `deceased`, `isDeceased()`...)
- be serializable (i.e. implement the `Serializable` interface).

E.g. JavaBeans can be used in JSPs...

(Later we’ll meet Enterprise Java Beans, which are different...)

Using a JavaBean in a JSP: also note tag use

```
<% // Use of PersonBean in a JSP. %>
<jsp:useBean id="person" class="PersonBean" scope="page"/>
<jsp:setProperty name="person" property="*"/>

<html>
<body>
Name: <jsp:getProperty name="person" property="name"/><br/>
Deceased? <jsp:getProperty name="person" property="deceased"/><br/>
<br/>
<form name="beanTest" method="POST" action="testPersonBean.jsp">
Enter a name: <input type="text" name="name" size="50"><br/>
Choose an option:
<select name="deceased">
 <option value="false">Alive</option>
 <option value="true">Dead</option>
</select>
<input type="submit" value="Test the Bean">
</form>
</body>
</html>
```

PersonBean.java: note naming conventions, null constructor

```
public class PersonBean implements java.io.Serializable {
    private String name;
    private boolean deceased;
    public PersonBean() {}
    public String getName() {
        return this.name;
    }
    public void setName(final String name) {
        this.name = name;
    }
    public boolean isDeceased() {
        return this.deceased;
    }
    public void setDeceased(final boolean deceased) {
        this.deceased = deceased;
    }
}
```

TestPersonBean.java

```
public class TestPersonBean {
    public static void main(String[] args) {
        PersonBean person = new PersonBean();
        person.setName("Bob");
        person.setDeceased(false);

        // Output: "Bob [alive]"
        System.out.print(person.getName());
        System.out.println(person.isDeceased() ? " [deceased]"
                                               : " [alive]");
    }
}
```

Example from `http://en.wikipedia.org/wiki/JavaBean`

Support for servlets and JSP

As a minimum, you need a "web container" or "servlet container" such as Tomcat. This

- manages servlets' lifecycles
- receives requests from a web server, checks whether there is a servlet registered to handle the request, and passes it on if so
- provides *container-managed security* as specified in the servlet package.

(Actually Tomcat is a web server too, but is usually used with the Apache web server for better performance.)

Full Java EE application servers also do the job, of course – see later.

Deploying web application

A group of related servlets, JSPs, beans is packaged together with a web application deployment descriptor (web.xml) into a special JAR file with extension .war.

This is deployed to the web container.

The deployment descriptor specifies the security required. (Now using JAAS – see later. Key point: security can be flexible enough to e.g. permit or deny a request based on time of day, or information from a database, not just the text of the request.)

Struts

(Now Apache Struts, formerly Jakarta Struts)

JSP and servlets are useful but can lead to spaghetti applications. Struts aims to help systematically.

An MVC framework, allowing what might have been done in a single servlet to be structured following the MVC pattern.

MVC in Struts

Model: some Java class

View: a JSP

Controller: a servlet (actually, a filter, known as the ActionServlet

The View and the Controller are coupled using (usually) an XML file struts.xml

Let's look at the Hello World example from

`http://struts.apache.org/2.x/docs/hello-world-using-struts-2.html`

(I've deleted comments, whitespace and the odd boring bit marked by [...].)

Glue between view and controller: struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.devMode" value="true" />
<package name="basicstruts2" extends="struts-default">
  <action name="index">
    <result>/index.jsp</result>
  </action>
  <!-- If the URL is hello.action...
       If [...] success render the HelloWorld.jsp -->
  <action name="hello" class="[...].HelloWorldAction" method="execute">
    <result name="success">/HelloWorld.jsp</result>
  </action>
</package>
</struts>
```

View, part 1: index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Basic Struts 2 Application - Welcome</title>
</head>
<body>
<h1>Welcome To Struts 2!</h1>
<p><a href="<s:url action='hello'/>">Hello World</a></p>
</body>
</html>
```

View, part 2: HelloWorld.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html [...]>
<html>
<head>
<meta http-equiv="Content-Type" content=[...]>
<title>Hello World!</title>
</head>
<body>
<h2><s:property value="messageStore.message" /></h2>
</body>
</html>
```

Controller: HelloWorldAction.java

```
package org.apache.struts.helloworld.action;
import org.apache.struts.helloworld.model.MessageStore;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport {
        private static final long serialVersionUID = 1L;
        private MessageStore messageStore;
        public String execute() throws Exception {
                messageStore = new MessageStore() ;
                return SUCCESS;
        }
        public MessageStore getMessageStore() {
                return messageStore;
        }
        public void setMessageStore(MessageStore messageStore) {
                this.messageStore = messageStore;
        }
}
```

Model: MessageStore.java

