Introduction to J2EE

- What is J2EE?
- J2EE Development Process
- Overview of J2EE Technologies

What is J2EE?

- Java2 Enterprise Edition is a set of specifications for server-side Java programming
- Vendors implement J2EE by creating **containers** within an application server
- From the Sun Web site:

> The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications. The J2EE platform simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

Benefits of J2EE

- Portability across platforms
- Portability across vendors
- Containers provide high-level services
- Many implementations from which to choose
- Wide array of tools for development, testing and deployment

J2EE Vendors

A partial list of J2EE vendors/products:

- IBM WebSphere
- BEA WebLogic
- JBoss
- Oracle J2EE Containers
- Apache Tomcat (Web container only)
- Caucho Resin

The key is that it should be fairly easy to port applications from one vendor to another since J2EE defines standard specifications.

J2EE Development Tools

A partial list of J2EE development tools:

- IBM WebSphere Studio Application Developer (WSAD)
- BEA WebLogic Workshop
- Eclipse
- Netbeans
- JBuilder

The goal of these tools is to make the J2EE development and deployment process easier.

J2EE Technologies

A partial list of the Application Programming Interfaces (APIs) defined by J2EE (current version is 1.4) and their current version numbers:

- Servlets (2.4)
- JavaServer Pages (2.0)
- Enterprise JavaBeans (2.1)
- Java Naming and Directory Interface (1.2)
- JDBC (3.0)
- Java Messaging Service (1.1)
- Java Web Services (n/a)

Deployment Descriptors

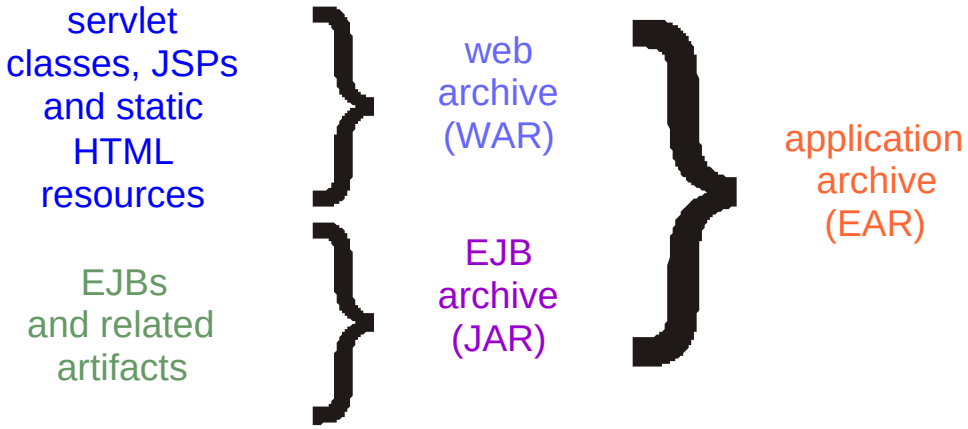
- Most J2EE technologies use XML-based **deployment descriptors** for configuration
- Many container vendors provide high-level tools to make it easier to work with these deployment descriptors

```
1    <?xml version="1.0"?>
2    <web-app>
3      <servlet>
4        <servlet-name>HelloCount</servlet-name>
5        <servlet-class>HelloCount</servlet-class>
6      </servlet>
7      <servlet-mapping>
8        <servlet-name>HelloCount</servlet-name>
9        <url-pattern>HelloCount</url-pattern>
10     </servlet-mapping>
11     <welcome-file-list>
12       <welcome-file>InputCount.html</welcome-file>
13     </welcome-file-list>
14   </web-app>
```


This page shows an example deployment descriptor for a Web application. At this point, it's not important that you understand the syntax -- just that you know that deployment descriptors are XML-based and configure the different parts of a J2EE enterprise application.

J2EE Development and Packaging

- Since J2EE is a Java technology, programmers write J2EE Java code and compile it in the normal fashion
- J2EE provides various ways to package complete applications to make them easier to deploy



These archives are internally similar to ZIP files.

Note that each type of archive has its own deployment descriptor. The deployment descriptor shown on the last page was for a WAR file.

