Advanced Java programming (J2EE)

Enterprise Java Beans (EJB)

Chris Wong
cwh@it.uts.edu.au
[based on prior course notes & the Sun J2EE tutorial]

Enterprise Java Beans

- Today we will briefly cover
  - Introduction to EJB
  - EJB basics
  - EJB Architecture
  - Session Beans
  - EJB Clients
  - EJB Development Process

Introduction to EJB

So far we have developed multi-tier applications:

- Web-based Applications (Servlets/JSPs/JDBC)
  - "thin" client – using browser
  - Model-View-Controller architecture
  - simple business logic
  - Some services provided by web container
- Distributed Applications (RMI/JDBC)
  - "fat" client – using Java application
  - Simple client/server architecture
  - You have to write all other services

Why EJB?

- HOWEVER... what about:
  - Reusability of components?
  - Security?
  - Resource management?
  - Scalability – Load Balancing, Clustering?
  - Fault tolerance? Error handling?
  - Deployment in other environments?
  - Persistence? Independence from specific databases?

- With web applications & RMI, we have to program this.
- EJB containers provide the above functions.
EJB Introduction

• EJB is the J2EE server-side component model

• The component software model is based on the idea of creating reusable components that “plug in” to “containers”

• It addresses some important software development goals:
  - reuse
  - high level development focus
  - development automation via tools
  - simplified deployment

• JavaBeans, EJB and ActiveX/COM are examples of component models

EJB Introduction

• Component models come in two basic flavours
  - client-side; and
  - enterprise

• Client-side component models such as JavaBeans are specialised to handle presentation and user-interface issues

• Enterprise component models such as Enterprise Java Beans are specialised to provide a framework for developing, deploying and executing distributed, transaction-aware middleware components

EJB Introduction

• Component developers write component “building blocks” that implement business logic

• Application developers hook up these pre-built components into finished applications (which may be components themselves)

• This building block approach facilitates off-the-shelf reuse of code packaged as components

Enterprise Java Beans

• Today we will briefly cover
  - Introduction to EJB
  - EJB Basics
  - EJB Architecture
  - Session Beans
  - EJB Clients
  - EJB Development Process

EJB – Types of Beans

• There are 3 types of Enterprise Java Bean:
  1. Session beans
     • stateless
     • stateful
  2. Message Driven beans
  3. Entity beans
     • This has been replaced with JPA

EJB Basics

• EJB interfaces and classes are defined in the javax.ejb package

• EJBs are configured and deployed in a similar manner to web applications
  - Packaged into a jar file
  - Optional XML deployment descriptor(s)
    - META-INF/ejb-jar.xml
    - weblogic-ejb-jar.xml
**EJB Standards Evolution**

The EJB specification is an evolving API

- **EJB 1.0** was originally defined as a Java extension in 1998, including definitions for Session beans and (optional) Entity beans

- **EJB 1.1** is the J2EE 1.2 was defined in December 1999 as part of the first J2EE platform specification. EJB 1.1 included:
  - Mandatory support for entity beans
  - Enhancements to the deployment descriptor, including support for an XML format
  - Significant tightening of the specification to improve EJB-based application portability

- **EJB 2.0** is the J2EE 1.3 standard.
  - JMS (Java Message Service) integration with Message Driven beans
  - Improved support for container-managed persistence (CMP)
  - Support for RMI/IIOP protocol for network interoperability

- **EJB 3.0** is the Java EE 5 standard.
  - Significantly simplified EJB development. Now use Java 5 annotations
  - Replaced Entity EJB with JPA

---

**EJB Containers**

- Containers provide services to deployed components so that component developers can concentrate on writing business logic instead of software infrastructure

- An EJB container is an environment in which Enterprise Java Beans execute. Its primary role is to serve as a buffer between EJBs and the outside world

- The container simplifies component development by providing services:
  - Transactions
  - Scalability (management of multiple instances)
  - Persistence
  - Security
  - Synchronisation
  - Distributed object invocation
  - Monitoring and management of EJB instances

---

**EJB and application servers**

- We are using a "web application server"

- Note that it has two different types of container:
  - Web container (for servlets and JSPs)
  - Business-logic container (for EJBs)

- They are logically two separate "tiers"
  - In some cases they are implemented by two separate products, e.g. Tomcat + JBoss
  - Each type of container provides different services

---

**EJB and application servers**

- EJB and object containers
  - J2EE Server
  - Servlet
  - JSP Page
  - Web Container
  - Application Container
  - Enterprise Bean
  - EJB Container
  - Database

