



Enterprise JavaBeans 3.0

Linda DeMichiel

EJB 3.0 Specification Lead

Senior Staff Engineer

Sun Microsystems



Agenda



- Goals of EJB 3.0
- Simplified EJB 3.0 API
- Persistence API
- Summary and Current Status

2

EJB 3.0 Focus: Ease of Development



- EJB 2.1
 - Very powerful, but complex to use
- EJB 3.0: Simplify EJB for developers
 - Simplified set of APIs
 - Reduce the number of classes and interfaces the developer needs to implement
 - Eliminate deployment descriptor from developer's view
 - Increase developer productivity
 - Attract broader range of developers to J2EE / EJB

3

EJB 3.0: Compatibility Goals



- Retain / improve power of EJB 2.1
- EJB 2.1 APIs are always available
- Provide interoperability between new EJB 3.0 APIs and existing EJB 2.1 APIs
 - For reuse of existing components in new applications
 - New applications can be clients of older beans
 - To allow components to be updated without affecting existing clients
 - Older clients can be clients of newer beans, without change

4

EJB 3.0 Approach to Simplification



- Java language metadata
- Simplification of all bean types
- Simplification of environment access
- Lighter-weight entity beans
 - Support for object/relational mapping
 - Expansion of EJB QL
- More work done by container; less by developer

5

Java Language Metadata Annotations



- Part of J2SE™ 5
- Used for many simplifications in EJB 3.0
 - Reduce the number of interfaces and classes the programmer needs to implement
 - Specify points for injection of container behavior and services
 - Eliminate need for separate deployment descriptor
 - Specify object/relational mapping
- Gives developer simple-to-use and yet powerful capability

6

XML Deployment Descriptors



- Preserve ability to use XML deployment descriptors for developers who prefer them
 - As alternative mechanism
 - For overriding of annotations
- Nearly all new features introduced with metadata will have equivalents in XML

7

Simplification of EJB Bean Types



- Elimination of requirements for EJB component interfaces
- Elimination of requirements for Home interfaces
- Elimination of requirements for javax.ejb.EnterpriseBean interfaces
- Elimination of required use of JNDI API for look-ups
- Elimination of all required interfaces for Entity Beans

8

EJB 2.1 Stateless Session Bean



```
public interface Calculator extends EJBObject {
    public int add (int a, int b)
        throws RemoteException;
    public int subtract (int a, int b)
        throws RemoteException;
}

public interface CalculatorHome extends EJBHome
{
    public Calculator create()
        throws CreateException, RemoteException;
}
```

9

Stateless Session Bean Class



```
public class CalculatorBean implements SessionBean {
    private SessionContext ctx;
    public void setSessionContext(SessionContext ctx){
        this.ctx = ctx;}
    public void ejbCreate () {}
    public void ejbActivate () {}
    public void ejbPassivate () {}
    public void ejbRemove() {}
    public int add (int a, int b) {
        return a + b;
    }
    public int subtract ( int a, int b) {
        return a - b;
    }
}
```

10

Deployment Descriptor



```
<session>
<ejb-name>CalculatorEJB</ejb-name>
<home>com.example.CalculatorHome</home>
<remote>com.example.Calculator</remote>
<ejb-class>com.example.CalculatorBean</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>
</session>
...
...
```

11

Same Example: EJB 3.0



```
@Stateless public class CalculatorBean implements
    Calculator {
    public int add (int a, int b) {
        return a + b;
    }

    public int subtract ( int a, int b) {
        return a - b;
    }
}

@Remote public interface Calculator {
    public int add (int a, int b);
    public int subtract (int a, int b);
}
```

12

Deployment Descriptor



13

Simplifications Here



- Elimination of Home Interface requirement
- Business interface is a plain Java interface
 - Bean can implement it or it can be generated
 - Can support remote access
 - EJB(Local)Object removed from client view
 - RemoteExceptions removed from client view
 - Bean can have more than one business interface
- No implementation of SessionBean interface

14

Metadata and Defaulting



- Defaulting of transaction management types
 - Container-managed transaction
- Defaulting of transaction attributes
 - REQUIRED
- Default use of security properties
 - unchecked methods
 - use of caller identity
- Defaulting of O/R mapping (for Entities)
- Etc...

