Enterprise JavaBeans™ 3.0

Linda DeMichiel

Sun Microsystems, Inc.

TS-3396

Goal of This Talk

Learn how to use the new EJB™ 3.0 API to simplify your applications

Agenda

EJB 3.0 approach

Simplified Bean classes

Client view

Using container services

Current status and summary

Motivation

- EJB™ 2.1 technology very powerful, but too complex
- Original goal of EJB technology: providing a managed environment to aid enterprise application development

EJB Container

- Managed environment for the execution of components
- Container interposes to provide services
- Container-provided services
  - Concurrency
  - Transactions
  - Environment access
  - Security
  - Distribution
  - EIS integration
  - Resource pooling
  - Persistence

Problem

- APIs to support use of services were designed for container, not application
  - EJBHome interface
  - EJBObject interface
  - EnterpriseBean interfaces
  - JNDI interfaces
  - Deployment descriptor
  - …
- They got the job done, but at the cost of complexity and heavy-weight component programming model

Example

```
// EJB 2.1 Stateless Session Bean: Bean Class

public class PayrollBean
             implements javax.ejb.SessionBean {
    SessionContext ctx;
    DataSource payrollDB;

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }

    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
```

Example

```
// EJB 2.1 Stateless Session Bean: Bean Class (continued)

    public void ejbCreate() {
        ...
        Context initialCtx = new InitialContext();
        payrollDB = (DataSource)initialCtx.lookup
    ("java:com/env/jdbc/empDB");
        ...
    }

    public void setTaxDeductions(int empId,int deductions)
    {
        ...
        Connection conn = payrollDB.getConnection();
        Statement stmt = conn.createStatement();
        ...
    }
}
```

Example

```
// EJB 2.1 Stateless Session Bean: Interfaces

public interface PayrollHome
                extends javax.ejb.EJBLocalHome {

    public Payroll create() throws CreateException;
}

public interface Payroll
                extends javax.ejb.EJBLocalObject {

    public void setTaxDeductions(int empID, int
deductions);

}
```

Example

```
// EJB 2.1 Stateless Session Bean: Deployment Descriptor

<session>
    <ejb-name>PayrollBean</ejb-name>
    <local-home>com.example.PayrollHome</local-home>
    <local>com.example.Payroll</local>
    <ejb-class>com.example.PayrollBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <resource-ref>
        <res-ref-name>jdbc/empDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</session>
```

Example

```
// Deployment Descriptor(continued)
<assembly-descriptor>
    <method-permission>
        <unchecked/>
        <method>
                <ejb-name>PayrollBean</ejb-name>
                <method-name>*</method-name>
        </method>
    </method-permission>
    <container-transaction>
        <method>
                <ejb-name>PayrollBean</ejb-name>
                <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
```

EJB 3.0 Goal

FIX THIS!

How?

EJB 3.0 Approach

- More work is done by container, less by developer
- Inversion of contractual view
- Contracts benefit developer rather than container
  - Bean specifies what it needs through metadata
  - No longer written to unneeded container interfaces
  - Deployment descriptor no longer required
  - Configuration by exception
- Container provides requested services to bean

Compatibility Constraints

- Existing EJB applications work unchanged
- New EJB components interoperate with existing EJB components
- Components can be updated or replaced without change to existing clients
- Clients can be updated without change to existing components
- Compatibility with other Java EE 5 APIs

Agenda

EJB 3.0 approach

Simplified Bean classes

Client view

Using container services

Current status and summary

Bean Classes

- In EJB 3.0, session beans, message-driven beans are ordinary Java classes
  - Container interface requirements removed
  - Bean type specified by annotation or XML
- Annotations
  - @Stateless, @Stateful, @MessageDriven
  - Specified on bean class
- EJB 2.x entity beans are unchanged
  - Java™ Persistence API entities provide new functionality
  - @Entity applies to Java Persistence API entities only

Example

```
// EJB 2.1 Stateless Session Bean: Bean Class

public class PayrollBean
                implements javax.ejb.SessionBean {
    SessionContext ctx;
    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }
    public void ejbCreate() {...}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}

    public void setTaxDeductions(int empId, int
    deductions) {
        ...
    }
}
```