---

**EJB Containers - services**

- EJB Containers must provide the following services
  - Transaction management
  - Security services
  - Multiple instance support
Container Services – Transactions

• Declarative transaction management:

  – An application's use of transactions can be declared in the XML deployment descriptor, rather than having a programmer write code to manage transactions

  – Less code to write
  – Less chance of programmer-generated errors
  – Allows container to manage resources efficiently

Container Services – Security

• Declarative security management:

  – An application's use of authentication (login) and confidentiality (encryption) can be declared in the XML deployment descriptor, rather than having a programmer write code to manage security

  – Less code to write
  – Less chance of programmer-generated errors
  – Allows system administrator to manage user access externally to the application code

Container Services – Instances

• Management of multiple instances:

  – EJBs are written as if they are single threaded classes being accessed by only one client

  – This is true for some beans (stateful session beans in particular), however other beans may be used concurrently by multiple clients

  – The EJB container must ensure that each client's requests are serviced in a timely manner

Container services - Instances

• In order to accomplish the goal of timely responses to client requests, the server may perform several tasks:

  – Instance passivation – the temporary swapping of an EJB out to storage. If a container needs resources, it may choose to temporarily passivate a bean

  – Instance pooling – the creation of a pool of bean instances, so that beans can be shared between multiple clients, rather than a new bean being created for every client request (not possible in the case of stateful session beans)

  – Database connection pooling – the creation of a pool of database connections, so that connections can be shared between multiple beans, rather than a new connection being created every time the database is accessed

Tasks containers perform (2)

• Pre-cached instances – the caching of EJB state information to accelerate the initial creation of EJBs

• Optimised method invocations – the short-circuiting of method calls between two beans in the same container in order to avoid the overhead of a full-blown remote method call (this is done by the container, not by the EJBs themselves - not strictly true as local interfaces in EJB 2.0 achieve the same)

• Instance passivation is the only optimisation required by the EJB specification, none of the others are explicitly required

Enterprise Java Beans

• Today we will briefly cover

  – Introduction to EJB
  – EJB Basics
  – EJB Architecture
  – Session Beans
  – EJB Clients
  – EJB Development Process
EJB Architecture

The primary application-specific elements of the EJB architecture shown in the diagram are:

**EJB Client**
- Optionally use JNDI to look up home interfaces, and use home and remote interfaces to utilise all EJB-based functionality.
- Dependency injection adds Home interface transparently
- EJB clients can also be external to the application server

**EJB Home Interfaces (and Stubs):**
- Provide lifecycle operations to create, remove and find handles to EJB remote object interfaces.
- Hidden from the developer by the container.
- Underlying stubs marshal home interface requests and unmarshal home interface responses for the client

**EJB Remote Interfaces (and Stubs):**
- EJB remote interfaces provide the business-specific methods
- Underlying stubs marshal remote interface requests and unmarshal remote interface responses for the client

**EJB Implementations:**
- Implementations are the application components implemented by developers to provide any application specific business logic.
- May also include lifecycle methods eg: Remove()

• The client side contains:
  - The EJB interfaces needed to invoke business methods on an EJB
  - Internally generated proxy/stubs for maintaining handles to the server-side objects

• The server side contains:
  - The instances of the actual EJB components
  - The EJB container maps calls from clients to EJB instances (after the appropriate service management infrastructure code has been executed)

• RMI remote interface semantics are implied by the interfaces to EJBs

EJB Architecture pieces (1)

EJB Architecture pieces (2)

EJB Architecture pieces (3)

Entire Java Beans

• Today we will briefly cover
  - Introduction to EJB
  - EJB Basics
  - EJB Architecture
  - Session Beans
  - EJB Clients
  - EJB Development Process

• Next lesson we will cover
  - Entity Beans
  - Message Driven Beans
  - EJB issues
EJB 3.0

- Part of Java EE 5
- Significantly simplifies EJB development
- Use Java 5 annotations.
  - Assumes common defaults for most infrastructure classes/methods
  - You write your bean as a POJO (Plain Old Java Object).
- You put one of the following annotations before the class declaration
  - @Stateless
    - Stateless session bean
  - @Stateful
    - Stateful session bean
  - @MessageDriven
    - Message bean
  - @Entity
    - Entity bean (See JPA)

Entity Beans

Entity beans are:
- EJBs that encapsulate some data contained by the system
- Implemented as JPA Entities
- "Nouns" of a particular problem domain that they are implemented to solve
- Provide CRUD (created, searched, update, delete) functionality