J2EE Roles

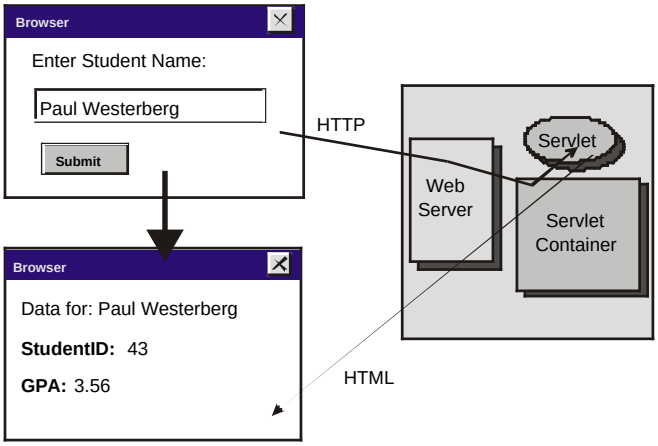
- Since J2EE is designed for large, enterprise applications, it's helpful to split the work into roles
- For smaller applications, a single person may play multiple roles

Role	Description
J2EE Product Provider	Provides the J2EE container(s)
Tool Provider	Provides J2EE development tools
Application Component Developer	Writes servlets, JSPs, EJBs and related Java code
Application Assembler	Packages and configures the application into archives
Deployer	Deploys the EAR into container
System Administrator	Configures, tunes and manages the container

These roles are defined in the J2EE specification and are just a guideline on how to organize the end-to-end application development and deployment process. It's common for individuals to play multiple roles.

Introduction to Servlets

- Servlets are a Java replacement for traditional CGI programs and let you add programmability to your Web sites
- Servlets perform well and are portable across container vendors and platforms
- A common use of servlets is to respond to HTML forms



When servlets were first introduced, their primary role was to generate HTML. This has changed somewhat with the introduction of JSPs -- now, servlets are generally more specialized -- they respond to requests, execute presentation-tier logic and then transfer control to the JSP that will generate the HTML.

Introduction to JSPs

- Like servlets, JSPs let you add programmability to Web sites, but JSPs are more focused on presentation rather than Java coding
- Upon first access, the container converts JSPs into servlets, so they have the same performance and portability characteristics

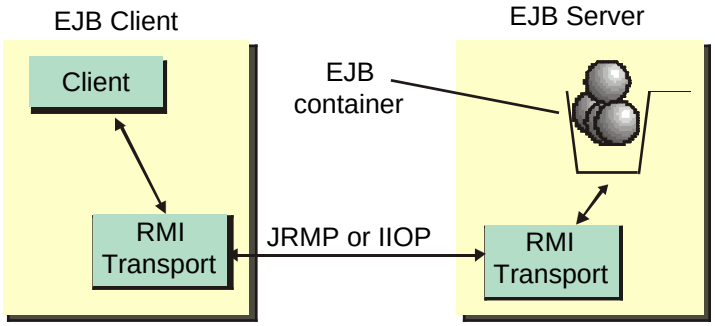
```
1    <html>
2      <body>
3    <%
4      for (int i=0; i<5; i++ )
5        out.println ( "Hello<br>" );
6    %>
7      </body>
8    </html>
```


Though you could create an entire Web site using only JSPs, it turns out that putting a lot of code in a JSP makes them difficult to maintain. So as mentioned on the last page, a good strategy is to combine servlets and JSPs, putting the bulk of the presentation-tier code in the servlet and letting the JSP handle HTML generation.

Introduction to EJBs

- Enterprise Java Beans are distributed components that run within the bounds of an EJB container
- The container provides services such as pooling, transactions, security and persistence
- From the EJB specification:

  Enterprise JavaBeans is an architecture for component-based computing. Enterprise beans are components of transaction-oriented enterprise applications.

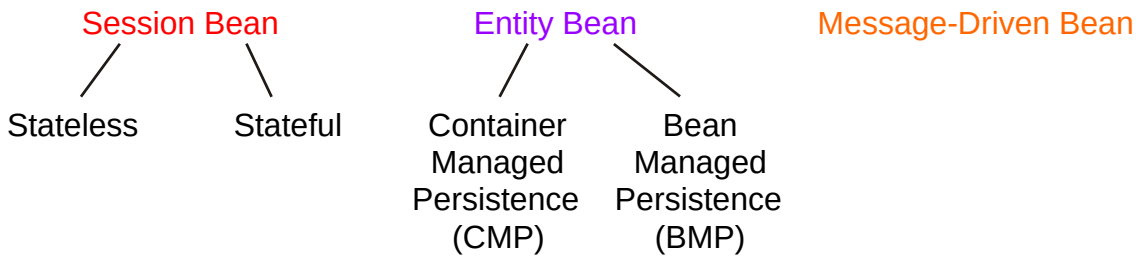


One key to understanding EJBs is that they can live remotely from the program that calls them. It's quite common in J2EE applications to architect a servlet as the EJB "client".

