EJB 3.0

BOF

Overall Agenda

- Goals of EJB 3.0
- JDK 5.0 Annotations
- Session Beans
- Entity Beans
- Interceptors
- Callback Listeners

Goals of EJB 3.0

- EJB 2.1 is too "noisy"
  - ✓ Too many interfaces to implement
  - ✓ "XML Hell" too many complex deployment descriptors
  - ✓ API is too verbose
  - ✓ API too complicated in general
- Simplify the EJB programming model
- Focus on ease of use
- Facilitate Test Driven Development
- Make it simpler for average developer
- Increase developer base

JDK 5.0 Annotations

- XDoclet like metadata, C#-like metatags
  - ✓ Typesafe, compiler-checked
- Syntax you can add to the Java language
- Metadata you can attach to class, method, field, constructor, or parameter
  - ✓ Metadata is compiled into class file
  - ✓ Available at compile time and/or runtime
- EJB 3.0 makes extensive use of annotations to replace XML

JDK 5.0 Annotations

```
public @interface MyMetadata {
        String value() default "hello";
}
-----------------------------------------------------
import org.acme.MyMetadata;

@MyMetadata("stuff") public class MyClass {
        @MyMetadata public void someMethod() {...}
}
```


Session Beans

EJB 3.0: Goals

- Remove unnecessary interfaces
- Remove unnecessary callbacks
- Make deployment descriptors optional
- Make beans pojo-like
- Use default values where they make sense

7

Required Interfaces

- Homeless
- Methods don't throw RemoteException
- No verbose interface implementations

```
@Remote public interface Calculator  {
        public int add(int x, int y);
        public int subtract(int x, int y);
}
@Stateless public class CalculatorBean implements Calculator {
        public int add(int x, int y) {
                return x + y;
        }
        public int subtract(int x, int y) {
                Return x – y;
        }
}
```

8

Stateful Beans

- Still homeless
  - ✓ Created as they are looked up
- @Remove replaces EJBObject.remove
- Stateful bean is removed after method called

```
@Remote public interface ShoppingCart  {
        public void addItem(int prodId, int quantity);
        public void checkout();
}

@Stateful public class ShoppingCartBean implements ShoppingCart {

        @Remove
        public void checkout() {
          ...
        }
}
```

9

MDBs

- Just implements MessageListener
- XML turns to annotations

```
@MessageDriven(
    activation={ActivationSpecAttribute(name="destinationType",
value="javax.jms.queue")})
public class EmailBean implements MessageListener {

        void onMessage(Message msg) { }
        public int add(int x, int y) {
                return x + y;
        }

}
```

10

Transactions and Security

```
@Stateful public class ShoppingCartBean implements ShoppingCart {

        @Remove
        @TransactionAttribute(REQUIRED)
        @MethodPermission({"valid_customer"})
        public void checkout() {
          ...
        }
}
```

11

Dependency Injection

- Bean class specifies dependencies instead of lookup
- Facilitates Test Driven Development

```
@Stateful public class ShoppingCartBean implements ShoppingCart {

        @Inject private SessionContext ctx;

        @EJB(name="CreditProcessorEJB")
        private CreditCardProcessor processor;

        private DataSource jdbc;

        @Resource(jndiName="java:/DefaultDS")
        public void setDataSource(DataSource db) { this.jdbc = db; }

}
```

12

EJB 3.0: Callbacks on demand

- Callback methods still available
- Annotate on an as-needed basis

```
@Stateless public class FacadeBean implements Facade {
        @EjbTimeout void licenseExpired(Timer t) {...} // committee is leaning towards this

        @PostCreate public void initialize() {} // currently in public draft
}
```

EJB 3 Deployment Descriptors

- Believe it or not, people like XML deployment descriptors
- Externalize configuration
- Externalize system architecture

- Although not in draft, XML DDs are optional
- Replace annotations with XML and you get pure POJOs

EJB 3 Deployment Descriptors

```
@Remote public interface ShoppingCart {
        public void addItem(int prodId, int quantity);
        public void checkout();
}
@Stateful public class ShoppingCartBean implements ShoppingCart {
        @Inject private EntityManager manager;
        ...

        @Remove
        @TransactionAttribute(REQUIRED)
        @MethodPermission({"valid_customer"})
        public void checkout() {
          ...
        }
```

EJB 3 Deployment Descriptors

```
public interface ShoppingCart {
        public void addItem(int prodId, int quantity);
        public void checkout();
}


public class ShoppingCartBean implements ShoppingCart {
        private EntityManager manager;
        ...

        public void checkout() {
          ...
        }
}
```