Session Beans

Session beans are:
- EJBs that perform actions on the enterprise system.
- Model a connection (session) with a single client
- "Controllers" for the system that are exposed to the client for manipulation
- "verbs" of the particular problem domain that they are implemented to solve
- Well suited to encapsulate coarse-grained "front line" service entry points into a system

Session & Entity Beans

Session beans can be grouped into 2 distinct classifications:
- Stateless session beans:
  - No conversational state kept between client and bean
  - Typically pooled – so 1 bean may actually service 100's of clients
  - Easy to cluster, no affinity between client and physical server
- Stateful session beans:
  - Keeps conversational state between client and bean
  - 1:1 between client and bean. So if 100 clients, need 100 beans.
  - Need to manage lifecycle – performance improved by container "paging out" (Passivating) and reactivating beans

Note: Session beans persist only for the life of the connection with the client – if the server crashes, the session bean dies
**Session Beans**

- You need to create two files:
  - Remote/Local interface (declares your business logic)
  - implementation class (method implementations)

- Let's start with a stateless session bean
  - the easiest kind

**EJB 3.0 stateless session bean**

- Just plain old Java Objects! Eg: BankTeller:
  ```java
  @remote
class BankTeller {
    public string sayHello(String name);
  }
  
  @stateless(name="BankTeller")
class BankTellerBean implements BankTeller {
    public string sayHello(String name) {
      return "Hello, " + name;
    }
  }
  ```

**Stateless Bean – code notes**

- @Remote indicates that the bean can be accessed via RMI-IIOP
- @Local indicates that this is just run within the same JVM
- @Stateless can have optional attributes to override defaults
  - eg: name == name of bean
  - beanInterface == name of bean class
  - mappedName == JNDI name of bean

**Simple lifecycle**

- When bean created, you can OPTIONALLY have callback method defined for initialisation
  ```java
  @PostConstruct
  public void init() {
    // initialise classes?
  }
  ```

- When bean created, you can OPTIONALLY have callback method defined for initialisation
  ```java
  @PreDestroy
  public void close() {
    // close databases?
  }
  ```

**Stateful Session Beans**

- Stateful session bean is basically identical to a stateless session bean

- The difference? For stateful beans:
  - you should define some `state`, i.e. instance variables
  - inside your business methods, you can manipulate the state (read and change the variable values)

- 1 instance per client

- Need to deal with more complex lifecycle
Stateful beans – Lifecycle

- Keep state – eg: via instance variable
- Lifecycle tasks still optional. Extra callback methods
  - @Init
    - Similar to @PostConstruct, but can have more than 1 init method.
  - @PrePassivate
    - Use this to store/save conversational information eg: to database
  - @PostActivate
    - Use this to retrieve conversational information eg: read from database
  - @Remove
    - Use this to delete the bean – eg: drop conversational state information

Stateful Beans – how not to use

- Stateful session beans do NOT represent the data in your application
  - data is represented by Entity Beans
  - stateful session beans are transient – they come and go. Data is meant to be persistent
  - if a server crashes unexpectedly, a stateful session bean will be destroyed with the crash (same for stateless session beans). Data is meant to be persistent.

Stateful Scenario

- Scenario: When you walk up to a bank teller, you tell him/her your account number. You can then perform any number of deposits and withdrawals with that teller, without having to tell him/her the account number again each time – he/she remembers it

  - When you create() the BankTeller EJB, you supply: the account number
  - When you call deposit() and withdraw(), you pass only one parameter: the amount
  - When you call deposit() or withdraw(), it uses ("remembers") the account number that you specified at creation time

Session Beans – modelling

- When do you use stateless beans vs. stateful beans?
  - In most cases, it depends on how you choose to model your application
  - Consider the bank teller examples on the next two slides
    - one is stateful, one is stateless
Stateless Scenario

EJB container

BankTeller

- deposit()
- withdraw()

Client (e.g. server)

create()

deposit(1234,$100)

Teller Example 2 – stateless

- Scenario: When you walk up to a bank teller, you don't give him/her any information. You can perform any number of deposits and withdrawals, but each time, you must specify your account number. He/she does NOT remember your account number from one transaction to the next.

  - When you create() the BankTeller EJB, you supply: nothing
  - When you call deposit() and withdraw(), you pass two parameters every time: account number and amount

Stateless vs. stateful implications

- With the stateless teller, each time you perform a transaction, you can go to any teller
  - The teller you go to doesn't "remember" anything about you, so you can go to a different one each time
  - EJB containers create a pool of stateless session beans, and every time a client calls a method, that method is directed to any bean in the pool

- With the stateful teller, you must go back to the same teller each time...
  - ... because that teller knows your account number. If you go to a different teller, they won't know what your account number is
  - EJB containers assign one stateful session bean to every client
  - There is a one-to-one correspondence between clients and stateful session beans – no pooling!