JRMP and IIOP are protocols for distributed objects -- EJBs can use either one.

Types of EJBs

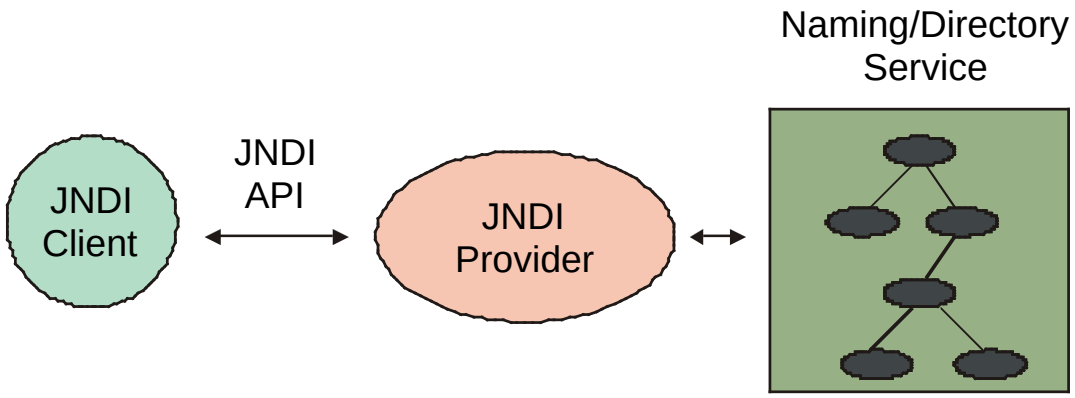
- The EJB specification allows for session, entity and message-driven beans
- It's a good practice to use session or message-driven beans for high-level business processes and have them access entity beans to implement the process -- this is the Session Facade design pattern



Session beans represent business processes. Entity beans represent business data and the operations that work on a specific chunk of data. Message-driven beans are similar to session beans, but they are invoked asynchronously (i.e. the caller doesn't have to wait for the operation to complete).

Introduction to JNDI

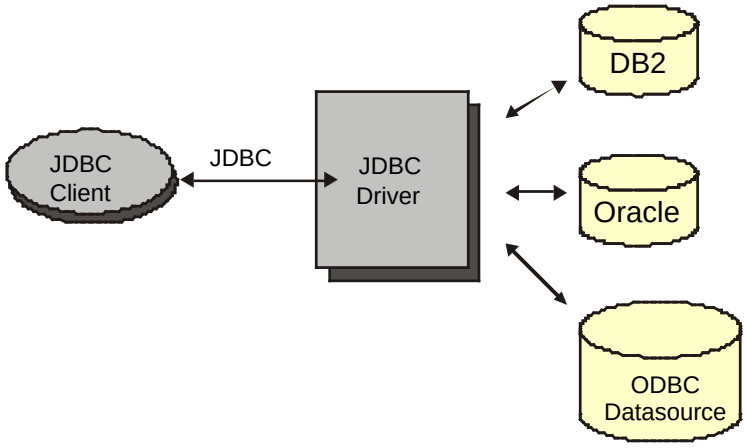
- The Java Naming and Directory Interface provides a standard way to access naming services such as LDAP
- All J2EE compliant containers must provide a JNDI implementation since other J2EE technologies such as EJBs depend on JNDI



Most enterprise-class applications need the ability to look up resources in some sort of directory. Prior to JNDI, Java programmers needed to code to a particular naming/directory service and then if they changed services, all of the code would need to be rewritten. JNDI avoids that by providing a standard programming interface to any naming/directory service that has a JNDI provider written for it.

