Enterprise Java Beans (EJB)

Part 2 – Entity & Message Beans

Chris Wong
chw@it.uts.edu.au
(based on prior class notes & Sun J2EE tutorial)

Enterprise Java Beans

Session Façade

- Interlude-

• Earlier we discussed why we use EJB's.
• EJB containers provide much of the 'plumbing' that traditionally programmers had to develop and consider.
• We still need good programming practices to use EJB's well.
• Use Design Patterns
to re-use well known programming techniques.
• Good references:
  1. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software." Addison-Wesley 1995

The Session Façade pattern

• This is the most common design pattern used with EJB's.
• You partition business logic to minimise dependencies between client and server.
• This optimises network traffic and executes as one transaction.
• Idea is to wrap an entity EJB with a session EJB.
• Only one interface between client and server.
• Completely hides internal implementation from EJB.
• Completely hides internal implementation from client.
• Only one interface between client and server.

Traditional client/server

- Interlude-

- This lesson will cover:
  - EJB issues
  - Message-driven Beans
  - Entity Beans
  - EJB development process
  - EJB Clients
  - EJB containers
  - EJB deployment
  - Introduction to EJB
  - Last lesson we looked at:

Part 2 – Entity & Message Beans (EJB)
Enterprise Java Beans

• Entity beans are EJBs that are created to encapsulate some data contained by the system (such as a row in the database or, more generally, to an entry that exists in persistent storage)

• Entity beans can be thought of as the "nouns" of a particular problem domain that they are implemented to solve

• The data can be created, searched for (found) or removed by clients

• The data can also be retrieved and updated by clients

• In addition to the usual EJB implementation, home interface and remote interface, Entity beans also have special primary key classes defined for them that relate to the primary keys of an associated entity stored in the database

• Entity beans are very different from Session beans. Entity beans:
  - can be used concurrently by several clients
  - are long-lived
  - they are intended to exist beyond the lifetime of a client
  - will survive server crashes
  - directly represent data in a database

• Because clients could potentially hold many entity references as a result of a find operation, many effective EJB designs will only provide access to entity beans via session beans and limit how many entity bean handles are returned to the client

• Scalability of the system can become compromised if too many remote entity bean references are handed out to clients

• Articles describing various patterns for using entity beans are listed in the references

• Entity beans assume that more than one client will use them concurrently. Interactions with the bean are moderated by the container so as to maintain the integrity of the underlying data

• The container may accomplish this by either:
  - queuing client requests one at a time
  - creating an instance of the bean for each client and relying on the underlying database to deal with synchronisation issues

• Remember, session beans do not support concurrent access. A stateful bean is an extension of the client and stateless beans don't

• Entity beans can be thought of as the "nouns" of the problem domain in an EJB application. Each entity bean corresponds to one entity in the database

• Entity beans are EJBs that are defined outside the application and are accessed by clients via the session beans

• Entity beans are defined in their own interface, which may be inferred from the associated entity's interface

• In addition to the usual EJB implementation, home interface and remote interface, Entity beans also have special primary key classes defined for them that relate to the primary keys of an associated entity stored in the database

• Entity beans are very different from Session beans. Entity beans:
  - can be used concurrently by several clients
  - are long-lived
  - they are intended to exist beyond the lifetime of a client
  - will survive server crashes
  - directly represent data in a database
Entity Beans – persistence

• Entity beans represent data in an underlying database.

• Their lifetime matches the lifetime of the underlying data.

• Entity beans can be removed in one of two ways:
  – A client explicitly invokes the remove() method of the entity bean,
  – The data for the entity is deleted from the underlying database.

• Entity beans typically map to data in an underlying database
  – Usually to a single row in a table.