Enterprise Java Beans

- Today we will briefly cover
  - Introduction to EJB
  - EJB Basics
  - EJB Architecture
  - Session Beans
  - → EJB Clients
  - EJB Development Process
- Next lesson we will cover
  - Entity Beans
  - Message Driven Beans
  - EJB issues

EJB Clients

- In the EJB architecture, the client undertakes the following tasks:
  1. Finding the bean
  2. Calling the bean's methods (call Remote interface methods)
  3. Getting rid of the bean

- A client can be local (in same JVM as bean)
  - Eg: another EJB, Java class or web application component running in the same application server

- or remote
  - Eg: a standalone Java application or a component running in another application server

1. Finding the bean

2 techniques:

1. Use Dependency Injection – this works only with Managed beans eg: Servlets, EJB.
   Container looks for the bean class
   - Eg: servlets, ejb's, etc
   @EJB BankTeller;

   CAUTION: Only works for servlets version 2.5!!
   Does NOT work for JSP. See next page
1. Finding the bean (2)

For non-managed clients
- eg: Java Application, Java Beans or JSP's

2. Use JNDI - looks for the advertised name of the bean's interface.
   BankTeller teller = (BankTeller)
   InitialContext().lookup("BankTeller");

   - JNDI name is vendor specific.
   - Use the name & mappedName attribute on the EJB to define this
   - Can also use XML deployment descriptor

2. Calling the bean's methods

- Now the client has a remote interface, it can invoke the methods the bean has made public
  - i.e. the methods defined in the Remote interface

   out.println( teller.sayHello( "chris"));

3. Getting rid of the bean

- Stateless session beans are automatically removed by the container at its discretion
  - no problem – they don’t hold any state info

- Stateful session beans should be removed explicitly by the client by calling the method annotated with @remove
  - Container won't remove these automatically because they contain valuable state information – your application needs to decide when it is finished with the state info

Enterprise Java Beans

- Today we will briefly cover
  - Introduction to EJB
  - EJB Basics
  - EJB Architecture
  - Session Beans
  - EJB Clients
  - \[ EJB Development Process \]

EJB Development Process

Checkpoint!

- Until now we have considered:
  - what EJBs are, what the different kinds are
  - what containers do and why they're a "good thing"
  - what session beans look like and how they're used

- Now we change focus and look at the EJB development process
  - same process for all kinds of EJBs

EJB Development Steps

- 2 streams
  - server side and client side development

- Server-side development of EJBs is simpler than RMI
  - communications, state management, resource allocation and thread management infrastructure coding provided by the container

- Similar for EJB Clients
EJB Development Steps (1)

• 9 steps involved in server-side EJB development:

1. Create the remote interface
   - Defines business methods

2. Create the EJB implementation class
   - Implements the bean

3. Compile the java code

EJB Development Steps (2)

4. Write the Optional EJB deployment descriptors
   - Create an ejb-jar.xml file and maybe others

5. Package the EJB into a JAR file
   - EJBs get packaged in JAR files

6. Compile the EJB stubs
   - Use an EJB compiler tool

EJB Development Steps (3)

OPTIONAL STEPS:

7. Configure the application deployment descriptor (optional)
   - Deployment descriptor for the whole application, not the EJB

8. Package the EJB JAR into an EAR file (optional)
   - "Applications" are packaged in EAR files

9. Deploy the EJB JAR or J2EE application EAR
   - Deploy to the application server

Client Development Steps

• 3 steps for writing EJB clients:

1. Establish the proper EJB client libraries – make available the correct versions of approriate libraries for the client – JNDI, EJB, RMI/IIOP and possibly JMS and JDBC if required

2. Implement client code – create your client application using the EJB client libraries and your EJB client interfaces

3. Compile / package / deploy the client – as appropriate for the client type

1. Create Remote interface

```java
package bank;
import javax.ejb.*;
@Remote
public interface BankTeller {
    public string sayHello(String name);
}
```

• (Same as previous example)
3. Compile Java code

- `javac -d . *.java`

4. Write EJB deployment desc.

- **OPTIONAL:** Standard deployment descriptor (META-INF/ejb-jar.xml)
  - EJB name
  - Java class names for remote interface, home interface and bean implementation class
  - Whether bean is session or entity
  - For session beans, whether bean is stateful or stateless