Introduction to JDBC

- The JDBC API provides a standard way to access relational data so that your Java programs can remain independent of the actual database product
- To use JDBC, you must obtain a JDBC driver for your particular database



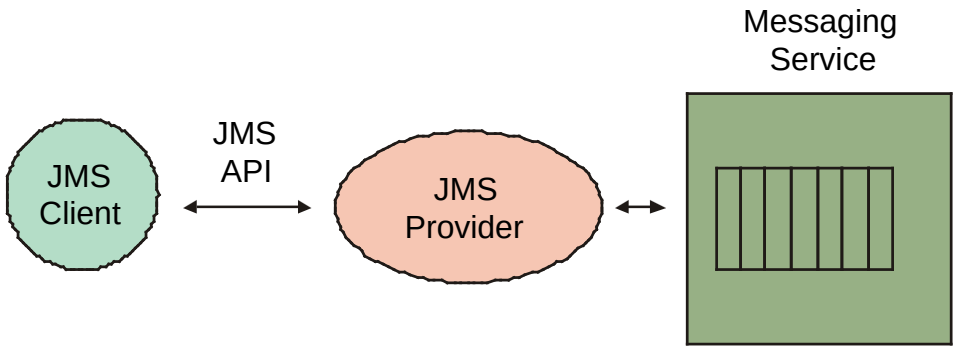
3 - 15

JDBC is one of the most well known and widely used J2EE APIs (in fact, JDBC actually predates J2EE!).

JDBC provides a standard programming interface for relational databases so that your Java code is independent of the actual database product. That way, if you need to switch database vendors, your code should still work.

Introduction to JMS

- The Java Messaging Service provides a standard way to access message-oriented middleware (MOM) products such as IBM's WebSphere MQ in a product-independent fashion
- To use JMS, you must obtain a JMS provider for your particular MOM product



Messaging systems provide a way to create loosely coupled applications that are resistant to failure. There are several messaging services available, each with their own proprietary API. The JMS API provides a standard programming interface so that Java programs are independent of the actual messaging service. That way, if you need to change services, your code should still work.

Introduction to JavaMail

- The JavaMail API provides a standard way for Java applications to send and receive Internet email without having to program the SMTP protocol directly

```
1    Properties p = System.getProperties();
2    p.put("mail.host", "localhost" );
3
4    MimeMessage message =
5          new MimeMessage ( Session.getInstance(p, null ) );
6    message.setFrom
7           ( new InternetAddress ( "me@mycompany.com" ) );
8    message.setRecipients(Message.RecipientType.TO,
9          InternetAddress.parse ( "you@yourcompany.com" ) );
10
11   message.setSubject ( "Great Subject" );
12   message.setText( "The best message ever!" )
13   Transport.send(message);
```


JavaMail is an API for sending and receiving email in Java programs. The code fragment on this page shows sending a message to "you@yourcompany.com" from "me@mycompany.com" with the subject "Great Subject" and message body containing the text "The best message ever".

While it's not important at this point that you understand this code completely, it should give you a feel for how the JavaMail API works.

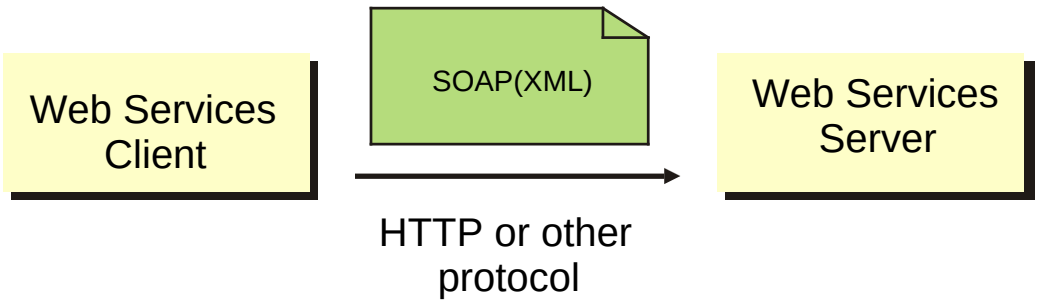
Introduction to JTA

- The Java Transaction API provides a standard interface to transaction managers such as those included with database products like IBM's DB2 or Oracle
- Most J2EE programmers do not use JTA directly, but instead use **declarative security** and let the container handle transactions

Declarative security means that you configure security constraints in a deployment descriptor rather than coding to the JTA directly.

Introduction to Web Services

- Web services use the SOAP protocol to allow for remote procedure calls and document exchange
- J2EE 1.4 defines the JAX-RPC, JAX-M and JAX-R APIs so that J2EE developers can use Web services in a vendor-independent fashion



Web services provide a platform and language-independent way to invoke remote services or pass arbitrary business documents. Large companies such as Microsoft and IBM have adopted Web services as an integration technology.

Many Java vendors have created Web services toolkits -- J2EE 1.4 provides standard APIs for programming Web services in Java so that your code is portable across toolkit vendors.

Chapter Summary

In this chapter, you learned:

- What J2EE is all about
- Fundamentals of several J2EE technologies