  The instance variables in the Entity bean map directly to the fields in

• 2 Types of Entity beans
  – Bean Managed Persistence (BMP)
  – Container Managed Persistence (CMP)

Entity Beans – BMP vs. CMP

• Bean Managed Persistence (BMP)
  – bean developer takes control of bean’s persistence
    – eg: use JDBC and SQL directly

• Container Managed Persistence (CMP)
  – container manages the process of saving and restoring
    the state of beans
  – the bean’s implementation code is generated by the container
  – Bean Managed Persistence (BMP)

Entity Beans – primary keys

• Each Entity bean instance has a primary key.

• A primary key is:
  – A value (or combination of values, called compound primary key)
    that uniquely identifies the entity

• Implemented as a Java class NOT a primitive (e.g., java.lang.Integer vs int)

• You can write your own class to represent a primary key
  – Must be serializable ie: implement the java.io.Serializable
  – Must have a no argument constructor
  – May have a constructor with the compound key values
  – Must implement a hashcode() and equals() method

  – Easiest way is to use code generator to do this
    eg: Eclipse -> Source -> generate constructor from fields
    Source -> generate hashCode() and equals()

Entity Beans – primary keys

```java
class CustomerPK implements Serializable {
    public int custId;
    public String surname;

    public CustomerPK(int custId, String surname) {
        this.custId = custId;
        this.surname = surname;
    }

    public int hashCode() {
        final int PRIME = 31;
        int result = 1;
        result = PRIME * result + custId;
        result = PRIME * result + ((surname == null) ? 0 : surname.hashCode());
        return result;
    }

    public boolean equals(Object obj) {
        if (custId != other.custId) return false;
        if (surname.equals(other.surname)) return false;
        return true;
    }
}
```

Entity Beans lifecycle

• An entity bean can have one of these states:

  • Does-not-exist state
    • not yet been instantiated. This state provides a beginning and an end for the life cycle of a bean instance.

  • Pooled state
    • bean has been instantiated by the container but not associated with an EJB object.

  • Ready state
    • it has been associated with an EJB object and is ready to respond to business method invocations.

  Business method(s)
  – newInstance()
  – setEntityContext(ec)
  – unsetEntityContext()
  – ejbCreate(args)
  – ejbPostCreate(args)

  Container managed persistence (CMP)
  – Bean managed persistence (BMP)
When using CMP the container will implement most of the methods for:

- All entity bean implementation classes must implement the methods that need to be implemented depending on the persistence model you are implementing – BMP or CMP.

- BMP
  - Cons
  - Can better accommodate a complex or unusual set of data
  - More work is required
  - Gives you more flexibility

- CMP
  - Pros
  - Doesn't have to code the database access calls in the entity bean
  - Can better accommodate a complex or unusual set of data
  - More work is required
  - Gives you more flexibility

- EJB 2.0 only supported BMP
- BMP always maps a single row of data (this corresponds to an INSERT on the database)

EJB 2.0 CMP Entity Bean (1)

- The container will call the ejbCreate() method
- The container will call the ejbActivate() method
- The container will call the ejbPassivate() method
- The container will call the ejbRemove() method
- EJB 2.0 CMP Entity Bean (2)

- Existing entity beans are loaded into the container via the deployment descriptor
- The developer must create the primary key class used to find instances of an existing bean.

EJB 2.0 CMP Entity Bean (3)
EJB 2.0 CMP Entity Bean (3)