Example

```
// EJB 3.0 Stateless Session Bean: Bean Class

@Stateless public class PayrollBean implements Payroll {

    public void setTaxDeductions(int empId,int deductions)
    {
        ...
    }
}
```

Business Interfaces

- Plain Java language interface
  - Unless you want something else
  - EJBObject, EJBHome interface requirements removed
- Either local or remote access
  - Local by default
  - Remote by annotation or deployment descriptor
  - Remote methods not required to throw RemoteException
- Bean class can implement its interface
- Annotations: @Remote, @Local, @WebService
  - Can specify on bean class or interface

Example

```
// EJB 2.1 Stateless Session Bean: Interfaces

public interface PayrollHome
                extends javax.ejb.EJBLocalHome {

    public Payroll create() throws CreateException;
}

public interface Payroll
                extends javax.ejb.EJBLocalObject {

    public void setTaxDeductions(int empId, int
deductions);

}
```

Example

```
// EJB 3.0 Stateless Session Bean: Business Interface

public interface Payroll {

    public void setTaxDeductions(int empId, int
deductions);

}
```

Example

```
// EJB 3.0 Stateless Session Bean: Remote Interface

@Remote public interface Payroll {

    public void setTaxDeductions(int empId, int
deductions);

}
```

Example

```
// EJB 3.0 Stateless Session Bean:
// Alternative: Remote Interface specified on bean class

@Stateless @Remote public class PayrollBean
       implements Payroll {

    public void setTaxDeductions(int empId,int deductions)
    {
        ...
    }
}
```

Message-Driven Beans

- Message listener interface is business interface
  - Bean class implements it or designates with @MessageListener
- No requirement to implement other interfaces
- Annotations
  - @MessageDriven

Example

```
// EJB 3.0 Message-driven bean: Bean Class

@MessageDriven public class PayrollMDB
       implements javax.jms.MessageListener {

    public void onMessage(Message msg) {
        ...
    }
}
```

Environment Access

- By dependency injection or simple lookup
  - Use of JNDI interfaces no longer needed
- Specify dependencies by annotations or XML
- Annotations applied to:
  - Instance variable or setter property => injection
  - Bean class => dynamic lookup

Environment Access Annotations

- @Resource
  - For connection factories, simple environment entries, topics/queues, EJBContext, UserTransaction, etc.
- @EJB
  - For EJB business interfaces or EJB Home interfaces
- @PersistenceContext
  - For container-managed EntityManager
- @PersistenceUnit
  - For EntityManagerFactory

Dependency Injection

- Bean instance is supplied with references to resources in environment
- Occurs when instance of bean class is created
- No assumptions as to order of injection
- Optional @PostConstruct method is called when injection is complete

Example

```
// EJB 3.0 Stateless Session Bean: Bean Class
// Data access using injection and Java Persistence API

@Stateless public class PayrollBean implements Payroll {

    @PersistenceContext EntityManager payrollMgr;

    public void setTaxDeductions(int empId,int deductions)
    {
       payrollMgr.find(Employee.class,
    empId).setTaxDeductions(deductions);
    }
}
```

Dynamic Environment Lookup

- Use EJBContext lookup method
- Dependencies declared using annotations on bean class

Example

```
// EJB 3.0 Stateless Session Bean
// Using dynamic lookup

@PersistenceContext(name="payrollMgr")
@Stateless public class PayrollBean implements Payroll {

    @Resource SessionContext ctx;

    public void setTaxDeductions(int empId,int deductions)
    {
      EntityManager payrollMgr = ctx.lookup("payrollMgr");
      payrollMgr.find(Employee.class,
    empId).setDeductions(deductions);
    }
}
```

Agenda

EJB 3.0 approach

Simplified Bean classes

Client view

Using container services

Current status and summary

Simplification of Client View

How?

