TDDD05
Enterprise Java Beans - Part 1
Lecture 07

Definition

- **EJB** – component model for building distributed server-side Java-based enterprise application components
- An **EJB** – distributed server-side non-visual component encapsulating application business logic

Developers say

- **EJB** is a piece of code executing in a special container
- **EJB = Bean**
  - Bean encapsulates business logic
  - Distributed objects with distinct address spaces
  - Programmer writes Beans to a standard
  - Portability and scalability
  - Components with lifecycle management
  - Transactional access to remote objects
  - Container services; out of the box
  - \{ ... \}

EJBs Provide (1)

- **Model** for defining server-side components
- **Model** for defining distributed client interfaces to services provided by server-side components
- **Framework** for building server-side components
- **Standard operations and semantics** for allowing a container to create, destroy, allocate, persist, activate, and invoke component instances
- **Model** for defining a component that maintains a session with a client where the session is managed by the container

EJBs Provide (2)

- **Model** for defining a component that encapsulates a data source entry with object-to-relational data mapping handled by the container – a.k.a. Entity bean (EJB2) or JPA Entity (EJB3)
- **Model** for defining a component that encapsulates an asynchronous message consumer with messaging service handled by the container – a.k.a. Message-driven beans
- **Model** for defining a component that encapsulates a Web service with SOAP messaging interactions handled by the container

EJBs Provide (3)

- **Standard** for configuring and deploying distributed components
- **Standard** for declaratively specifying the security attributes of a component
- **Standard** for declaratively specifying the transactions attributes of a component
- **Standard** component interface contract allowing components to run in any vendor-compliant container/server which implements this standard interface contract
Implementation

- **EJB ∈ Java EE standard** (a.k.a. J2EE)
  - Component specification
  - Distributed component model standard
- **Implementation** by independent vendors
  - Tools and Containers
    - Proprietary: IBM (WebSphere), BEA (WebLogic), Sun and Netscape (Planet), Oracle, Borland
    - Open source: GlassFish, JBoss

EJB vs. JavaBeans

- **JavaBeans model** – a means for building Java components
- **Enterprise JavaBeans model** – a means for building Java components for use in containers that offer distributed client connectivity, have exclusive server-side semantics, provide various standard services and offer sophisticated component lifecycle management

EJB & Java Evolution

- **EJB 3.1**
  - Java EE 6
  - Web Services
  - JPA
  - POJOs
- **EJB 3.0**
  - Java EE 5
  - JPA
- **EJB 2.0**
  - Java EE 4
  - JPA

EJB Architecture Basics

- **EJB Architecture Basics (1)**
- **EJB Architecture Basics (2)**
EJB Architecture Basics (3)

Distributed Objects (1)

Distributed Objects (2)

Distributed Objects (3)

Discover EJBs – JNDI (1)

Discover EJBs – JNDI (2)

Java Naming and Directory Interface (JNDI)
- Similar to CORBA naming service

4/26/13
Discover EJBs – JNDI (3)

Java Naming and Directory Interface (JNDI)

- Centralized repository
  - No metalevel descriptions
  - Resources identified by names submitted to JNDI server
    - Example: mappedName = "ejb/TODOSYS" (see below)
    - Client must know the name of EJB to get it
      - No metalevel description of EJB component semantics
      - Not possible: Get me EJB that computes (a + b)
      - Possible: Get me EJB with name "A"

```java
import javax.ejb.Stateless;

@Stateless (mappedName="ejb/TODOSYS")
public class SimpleSessionBean implements SimpleSession {
...
```
**EJB Object – Remote Interface (2)**
- Acts as Proxy
  - One per client request and Bean type (component type)
  - Either none or single associated Bean instance
  - Access to container services
  - Access to Bean configuration

**EJB Object – Remote Interface (3)**
- Must extend `javax.ejb.EJBObject`
- Lists business methods available to clients
- Created via `EJBHome` object factory

```java
public interface SingleSession extends javax.ejb.EJBObject {
    public String sayHello() throws javax.rmi.RemoteException;
}
```

**EJB Home – Home Interface (1)**
- Must extend `javax.ejb.EJBHome`
- Factory to create EJB Object instances
- Singleton – one instance per EJB component
- Allows clients to create / remove / locate EJBs
- Generated by the container
- Registered to JNDI
- Access point for clients

```java
public interface SingleSessionHome extends javax.ejb.EJBHome {
    SingleSession create() throws javax.rmi.RemoteException,
            javax.ejb.CreateException;
}
```

**EJB 2 Architecture (3)**
- Combine EJB Home and EJB Object in one?
- No good, because
  - EJB Home is a Singleton
    - One factory per EJB component sufficient
    - Can create as many EJB Object instances as needed
  - EJB Home as Singleton is good OO – reduces code duplication
  - Separation of Concerns (see AOP lecture)
    - EJB Home encapsulates no Bean's Business Logic
    - EJB Home encapsulates EJB framework code
    - Create / destroy / locate EJBs

```java

**Discover EJBs (1)**
- Remote Interfaces
- One instance of type B
- EJB Object – EJB object for Beans of type B
- EJB Home – EJB home object for Beans of type B
```
Discover EJBs (2)

- Classical JNDI lookup (cf. DI in EJB 3)
  - Gives the Client a reference to EJBJava Object
  - Clients reference JNDI server via InitialContext (see example)
- JNDI implementation
  - Vendor specific
  - Bound to J2EE server implementation

```java
context = new InitialContext();
HelloWorldHome ejbHome = (HelloWorldHome) context.lookup("HelloWorld");
HelloWorld ejObject = ejbHome.createObject();
String message = ejobject.speakHello();
```

Session Beans (1)