Entity Beans

POJO based persistence

Goals of Entity Beans

- Same goals as session beans
  - ✓ Fewer interfaces, optional XML DDs, etc.
- No required interfaces or subclassing
- Plain Java based
  - ✓ Allow new()
- Provide full Object/Relational mapping
- Supports Inheritance
- Expanded EJBQL
  - ✓ Fully featured
  - ✓ Parallel SQL
  - ✓ Polymorphic Queries

Defining Entity Beans

Full Object/Relational Database Mapping

EJB 3.0 Entity Beans

- O/R Mapping Metadata as annotations
  - ✓ Table mappings, `@Table, @SecondaryTable`
  - ✓ Column mappings, `@Column, @JoinColumn`
  - ✓ Relationships, `@ManyToOne, @OneToOne, @OneToMany, @ManyToMany`
  - ✓ Multi-Table mappings, `@SecondaryTable`
  - ✓ Embedded objects, `@Dependent`
  - ✓ Inheritance, `@Inheritance, @DiscriminatorColumn`
  - ✓ Identifier + Version properties, `@Id, @Version`

20

Entity Annotations

```
@Entity
@Table(name="AUCTION_ITEM")
public class Item {
   private long id;
   private String description;
   private String productName;
   private Set<Bid> bids = new HashSet();
   private User seller;

  @Id(generate=GeneratorType.AUTO)
  @Column(name="ITEM_ID")
   public long getId() {
      return id;
   }
   public void setId(long id) {
      this.id = id;
   }

....
```

relational table declaration

auto-key generation

21

Entity Annotations

```
@Entity
@Table(name="AUCTION_ITEM")
public class Item {
   private long id;
   private String description;
   private String productName;
   private Set<Bid> bids = new HashSet();
   private User seller;

  @Id(generate=GeneratorType.AUTO)
  @Column(name="ITEM_ID")
   public long getId() {
      return id;
   }
   public void setId(long id) {
      this.id = id;
   }

....
```

```
create table AUCTION_ITEM
(
ITEM_ID Number,
DESC varchar(255),
ProductName varchar(255),
USER_ID Number
);
```

22

Entity Annotations

```
@Entity
@Table(name="AUCTION_ITEM")
public class Item {
   private long id;
   private String description;
   private String productName;
   private Set<Bid> bids = new HashSet();
   private Owner owner;

  @Id(generate=GeneratorType.AUTO)
  @Column(name="ITEM_ID")
   public long getId() {
      return id;
   }
   public void setId(long id) {
      this.id = id;
   }

....
```

```
create table AUCTION_ITEM
(
ITEM_ID Number,
DESC varchar(500),
ProductName varchar(255),
OWNER_ID Number
);
```

23

Entity Annotations

```
...

@Column(name="DESC", length=500)
 public String getDescription() {
    return description;
 }
 public void setDescription(String desc) {
    this.description = desc;
 }


 public String getProductName() {
    return productName;
 }
 protected void setProductName(String name) {
    this.productName  = name;
 }

 ...
```

column mapping

intuitive defaults

24

Entity Annotations

```
...
  @Column(name="DESC", nullable=false,
length=500)
   public String getDescription() {
      return description;
   }
   public void setDescription(String desc) {
      this.description = desc;
   }


   public String getProductName() {
      return productName;
   }
   protected void setProductName(String name) {
      this.productName  = name;
   }
...
```

```
create table AUCTION_ITEM
(
ITEM_ID Number,
DESC varchar(500),
ProductName varchar(255),
OWNER_ID Number
);
```

25

Relationships

- Relationships,
  - ✓ @ManyToOne, @OneToOne, @OneToMany, @ManyToMany
  - ✓ Supports lazy and eager loading of relationships
  - ✓ Cascades: delete, create, and merge
  - ✓ No CMR: You must manage the relationships somewhat.

26

Entity Relationships

```
@OneToOne(fetch=LAZY)
@Column(name="OWNER_ID")
public Owner getOwner() {
   return owner;
}

protected void setOwner(Owner owner) {
   this.owner = owner;
}

@OneToMany(cascade=ALL)
@JoinColumn(name="ITEM_ID")
protected Set<Bid> getBids() {
   return bids;
}
protected void setBids(Set<Bid> bids) {
   this.bids = bids;
}
```

lazy/eager loading

various cascade types

27

Entity Relationships

```
@OneToOne(fetch=LAZY)
@Column(name="OWNER_ID")
public Owner getOwner() {
   return owner;
}

protected void setOwner(Owner user) {
   this.owner = owner;
}

@OneToMany(cascade=ALL)
@JoinColumn(name="ITEM_ID")
protected Set<Bid> getBids() {
   return bids;
}
protected void setBids(Set<Bid> bids) {
   this.bids = bids;
}
```

```
create table AUCTION_ITEM
(
ITEM_ID Number,
DESC varchar(255),
ProductName varchar(255),
OWNER_ID Number
);

create table BID
(
...
ITEM_ID Number
...
);
```