- Use of dependency injection
- Simple business interface view
- Removal of need for Home interface
- Removal of need for RemoteExceptions
- Removal of need for handling of other checked exceptions

Example

```
// EJB 2.1: Client View

...
Context initialContext = new InitialContext();
PayrollHome payrollHome = (PayrollHome)
   initialContext.lookup("java:comp/env/ejb/payroll");

Payroll payroll = payrollHome.create();
...


// Use the bean

payroll.setTaxDeductions(1234, 3);
```

Example

```
// EJB 3.0: Client View

@EJB Payroll payroll;


// Use the bean

payroll.setTaxDeductions(1234, 3);
```

Removal of Home Interface

- Stateless Session Beans
  - Home interface not needed anyway
  - Container creates or reuses bean instance when business method is invoked
    - EJB 2.1 Home.create() method didn't really create
- Stateful Session Beans
  - Container creates bean instance when business method is invoked
  - Initialization is part of application semantics
    - Don't need a separate interface for it!
- Both support use of legacy home interfaces

Agenda

EJB 3.0 approach

Simplified Bean classes

Client view

Using container services

Current status and summary

Transactions

Transaction Demarcation Types

- Container-managed transactions
  - Specify declaratively
- Bean-managed transactions
  - UserTransaction API
- Container-managed transaction demarcation is default
- Annotation: @TransactionManagement
  - Values: CONTAINER (default) or BEAN
  - Annotation is applied to bean class (or superclass)

Container-managed Transactions

Transaction Attributes

- Annotations are applied to bean class and/or methods of bean class
  - Annotations applied to bean class apply to all methods of bean class unless overridden at method-level
  - Annotations applied to method apply to method only
- Annotation: @TransactionAttribute
  - Values: REQUIRED (default), REQUIRES_NEW, MANDATORY, NEVER, NOT_SUPPORTED, SUPPORTS

Example

```
// EJB 3.0: Container-managed transactions

@Stateless public class PayrollBean implements Payroll {

    @TransactionAttribute(MANDATORY)
    public void setTaxDeductions(int empId,int deductions)
    {
        ...
    }

    public int getTaxDeductions(int empId)
    {
        ...
    }
}
```

Example

```
// EJB 3.0: Container-managed transactions

@TransactionAttribute(MANDATORY)
@Stateless public class PayrollBean implements Payroll {

    public void setTaxDeductions(int empId,int deductions)
    {
        ...
    }

    @TransactionAttribute(REQUIRED)
    public int getTaxDeductions(int empId)
    {
        ...
    }
}
```

Example

```
// EJB 3.0: Bean-managed transactions

@TransactionManagement(BEAN)
@Stateless public class PayrollBean implements Payroll {

    @Resource UserTransaction utx;
    @PersistenceContext EntityManager payrollMgr;

    public void setTaxDeductions(int empId, int
    deductions) {
        utx.begin();
        payrollMgr.find(Employee.class,
    empId).setDeductions(deductions);
        utx.commit();
    }
...
}
```

Security

Aspects

- Method permissions
  - Security roles that are allowed to execute a given set of methods
- Caller principal
  - Security principal under which a method is executed
    - @RunAs for run-as principal
- Runtime security role determination
  - isCallerInRole, getCallerPrincipal
    - @DeclareRoles

Method Permissions

- Annotations are applied to bean class and/or methods of bean class
  - Annotations applied to bean class apply to all methods of bean class unless overridden at method-level
  - Annotations applied to method apply to method only
  - No defaults
- Annotations
  - @RolesAllowed
    - Value is a list of security role names
  - @PermitAll
  - @DenyAll (applicable at method-level only)

Example

```
// EJB 3.0: Security View

@RolesAllowed(HR_Manager)
@Stateless public class PayrollBean implements Payroll {

    public void setSalary(int empId, double salary) {
        ...
    }

    @RolesAllowed({HR_Manager, HR_Admin})
    public int getSalary(int empId)
    {
        ...
    }
}
```

Event Notification

Bean Lifecycle Events

- EJB 2.1 specification required EnterpriseBean interfaces
- EJB 3.0 specification: only specify events you need
- Annotations:
  - @PostConstruct
  - @PreDestroy
  - @PostActivate
  - @PrePassivate
- Annotations applied to method of bean class or method of interceptor class
- Same method can serve for multiple events