- Distributed objects, components
  - Home Interface, Remote / Local interface, Bean class
- Act as agents to the client
  - No DB persistence
  - Low resource requirements
  - Won’t survive a server crash or shutdown
  - Fast response
  - Perfect for short requests to perform a unit of work

Types
- Stateless
- Stateful

Session Beans (2)

Stateless

- Single request business model
  - No client – conversational – state maintained across client requests
- Conversational state spans over a single method call
  - Container may destroy / return to pool Bean instance after each method call
- Client-independent state can be kept in member variables
- Bean member variables can be populated at server start
- Heavy computations done once
- Data provided as parameters or retrieved
- Require little container resources
  - No state
  - Data always in RAM or destroyed
  - Efficient Instance Pooling

[See also Instance Pooling under the <<Container Services>> section below]

Session Beans (3)

Stateful (1)

- Designed to service business processes that span over multiple method calls (requests) or transactions
  - Conversational state
  - Maintained by EJB container
  - Maintained for lifetime of Beans service to client
  - Maintained in container cache
- Require more resources from the container
- No Bean Instance Pooling
  - Activation / Passivation instead

[See also Instance Pooling under the <<Container Services>> section below]

Session Beans (4)

Stateful (2)

- Limitations and Alternatives
  - Performance
    - Concurrency state comes at a cost
  - Memory
    - Complex states require much memory
  - Alternatives
    - Combine Stateless Bean with Persistence
    - Code state persistence yourself (memory, file)
  - Maintain session in Web Container instead (Java Servlet API)

No, No!

- More under Dependency Injection in <<Container Services>> section below
- Use (instantiate) Stateful EJB in a Stateless EJB
- Conversational state of a client may leak
Entity Beans (1)

- **Reusable EJB component with automated persistence**
  - **View into a data source a.k.a. relational database**
    - Represent business data stored in a database
    - Bean instance map to database table records
    - Changes in Bean properties (Bean state) persisted automatically to associated data source
  - Data source outside Java EE application
  - Cf. secondary storage for Stateful Session Beans inside Java EE

- **Distributed object**
  - Alike Session Beans and Message-driven Beans
  - Can have remote clients from outside JVM
  - Cf. EJB 3 JPA Entity
  - JPA Entity – POJO, not distributed object
  - All clients must be local to the JVM

Entity Beans (2)

- **EJBHome contains**
  - **create()**
  - **findByIdPrimaryKey()**
  - Optionally other **finder** method declarations

- **Bean class contains**
  - **Properties**
  - **Getters / Setters**
  - **Callbacks**
    - **ejbCreate()**
    - **ejbStore()**
    - **ejbRemove()**
    - **ejbActivate()**
    - **ejbPassivate()**
  - **setEntityContext()**

Entity Beans (3)

- **EJB 2 Persistence (1)**
  - **Object to relational database mapping (common)**
  - **Object databases (less common)**
  - Container generates persistence code
  - All in case of Container Managed Persistence
  - Parts in case of Bean Managed Persistence
  - EJB-QL, query language
  - No client SQL code (SELECT, etc.)

Entity Beans (4)

- **EJB 2 Persistence (2) [ programmer must choose ]**
  - **Container managed (CMP, implicit)**
    - Entity Bean defined as Abstract class
    - Abstract getters / setters
    - At deployment container creates concrete implementation classes
  - **Bean managed (BMP, explicit)**
    - Entity Bean defined as a class
    - Programmer codes getters / setters using JDBC
    - Programmer also codes using JDBC
      - **Callbacks**
        - ejbCreate()
        - ejbStore()
        - ejbFindByPrimaryKey()
      - Other **finder** methods implementations

Entity Beans (5)

- **Traditional vs. EJB Persistence**
  - **Traditional DB persistence (manual, explicit)**
    - Each client requires a DB connection
    - Business logic resides in both client and DB
    - Little reusability
  - **Entity Beans (implicit persistence)**
    - Container handles DB connections
    - Business logic in a server side EJB
    - Further free container services
      - Transactions
      - Redundancy
      - Security

[ See also "Container Services" section below ]

Entity Beans (6)

- **Limitations**
  - Record oriented
    - No business logic beyond getters / setters and basic validation
    - No inheritance
      - Semantics of Table1.id inherits Table2.id is undefined
    - No polymorphism
    - No good for Domain-driven Design (DDD)
    - Not usable outside EJB container
    - Difficult to test
**Message-Driven Beans (1)**
- No Home, Remote or Local interfaces
- Have a single business method
  - onMessage()
- No static type check
- No return values
- No exceptions
- Stateless

**Message-Driven Beans (2)**
- Why MDBs?
  - Performance
  - Reliability
  - Support for multiple senders and receivers
  - “Easy” integration with legacy systems

**EJB 2 elements**
- Enterprise Bean class
- Supporting classes
- EJB Object
- Remote interface
- Home object
- Deployment descriptor (XML)
- Vendor-specific files
- (Local interface)

**EJB 2 Deployment**
- EJB deployment descriptor (XML)
- ejb-jar.xml
- Attributes of the beans specified declaratively
- Deployment descriptor language is a composition language
- EJB-jar file is verified by container
- Container generates stubs and skeletons

**EJB 2 Final thoughts**
- Not object-oriented
  - Data and operations separated
  - Entity beans encapsulate data (DB record-oriented)
  - Session beans encapsulate functionality (no data)
  - No component inheritance (EJB 2.0)
  - EJB 3.0 – Beans are POJOs
    - Component inheritance implemented as classical OO inheritance
- Development and Application
  - Strict architecture
  - Complex
  - Beans are difficult to test
    - Container required
    - Deployment errors mistaken for Business Logic errors

**XDoclet**
- Deployment descriptor
- Generate from declarative specification
  - Remote interface
  - home interface
  - local interface
  - local home interface
  - primary key class
- Specification as comments in the Bean class