Metadata annotations are available to further specify / customize all of these

15

Simplified Environment Access



- Specify dependencies in metadata
- Container supplies bean with what it needs
- Resources, beans, etc. are “injected”
- Simple lookup method added to EJBContext
- These mechanisms allow developer to avoid use of complex JNDI APIs

16

Instance Variable Injection



```
@Stateless public class MySessionBean {  
  
    @Resource public DataSource customerDB;  
  
    public void myMethod (String myString) {  
        try  
        {  
            Connection conn = customerDB.getConnection();  
            ...  
        }  
        catch (Exception ex)  
        { ... }  
    }  
}
```

17

Simple Dynamic Lookup



- lookup() method added to EJBContext
- EJBContext is injected

```
@Resource SessionContext ctx;  
...  
myShoppingCart = (ShoppingCart)ctx.lookup  
("shoppingCart");  
...
```

18

Entity Beans: Goals



- Simplify programming model
 - Like other EJB 3.0 beans
- Support for light-weight domain modeling
 - Support for modeling of inheritance
 - Additions to query language
 - Support for object/relational mapping specification
- Make instances usable outside the container
 - Remove need for data transfer objects ("DTOs")
 - Make easier to test
- Complete query capabilities

19

Simplifications to Entity Beans



- Concrete classes (not abstract)
- Support use of new()
- No required bean interfaces
- getter/setter "property" methods
 - Can contain logic, e.g., for validation, transformation, etc.
 - Can access persistent instance variables in bean class
- Collection interfaces for relationships
- No required callback interfaces
- Usable outside the container; serializable

20

EntityManager API



- EntityManager serves as untyped "home" for entity operations
- Similar in functionality to Hibernate Session, JDO PersistenceManager, etc.
- Methods for life cycle operations for entity beans
 - persist, remove, merge, flush, refresh, etc
- Factory for Query objects

21

Entity Lifecycle Operations



- new()
 - Instance is not yet managed
- persist()
 - Instance becomes managed by EntityManager
 - Instance becomes persistent when transaction commits
- remove()
 - Persistent instance is deleted when transaction commits
- merge()
 - Merge detached instance state back into persistence context
 - Detached instances are instances that exist outside of original persistence context

22

Entity Bean Example



```
@Entity public class Customer {
    private Long id;
    private String name;
    private Address address;
    private HashSet orders = new HashSet();

    // automatically generated primary key
    @Id(generate=SEQUENCE)
    public Long getID() {
        return id;
    }

    protected void setID(Long id) {
        this.id = id;
    }
}
```

23

Example (cont.)



```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

@OneToMany(cascade=ALL, mappedBy="customer")
public Set<Order> getOrders() {
    return orders;
}

public void setOrders(Set<Order> orders) {
    this.orders = orders;
}
}
```

24

Client View



```
@Stateless public class OrderEntryBean {  
  
    @Resource private EntityManager em;  
  
    public void addOrder(int cID, Order newOrder) {  
        Customer c = em.find(Customer.class, cID);  
        c.getOrders.add(newOrder);  
        newOrder.setCustomer(c);  
    }  
  
    // other business methods  
}
```

25

Object/Relational Mapping



- Provides an automated facility for developer working with a Java object model mapped to a relational database
 - Ease of development facility for developer working with relational database
- Developer is aware of mapping between database schema and domain object model
 - Programmer is in control of mapping
 - Programmer can depend on mapping (e.g., programmer can write SQL queries).