- Container-specific deployment descriptor (META-INF/weblogic-ejb-jar.xml)
  - For WebLogic stateless session beans, size of pool
  - JNDI name (name by which the bean is advertised to clients)

4a. ejb-jar.xml

```xml
<?xml version="1.0"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
version="3.0">
<enterprise-beans>
<session>
<ejb-name>BankTellerEJB</ejb-name>
<home>bank.BankTellerHome</home>
<remote>bank.BankTeller</remote>
<ejb-class>bank.BankTellerBean</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>
</session>
</enterprise-beans>
<assembly-descriptor>
<container-transaction>
<method>
<ejb-name>BankTellerEJB</ejb-name>
<method-intf>Remote</method-intf>
<method-name>*</method-name>
</method>
<trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

4b. weblogic-ejb-jar.xml

```xml
<?xml version="1.0"?>
<weblogic-ejb-jar>
<weblogic-enterprise-bean>
<ejb-name>BankTellerEJB</ejb-name>
<stateless-session-descriptor>
<pool>
<max-beans-in-free-pool>100</max-beans-in-free-pool>
</pool>
</stateless-session-descriptor>
</weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

5. Package EJB as a JAR file

- Same concept as for WAR files
  - but EJBs are packaged in JAR files
- EJB deployment descriptors stored in a subdirectory called "META-INF"
- Directory structure in JAR must mirror Java package structure (as per normal Java rules)
  ```
  jar cf ../MyEJB.jar *
  ```

6. Compile EJB stubs

- Run appc (weblogic EJB compiler) on the EJB JAR file
  - this generates and compiles stubs and skeletons
  ```
  java weblogic.appc MyEJB.jar
  ```
- Different development tools may have different ways of compiling the EJB code and generating stubs/skeletons
Deploy

- You can now deploy

- if your client is not in same JAR file, then you need to have a copy of the bean interfaces in the classpath
  - eg: WAR file can contain the BankTeller.class in the /WEB-INF/classes directory

- You can also put multiple bean JAR files and WAR files together to make a "Java EE Application" (EAR) archive

7. Configure app deployment desc.

- We'll come back to this next session (see next slide for a brief outline)

- Suffice to say that this is an optional step, and for now, we're going to leave it out!

8. Package app as an EAR file

- In J2EE terms, an "application" consists of:
  - one or more WAR files
    - containing servlets and JSPs
  - one or more JAR files
    - containing EJBs

- A collection of WARs and JARs that comprise a single application may be packaged in an EAR file
  - EAR = Enterprise Application Archive

- We'll come back to this next session

10. Deploy

- Deploy your JAR file into the application server

- In WebLogic, either:
  - copy the JAR file into your "autodeploy" subdirectory
  - run the command line tool to deploy the file
  - upload the file via the management console

- Exactly the same process as deploying WAR files

- If all goes well, in the management console you should see:
  - a new EJB appear under the "Deployments > EJB" menu
  - a new entry appear in the JNDI tree

EJB Development

- There are quite a few steps, but each is simple

- You will probably find it useful to create a short shell script / batch file to run the various commands (I use "ant" to do this)
  - what is "ant"? - a build tool similar to makefiles on Unix

- The most common error is ...
  - ... typos in the XML deployment descriptor
  - Running appc will pick up many errors
  - Use an XML editor to reduce syntax errors

Summary – EJB Part 1

- Reflect on how little code was required
  - only 2 Java files, an Interface and Implementation
  - plus 2 Optional deployment descriptors

- Hard part about EJB is understanding:
  - the J2EE architecture and how to model the real world
  - how EJBs are invoked / loaded / etc.
  - what the container does
Questions?

Clustering

- EJB is specifically designed in mind for clustering
- Scalability
- Availability

- Load Balancing
- Failover/Migration

Definition: Clustering

- A cluster is a group of Oracle WebLogic Server instances that work in coordination.
- Clustering provides:
  - High availability
  - Load balancing
  - Scalability

Basic Cluster Architecture

- A basic cluster architecture combines static HTTP, presentation logic, business logic, and objects into one cluster.

Multitier Cluster Architecture

- The Web tier and the business logic with services can be separated into two clusters.

Basic Cluster Proxy Architecture

- This is similar to the basic cluster architecture, except that static content is hosted on nonclustered HTTP servers.
Multitier Cluster Proxy Architecture

- This is similar to the multitier cluster architecture, except that static content is hosted on nonclustered HTTP servers.