```
package org.apache.struts.helloworld.model;
public class MessageStore {
        private String message;
        public MessageStore() {
                setMessage("Hello Struts User");
        }
        public String getMessage() {
                return message;
        }
        public void setMessage(String message) {
                this.message = message;
        }
}
```

Deployment descriptor: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" [...]>
<display-name>Hello_World_Struts2_Ant</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>


    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>[...].StrutsPrepareAndExecuteFilter</filter-class>
    </filter>

     <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

Putting it together

- Build that lot up into a project (in your IDE! you don't want to do this stuff by hand) and compile it to a .war file.
- Deploy to a servlet container that supports Struts.
- Visit the appropriate URL: get a Hello World hyperlink which, when you click on it, displays a new page saying "Hello Struts User".

How it works, quoting the tutorial 1

Your browser sends to the web server a request for the URL `http://localhost:8080/Hello_World_Struts2_Ant/hello.action`.

1. The container receives from the web server a request for the resource hello.action. According to the settings loaded from the web.xml, the container finds that all requests are being routed to org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter, including the *.action requests. The StrutsPrepareAndExecuteFilter is the entry point into the framework.
2. The framework looks for an action mapping named "hello", and it finds that this mapping corresponds to the class "HelloWorldAction". The framework instantiates the Action and calls the Action's execute method.

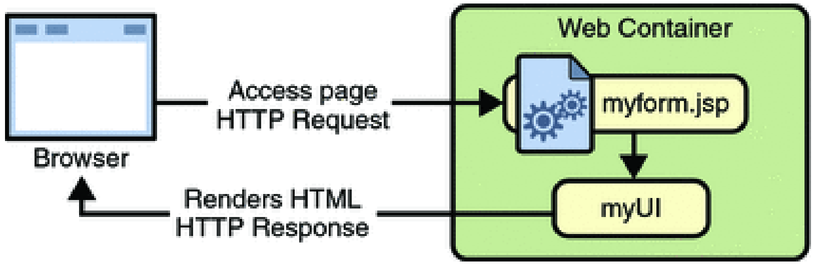
How it works, quoting the tutorial 2

3. The execute method creates the MessageStore object and returns SUCCESS. The framework checks the action mapping to see what page to load if SUCCESS is returned. The framework tells the container to render as the response to the request, the resource HelloWorld.jsp.
4. As the page HelloWorld.jsp is being processed, the <s:property value="messageStore.message" / > tag calls the getter getMessageStore of the HelloWorld Action and then calls the getMessage of the MessageStore object returned by getMessageStore, and the tag merges into the response the value of the message attribute.
5. A pure HTML response is sent back to the browser.

JSF (1)

Java Server Faces: server-side UI.

JSF post-dates servlets and JSP, and simplifies typical tasks done using those technologies. It adds a level of indirection: lets the presentation be defined separately from its representation in HTML.



picture from `http://java.sun.com/javaee/5/docs/tutorial/doc/` Ch10

JSF (2)

JSF defines a library of UI components.

This is conceptually separate from the rendering of these components as UI elements. JSF comes with a render kit to render components in HTML. Tags from this custom tag library are used in the JSP page to say how the UI is to be rendered.

A JSF page is just a JSP that uses JSF tags.

Typically, you write one backing bean for each JSF page. The bean manages the properties referred to from the page.

Navigation is defined separately from the pages, in the application configuration resource file (faces-config.xml).

Struts vs JSF

Struts and JSF are both doing basically the same job: helping build more maintainable fairly simple web applications.

A useful (but old, and JSF-biased) comparison is

`http://websphere.sys-con.com/node/46516`

Headline: JSF gives more support for View development, Struts for Controller and Model development and integration.

ASP: Microsoft's Active Server Pages

Comparable to JSP, but only Microsoft web servers understand them.

Ajax

Asynchronous JavaScript and XML

A way of developing client-side applications using a bunch of technologies... Main characteristic: decouple

- retrieval of data from the server
- the user interface.

APIs and SPIs

The general way that Java technologies are organised is that the JCP – Java Community Process – defines/ratifies a JSR – Java Specification Request – defining a technology.

Often, what is defined is an API – application programmer's interface – defining what the service should offer.

If the service wraps a technology, e.g. a database, that may be implemented in several ways, there is often a SPI – service provider's interface – on the other side. This consists of interfaces that must be implemented (or abstract classes that must be extended). The Adapter pattern may be useful.