26

Example: Mapping Relationships



```
@Entity public class Employee {  
    private Address address;  
  
    @ManyToOne  
    public Address getAddress () {  
        return address;  
    }  
  
    public void setAddress(Address address) {  
        this.address = address;  
    }  
}  
  
@Entity public class Address{  
    ...  
}
```

27

Mapping Relationships: Use of Defaulting



- Entity Employee is mapped to table EMPLOYEE
- Entity Address is mapped to table ADDRESS
- Table EMPLOYEE contains foreign key to table ADDRESS
- Foreign key type is same as primary key of ADDRESS
- Foreign key name is automatically derived from primary key of ADDRESS

Can use metadata to override defaults and/or configure for preexisting database

28

Mapping Classes to Tables



- Use Java metadata to specify mapping
- Support for usual inheritance mapping strategies
 - Table per class
 - Table per class hierarchy
 - Joined subclass
- Default type mappings are (of course) available

29

Inheritance Mapping



```
// Developer could specify it this way,  
// but doesn't need to  
  
@Entity  
@Table(name="CUST")  
@Inheritance(strategy=SINGLE_TABLE,  
              discriminatorType=STRING,  
              discriminatorValue="CUST")  
public class Customer{...}  
  
@Entity  
@Inheritance(discriminatorValue="PCUST")  
public class PreferredCustomer extends  
Customer{...}
```

30

Same Example, Using Defaults



```
@Entity
public class Customer {...}

@Entity
public class PreferredCustomer extends
Customer{...}
```

31

EJB QL Enhancements



- Projection list in SELECT clause
- Explicit joins (both inner joins and outer joins)
- Group by, Having
- Subqueries (correlated and not)
- Bulk update and delete operations
- Additional SQL functions
 - UPPER, LOWER, TRIM
- Dynamic queries
- Named parameters

32

Examples: Subqueries



```
SELECT preferredCustomer
FROM Customer preferredCustomer
WHERE preferredCustomer.balance < (
    SELECT avg(c.balance) from Customer c)

SELECT DISTINCT emp
FROM Employee emp
WHERE EXISTS (
    SELECT spouseEmp
    FROM Employee spouseEmp
    WHERE spouseEmp = emp.spouse)
```

33

Examples: Joins



```
SELECT i.category
FROM Item i JOIN i.bids b
WHERE b.amount > :amount

SELECT i
FROM Item i LEFT JOIN FETCH i.bids
WHERE i.category = 'paintings'
```

34

Examples: Projection



```
SELECT c.id, c.status
FROM Customer c JOIN c.orders o
WHERE o.count > 100

SELECT new CustomerDetails(c.id, c.status, o.count)
FROM Customer c JOIN c.orders o
WHERE o.count > :ordercount
```

35

Examples: Bulk Update/Delete



```
DELETE
FROM Customer c
WHERE c.status = 'inactive'

UPDATE Customer c
SET c.status = 'outstanding'
WHERE c.balance < 10000
```

36

SQL Queries



- Allow direct SQL over actual database schema
 - Very useful for some applications
 - Database portability not important for some applications
- Allow SQL query results to be mapped into entity beans and/or instances of other Java classes
 - Metadata can be used to specify the mapping

37

Expansion in Scope beyond EJB



- EJB 3.0 persistence model, O/R mapping facility will be usable in non-J2EE environments
 - We will add APIs for acquiring and configuring EntityManager and managing transactions outside J2EE
- Persistence API published in separate specification document
 - Will have separate Reference Implementation, CTS
- EJB 3.0 Expert Group expanded to include leading JDO vendors + individual members
- Very good merger of expertise:

38

Current Status



- Early Draft 2 released last month
 - Two documents:
 - EJB Simplified API
 - Persistence API
- Additions:
 - Callbacks
 - Interceptors
 - EntityManager improvements
 - Improvements to O/R mapping + XML proposal
 - SQL queries
- Public Draft planned for at end of June
 - Will include persistence APIs for use outside J2EE
 - Will include full specification document for all of EJB

39

Summary



- Simplifications of all enterprise beans
- Simplification of persistence layer
 - Clear O/R mapping orientation
 - Improvement of query language capabilities
- Metadata is major enabling technology
- Major collaborative effort by Expert Group

40



Linda DeMichiel
linda.demichiel@sun.com

java.sun.com/j2ee