Entity EJB’s have additional parameters in the ejb-jar.xml Deployment Descriptor (DD):

```xml
<ejb-jar>
  <enterprise-beans>
    <entity>
      // usual EJB stuff
      <ejb-name>Customer</ejb-name>
      <home>bank.CustomerBeanHome</home>
      <remote>bank.CustomerBean</remote>
      <ejb-class>bank.CustomerBean</ejb-class>
      // set as CMP
      <persistence-type>Container</persistence-type>
      // define primary key as Integer type
      <prim-key-class>java.lang.Integer</prim-key-class>
      <reentrant>False</reentrant>
      // important! We are using CMP 2.0
      <cmp-version>2.x</cmp-version>
      // continued from prior page
      <abstract-schema-name>CustomerBean</abstract-schema-name>
      // define the fields – these correspond to a column in the table
      <cmp-field>
        <field-name>Id</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>Name</field-name>
      </cmp-field>
      // define the primary key – the container generates findPrimaryKey() method!
      <primkey-field>Id</primkey-field>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

EJB 2.0 CMP Entity Bean (4)

• You will also need 2 extra deployment descriptors
  – weblogic-ejb-jar.xml for weblogic specific parameters
  – weblogic-cmp-rdbms-jar.xml how weblogic maps the datasource name and table column names to the CMP bean.

Entity Beans – Summary

• Entity beans represent persistent data
• Two types of entity beans: CMP and BMP
• Entity beans must have a primary key class defined
• BMP must provide database statements for:
  – ejbCreate(), ejbRemove(), ejbFindByPrimaryKey(), ejbFindXXX()
  etc.
• More deployment descriptor (ejb-jar.xml) tags:
  – CMP fields (only for CMP EJBs)
  – Primary key class
  – Abstract schema name (only for EJB 2.0)
• Still need extra deployment descriptor for CMP entity beans (depending on the Application Server) to map bean to database table.
  – weblogic-cmp-rdbms-jar.xml for WebLogic.

EJB – Exception Types

• Note: javax.ejb.EJBObject and javax.ejb.EJBHome are subclasses of java.rmi.Remote.
  ie: your Home & Implementation will throw the java.rmi.RemoteException from each of their methods
  (and you will need to catch them in your client ☺)

• There are other exceptions thrown by EJBs eg:
  – CreateException, DuplicateKeyException
  – FinderException, ObjectNotFoundException
  – RemoveException
  – EJBException, NoSuchEntityException

EJB – Exception Types (cont)

Perrone et al “Building Java Enterprise Systems with J2EE” Sams 2000 fig 36.2
**EJB – Exception Types (cont)**

- **EJBException**
  - Thrown by an EJB when an application-specific method cannot be completed.

- **NoSuchEntityException**
  - Thrown by an EJB when an application-specific method cannot be completed because a particular entity bean does not exist.

- **CreateException**
  - Thrown when an EJB cannot be created (invalid arguments).

- **DuplicateKeyException**
  - A subclass of CreateException, thrown when a particular entity bean cannot be created because objects with the same key already exist.

- **RemoveException**
  - Thrown when a particular EJB cannot be removed.

- **FinderException**
  - Thrown by all query methods (find or select) when an application error (business-logic) occurs.

- **ObjectNotFoundException**
  - Thrown when a singular EJB cannot be found.

---

**Local Client API**

- **Only remote interfaces in EJB 1.1**
- **EJB 2.0 introduced local component interfaces**

**Local Interface**

```java
package bank;
import java.rmi.*;
import javax.ejb.*;

interface LocalBankTeller extends EJBLocalObject {
    String sayHello();
    void deposit(String acctnum, Float amount);
    void withdraw(String acctnum, Float amount);
}
```

**Notes:**
- Business logic methods for clients in the same container
- None of the methods throw RemoteException
- Identical to remote interface except for a couple of differences

---

**Local Home Interface**

```java
package bank;
import java.rmi.*;
import javax.ejb.*;

interface LocalBankTellerHome extends EJBLocalHome {
    public BankTeller create() throws CreateException;
}
```

**Notes:**
- None of the methods throw RemoteException

---

**EJB Local Interface**