JDBC

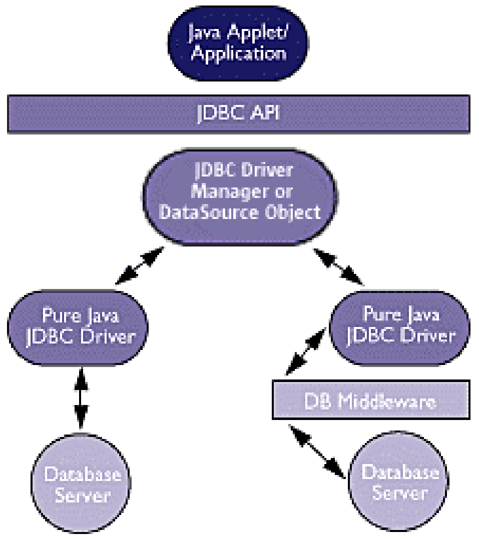
Java DataBase Connectivity

For the applet/application side: provides an API for accessing table-based databases, spreadsheets, flat files in a uniform way using SQL.

Various ways of accessing DBs on the server side:

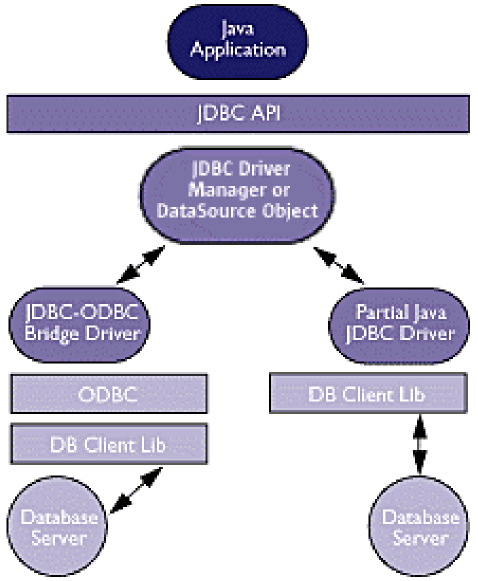
- pure Java, direct to database or via DB middleware
- partial Java, via a DB client library, maybe made available as ODBC (open database connectivity)

Pure Java server side



picture from http://java.sun.com/products/jdbc/overview.html

Partial Java server side



picture from http://java.sun.com/products/jdbc/overview.html

Java EE 5

So far everything has been in Java SE (standard edition) – now we move on to extra capabilities of Java EE (Enterprise Edition) aka J2EE.

A Java EE system has a clean distributed multitier architecture:

- client (thin browser-based client, or thick application client)
- web tier, using a Java EE server
- co-located business tier, using a Java EE server
- EIS tier (often: legacy systems and DB)

Java EE application servers

Java EE specifies many APIs, which application developers can use to simplify their lives.

An application server provides all these APIs.

It manages all the Java EE components, such as servlets and EJBs. Also provides a deployment tool.

Popular examples include:

- JBoss (RedHat, open source)
- WebSphere (IBM; proprietary and community editions)
- GlassFish (Oracle; GPLed)
- etc. etc....

Comparison at `http://en.wikipedia.org/wiki/Comparison_of_application_servers`

Containers

Notional model: developer-written Java EE components are deployed into *containers* which typically do the Dependency Injection required, and manage common requirements such as

- security
- persistence
- transactions

according to deployment specifications.

(“Notional” because e.g. an “applet container” is just “a web browser and Java Plug-in running on the client together.”)

Specifying deployment

Old model: write a separate deployment descriptor in XML to do the specification. Good if a non-developer must alter it (but is that wise?)

New model: use annotations in the Java classes. Easier to comprehend, easier to manage in a tool.

Usually possible to use either, or even a mixture, using a deployment descriptor to override what the annotations specify.

Where are we?

So far we've mostly talked about presentation technology, in the web tier

Now we focus on the business tier.

Lots of competing technologies – usually possible to combine what you want somehow...

Let's have a quick look at the popular Spring/Hibernate combination, before going on to look at EJB and the rest of the official Java EE stable of technologies.

Spring

Has a LOT of stuff in it... gives abstraction layers for transactions, persistence, web application development, JDBC; was influential on, and now implements, JSR330 (at-inject for dependency injection).

Spring MVC is a "lightweight" MVC framework, supposedly better than Struts or EJB...

Often used in one breath with Hibernate, which is complementary:

- Spring focuses on the business logic layer
- Hibernate provides the data access layer.

Hibernate

Object relational mapping framework: maps Java classes to relational database tables, and provides querying languages (HQL and a more object oriented one...)

Metadata either as annotations in the Java classes, or as a separate XML file...

Hibernate conforms to the JPA...

`http://en.wikipedia.org/wiki/Hibernate_(Java)`

JPA

Java Persistence API

Superficially, as for Hibernate... many of the ideas for JPA came from Hibernate.

JPQL, Java Persistence Query Language, an SQL-like language

and criteria queries...

Part of EJB 3.0 - replaces EJB 2.0 CMP (container-managed persistence): entity beans now deprecated.

EJB

Enterprise Java Beans

Framework for the server side of enterprise Java applications.

Original aim: reduce repetitive work involved in persistence, transaction management, security etc.

EJBs are business components as opposed to web components like servlets. Two kinds:

- session bean (ephemeral)
- message-driven bean (can also listen for messages, typically JMS ones)

Problem: difficulty of understanding what has to be done is more of a problem than time taken to write the code, and early versions of EJB didn't really help - hence plethora of "lightweight" alternatives. EJB3 attempts to simplify.

Stateless session EJB3: look, simple POJO!