Example

```
// EJB 3.0: Event Notification

@Stateful public class TravelBookingBean
    implements TravelBooking {

    @PostConstruct
    @PostActivate
    private void connectToBookingSystem() {...}

    @PreDestroy
    @PrePassivate
    private void disconnectFromBookingSystem() {...}

    ...

}
```

Interceptors

- Ease-of-use facility for more advanced cases
- Container interposes on all business method invocations
- Interceptors interpose after container
- Invocation model: “around” methods
  - Wrappered around business method invocations
  - Control invocation of next method (interceptor or business method)
  - Can manipulate arguments and results
  - Context data can be maintained by interceptor chain

Interceptors

- Default Interceptors
  - Apply to all business methods of components in ejb-jar
  - Specified in deployment descriptor
    - Due to lack of application-level metadata annotations
- Class-level interceptors
  - Apply to business methods of bean class
- Method-level interceptors
  - Apply to specific business method
- Very flexible customization
  - Ability to exclude interceptors, reorder interceptors for class or method

Exceptions

System Exceptions

- In EJB 2.1 specification
  - Remote system exceptions were checked exceptions
    - Subtypes of java.rmi.RemoteException
  - Local system exceptions were unchecked exceptions
    - Subtypes of EJBException
- In EJB 3.0, system exceptions are unchecked
  - Extend EJBException
  - Same set of exceptions independent of whether interface is local or remote
    - ConcurrentAccessException; NoSuchEJBException; EJBTransactionRequiredException; EJBTransactionRolledbackException; EJBAccessException

Exceptions

Application Exceptions

- Business logic exceptions
- Can be checked or unchecked
- Annotation: @ApplicationException
  - Applied to exception class (for unchecked exceptions)
  - Can specify whether container should mark transaction for rollback
    - Use rollback element
      - @ApplicationException(rollback=true)
    - Defaults to false

Deployment Descriptors

- Available as alternative to annotations
  - Some developers prefer them
- Needed for application-level metadata
  - Default interceptors
- Can be used to override (some) annotations
- Useful for deferred configuration
  - Security attributes
- Useful for multiple configurations
  - Java Persistence API O/R mapping
- Can be sparse, full, and/or metadata-complete

Agenda

EJB 3.0 approach

Simplified Bean classes

Client view

Using container services

Current status and summary

Current Status

- EJB 3.0 Specification Final Release last week
  - EJB Simplified API
  - EJB Core Contracts
  - Java Persistence API
- //jcp.org/en/jsr/detail?id=220
- Reference implementation under Project GlassFish as part of Java EE 5 platform release

Summary

- Major simplification of EJB technology for developers
  - Beans are plain Java classes with plain Java interfaces
  - APIs refocused on ease of use for developer
  - Easy access to container services and environment
  - Deployment descriptors available, but generally unneeded
- EJB 3.0 components interoperate with existing components/applications
- Gives developer powerful *and* easy-to-use functionality

For More Information

- Technical Sessions
  - TS-1365 Extending EJB 3.0 With Interceptors: Wed. @ 1:30pm
  - TS-3395 Java Persistence API: Wed. @ 2:45pm
  - TS-3616 Building EJB 3.0 Applications...: Thurs. @ 9:45am
  - TS-1887 Java Persistence in the Web Tier: Fri. @ 10:45am
  - TS-9056 Java Persistence in 60 Minutes: Fri. @ 2:30pm
- URLs
  - //jcp.org/en/jsr/detail?id=220
  - //java.sun.com/products/ejb/

Java EE 5

Focus is on Ease of Development

Major Revamp of Programming Model

EJB 3.0 specification support for POJOs means less to learn, less to code and less to maintain

New Java Persistence API makes object / relational mapping cleaner and easier

New and updated Web Services (JAX-WS 2.0 and JAXB 2.0) simplifies SOA implementation

JavaServer™ Faces 1.2 technology facilitates building Web 2.0 Applications with AJAX

Annotations eliminate the need for deployment descriptors, in most cases

Supported by NetBeans™ Enterprise Developer Pack 5.5 Preview today

Get the SDK:
http://java.sun.com/javaee/downloads/

Q&A

Enterprise JavaBeans™ 3.0

Linda DeMichiel

Sun Microsystems, Inc.

TS-3396