```xml
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>BankTellerEJB</ejb-name>
            <home>bank.BankTellerHome</home>
            <remote>bank.BankTeller</remote>
            <local-home>bank.LocalBankTellerHome</local-home>
            <local>bank.LocalBankTeller</local>
            <ejb-class>bank.BankTellerBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
</ejb-jar>
```

---

**Local Client API (cont’d)**

- Similar to Remote Client API but less complicated
- Other opens-to-optional in the same EJB container
- Defines the bean’s lifecycle methods that can be used by Local Component Interface
- Other opens-to-optional that are compiled
- Other opens-to-optional that are compiled
- Other opens-to-optional that are compiled
- EJB 2.0 introduced local component interfaces
- Only remote interfaces in EJB 1.1

---

**DD for Local Client**

- EJB 2.0 introduced local component interfaces
- Code generation – can be generated from the EJB bean
- EJB 2.0 introduced local component interfaces
- Only remote interfaces in EJB 1.1

---

**Local Interface**

- Local interface
- Local interface
- Local interface
- Local interface
- Local interface
- Local interface
- Local interface
- Local interface
EJB 2.0 CMP: EJB QL

- No standard language prior to EJB 2.0
- EJB QL is a language similar to SQL-99
- EJB QL statements are specified in the deployment descriptor and the container will generate SQL
- EJB QL is an attempt to map Java objects and rows/columns
- Use EJB QL:

Message Driven Beans

- Unlike session or entity beans, MDBs have no home or remote interface
- Clients interact with MDBs only indirectly, by sending a JMS message to a JMS Queue or Topic
- The EJB container automatically creates and removes MDB instances as needed to process incoming messages
- The goal of message-driven beans is to provide a queue of the messages received by the EJB container
- MDBs act as a message listener for JMS, listening for incoming messages and notifying the EJB container when a message arrives
- MDBs are stateless and are not responsible for maintaining state information
Message Driven Beans

- Queue or Topic is defined in the deployment descriptor
- Message selector is also defined in DD - restrictive

- MDB must implement `javax.jms.MessageListener` and `javax.ejb.MessageDrivenBean`
- `onMessage()` must not throw JMS or application exceptions; instead, they must be caught or handled. At least, they could be re-thrown as `EJBException`

- MessageDrivenBean specifies two methods:
  - `ejbRemove()` - is invoked just before the bean is removed
  - `setMessageDrivenContext(MessageDrivenContext ctx)` - sets the context for the bean

- `ejbCreate()` needs to be defined as well

MDB Life Cycle

- Does not exist
- Ready pool
- `onMessage()`
- `newInstance()`
- `setMessageContext(mdc)`
- `ejbCreate()`
- `ejbRemove()`

Example – Message-driven bean

```java
import javax.ejb.*;
import javax.jms.*;

public class MessageTraderBean implements MessageDrivenBean, MessageListener {
    public void ejbCreate() throws CreateException {
        // ejbCreate logic
    }
    public void ejbRemove() {
        // ejbRemove logic
    }
    public void setMessageDrivenContext(MessageDrivenContext ctx) {
        // setMessageDrivenContext logic
    }
    public void onMessage(Message msg) {
        try {
            // process message
        } catch (EJBException ex) {
            // handle exception
        }
    }
}
```

Example – Message-driven bean ejb-jar.xml