```
@Stateless
public class CalculatorImpl implements CalculatorRemote, CalculatorLocal {
  public int sum(int add1, int add2) {
    return add1+add2;
  }
  public int multiply(int mul1, int mul2) {
    return mul1*mul2;
  }
}
```

Source:

`http://openejb.apache.org/3.0/simple-stateless-example.html`

EJB3

The interface CalculatorRemote is annotated with

```
@Remote
```

and otherwise all is as you expect.

Local interface gives normal call-by-reference access to the functionality.

Remote interface gives call-by-value and types must be serializable!

Further annotations tell the EJB container how to manage the EJB...

JMS

Java Message Service: a Message Oriented Middleware API, part of Java EE

"allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous."

Two modes:

1. point-to-point (queue: each message goes to one receiver)
2. publish/subscribe

Many providers – every Java EE application server must include one.

Can use JNDI in conjunction.

`http://java.sun.com/products/jms/`

Managing communication and access to resources

We've mentioned dependency injection and containers managing it in passing.

Systematically, what's needed is a way to access a resource – whether that is a database, a property, an object, or whatever.

JNDI is the way this is done in Java EE.

We've already seen its use in fact: in our resource injection example the name for the thing injected is a JNDI name.

JNDI

Java Naming and Directory Interface

Pretty old and simple (earliest version 1997).

Look up Java objects by name or attributes.

Provides a common interface for naming services, e.g. LDAP, NDS, DNS, and NIS(YP).

Used by Java's RMI - Remove Method Invocation, which allows an object running in one JVM to invoke a method on an object running in another JVM

– but not by the more sophisticated JINI (Jini Is Not Initials!), now Apache River, which has its own equivalent.

Transaction management

Two kinds available to EJBs:

- container-managed transaction demarcation, the default: usually each business method is a separate transaction, *implicitly*;
- bean-managed (aka "application managed") transaction demarcation, making *explicit* use of JTA or JDBC transactions.

JTA

Java Transaction API

allows transactions to involve multiple databases even from different vendors

(but not nested transactions)

Outline of use of JTA UserTransaction

```
// In the session beans setSessionContext method,
// store the bean context in an instance variable

this.ctx = sessionContext;

// somewhere else in the beans business logic
UserTransaction utx = ctx.getUserTransaction();

// start a transaction
utx.begin();

// Do work

// Commit it
utx.commit();
```

Source:
`http://en.wikipedia.org/wiki/Java_Transaction_API`

Security

We've overlooked security so far... but could have spent the entire course on it, easily.

Java SE already provides many facilities, see

`http://download.oracle.com/javase/6/docs/technotes/guides/security/`

and

`http://download.oracle.com/javase/tutorial/security/index.html`

Security in Java SE

- A Security Manager controls the access that applets (or some applications) have to resources, e.g. files; it can be guided by a security policy file.
- JCA, Java Cryptography Architecture: digital signatures; public key infrastructure; signing code; en/decryption; secure random number generation, etc.
- JAAS: Java Authentication and Authorization Service, on a per-user or per-group basis.
- GSS-API: Java Generic Security Services for secure message exchange (token-based) : this plus JAAS permits using Kerberos in Java apps.
- JSSE: Java Secure Sockets Extension

Security in Java EE

Java EE security is managed by the containers of components, typically, EJBs. Conceptually split:

- application-layer security
- transport-layer security, e.g. use of SSL
- message-layer security, e.g. use of Web Services Security with SOAP messages

All managed by containers of Java EE components, specified in deployment descriptors/annotations (declarative security), or explicitly by code (programmatic security).

Seems to get specific to the choice of application server quite soon.

JCA

Java EE Connector Architecture

can be seen as a generalisation of JDBC: it's a way of connecting Java EE applications to legacy systems in general, not just legacy databases.

More specifically it lets you connect a Java EE Application Server to an EIS (enterprise information server) using generic tools for managing connection tools, security, etc.

Web services

- your functionality is made available over the Web
- implemented in Java or whatever you like;
- invoked using a service request which is an XML file sent over HTTP, e.g. a SOAP (Simple Object Access Protocol) message
- discovered using WSDL (Web Services Description Language)

JAX-WS simplifies writing web services, e.g. by wrapping access to SOAP.

And now there's just time for the test

:-)