```xml
<ejb-jar>
    <enterprise-beans>
        <message-driven>
            <ejb-name>exampleMessageDriven</ejb-name>
            <ejb-class>examples.ejb20.message.MessageTraderBean</ejb-class>
            <transaction-type>Container</transaction-type>
            <message-driven-destination>
                <jms-destination-type>javax.jms.Topic</jms-destination-type>
            </message-driven-destination>
            <security-identity>
                <run-as-specified-identity>
                    <role-name>foo</role-name>
                </run-as-specified-identity>
            </security-identity>
        </message-driven>
    </enterprise-beans>
</ejb-jar>
```

Enterprise Java Beans

- Last lesson we looked at:
  - Introduction to EJB
  - EJB basics
  - EJB Architecture
  - Session Beans
  - EJB Clients
  - EJB Development Process

- This lesson will cover:
  - Entity Beans
  - Message Driven Beans

EJB – Clustering Issues

- Highly scalable, mission critical enterprise applications will inevitably require more than a single application server to deliver services with minimal downtime and maximum performance
- In these environments, clustering is used to guarantee availability and scale to meet the demands placed on the system
- A cluster is a group of application servers that transparently run your J2EE application as if it was a single entity
- Scaling is accomplished by adding extra machines to the cluster.
  - Multiple machines provide redundancy that increases availability
EJB – Clustering Issues (cont)

• Clustering support isn’t defined in the J2EE specification – application server vendors are left to implement clustering in whatever manner they see fit.

Most J2EE application servers implement clustering around their implementation of JNDI.

1. Independent – each app server has its own independent JNDI tree
2. Centralised – a single, centralised JNDI tree is maintained for all app servers
3. Shared Global – each app server shares a global JNDI tree with the cluster. All objects bound into the JNDI tree are broadcast to all servers in the cluster. This is the mechanism used by WebLogic

You do have to be aware of how clustering is implemented in your application server when deploying a J2EE application to a cluster.

The relevant deployment properties will be container specific.

See the WebLogic server documentation for a description of how clustering of EJBs (and other components) is configured.

EJB – Clustering Issues (cont)

Alternatives?

Commercial products such as Oracle’s TopLink provide an alternative mechanism.

You can also use open source products such as JBoss.

JPA – Java Persistence API (JPA)

JPA allows new persistence mechanisms.

EJB Key Features Summary

They provide a model for defining server-side components.

They provide a model for defining distributed client interfaces to the services provided by these components.

They provide standard operations and semantics for allowing a container to create, destroy, allocate, persist and activate component instances.

They provide a standard model for defining a component that maintains a conversational session with a client, with session management handled by the container.

They provide a standard model for defining a component that encapsulates a data source (such as a database entry), with object-relational mapping handled by the container.

EJB Key Features Summary (cont)

EJB’s – Cautions

EJB 1.1 only had remote interfaces.

This meant that clients had to use the network to connect to EJBs, even on the same physical server!!

+/barb2right

Use Local interface instead.

+ new EJB-QL and selector methods available.

– But this means the session bean will run in the same JVM as the client.

Use EJBs where appropriate.

EJB containers have high overhead (and cost $$).

Sometimes using lightweight Web Containers such as Tomcat + JDBC/stored procedures is more appropriate.

Java Data Objects (JDO) provides an alternative mechanism.

See http://java.sun.com/products/jdo/index.jsp

Alternatively?

Java EE 5 has a new persistence mechanism based on Java 5 annotations.

– The Java Persistence API (JPA)

EJB 3.0 based on JPA

Alternatives?

Java EE 5 has a new persistence mechanism.

Java Persistence API (JPA)

EJB 3.0 based on JPA

Java EE 5 has a new persistence mechanism.

Java Persistence API (JPA)

Java EE 5 has a new persistence mechanism.

Java Persistence API (JPA)
EJB 3.0

• Just plain old Java Objects! Eg: BankTeller:
  @remote
  public interface BankTellerRemote {
  public String sayHello(String name);
  }

  @stateless(name="BankTeller")
  public class BankTeller implements BankTellerRemote {
  public String sayHello(String name) {
    return "Hello, " + name;
  }
  }

• Note: no XML, no "housekeeping" code!

EJB 3.0 persistence

• Eg: Account
  @Entity
  @Table (name="Accounts")
  @NamedQuery (name = "findByAcct",
  query="select Object(a) from Accounts where a.acct = ?1")
  public class Account implements Serializable {
    @Id // next field is primary key
    @column (name="Acct")
    public Integer getAcctNum() { … }
    @column (name="Balance")
    public float getBalance() { … }

…

Summary

• The past two lessons have introduced the component model for Java enterprise applications – Enterprise Java Beans
• We have touched on a number of aspects of EJB architecture and development
• You should now understand the different types of EJB, when they should be used, how they are developed and deployed, and how they are used from client applications
• EJBs are a fairly complex topic, and you will need to do additional study to fully understand the intricacies of EJB design and development