28

Multi-Table

- Multi-Table Mappings,
  - ✓ Entity can be stored in one or more tables
  - ✓ @SecondaryTables, @SecondaryTable

29

Multi-table Mappiings

```
@Entity
@Table(name="OWNER")
@SecondaryTable(name="ADDRESS"

join={@JoinColumn(name="ADDR_ID")})
public class Owner {
   private long id;
   private String name;
   private String street;
   private String city;
   private String state;

   @Id(generate=GeneratorType.AUTO)
   @Column(name="OWNER_ID")
   public long getId() {
      return id;
   }
   public void setId(long id) {
      this.id = id;
   }
```

```
create table OWNER
(
OWNER_ID Number,
NAME varchar(255),
);

create table ADDRESS
(
ADDR_ID Number,
STREET varchar(255),
CITY varchar(255),
STATE varchar(255)
);
```

30

Multi-Table Mappings

```
...
  @Column(name="STREET",
secondaryTable="ADDRESS")
  public String getStreet() {
    return street;
  }
  public void setStreet(String street) {
    this.street = street;
  }

  @Column(name="CITY",
secondaryTable="ADDRESS")
  public String getCity() {
    return city;
  }
  protected void setCity(String city) {
    this.city  = city;
  }
...
```

```
create table OWNER
(
OWNER_ID Number,
NAME varchar(255),
);

create table ADDRESS
(
ADDR_ID Number,
STREET varchar(255),
CITY varchar(255),
STATE varchar(255)
);
```


Embedded Objects

- Embedded Objects
  - ✓ Aggregated objects whose properties can be mapped
  - ✓ @Dependent, @DependentObject

Embedded Objects

```
@Entity
@Table(name="OWNER")
public class Owner {
  private long id;
  private String name;
  private Address address;

  @Id(generate=GeneratorType.AUTO)
  @Column(name="CUST_ID")
  public long getId() {
    return id;
  }
  public void setId(long id) {
    this.id = id;
  }

  @Column(name="NAME")
  public String getName() {
....
```

```
@DependentObject
Public class Address {
  private String street;
  private String city;
  private String state;

  public String getStreet() {
    return street;
  }

  public void setStreet(String street) {
    this.street = street;
  }

  public String getCity() {
    return city;
  }
...
```


Entity Annotations

```
...
@Dependent({
   @DependentAttribute(name="street", @Column(name="STREET")),
   @DependentAttribute(name="city", @Column(name="CITY")),
   @DependentAttribute(name="state", @Column(name="STATE"))})
public Address getAddress() {
   return address;
}
public void setAddress(Address address) {
   this.address = address;
}

...
```


Interacting With Entity Bean

Plain Java Objects

Entity Manager

- Entities created as any plain Java object
  - ✓ Customer cust = new Customer();
- Plain Java objects and homeless
- Can be detached and reattached to container
  - ✓ Can be serialized to remote client
  - ✓ Remote client can perform updates on local copy
  - ✓ Copy can be sent back to server and merged back in
- Persisted by the EntityManager service
  - ✓ All access through this service
  - ✓ Creation, retrieval, removal, and merging
  - ✓ Analogous to Hibernate Session

Create the objects

- Create the entities like you would any other object
- Allocate entire object graph like any other Java code

```
Item item = new Item();
item.setDescription("O'reilly's EJB 4th Edition");
item.setProductName("EJB 2.1 Book");
...
Owner bill = new Owner();
bill.setName("Bill");
item.setOwner(bill);
Bid bid = new Bid();
...
HashSet<Bid> bids = new HashSet();
bids.add(bid);
item.setBids(bids);
```

Entity Manager

- All entities persisted by the EntityManager service
  - ✓ All access through this service
  - ✓ Creation, retrieval, removal, and merging
  - ✓ Analogous to Hibernate Session
- Injected with dependency injection

EntityManager

```
@Stateless public class ItemDAOImpl implements ItemDAORemote {

  @Inject private EntityManager em;

  public long create(Item item) {
     em.create(item);
     return item.getId();
  }

  public Item findById(long id) {
     return (Item) em.find(Item.class, id);
  }

  public void merge(Item item) {
     em.merge(item);
  }
}
```

Inject the EntityManager service

EntityManager

```
@Stateless public class ItemDAOImpl implements ItemDAORemote {

  @Inject private EntityManager em;

  public long create(Item item) {
     em.persist(item);
     return item.getId();
  }

  public Item findById(long id) {
     return (Item) em.find(Item.class, id);
  }

  public void merge(Item item) {
     em.merge(item);
  }
}
```

- Item allocated remotely
- If cascade CREATE, entire object graph inserted into storage

EntityManager

```
@Stateless public class ItemDAOImpl implements ItemDAORemote {

  @Inject private EntityManager em;

  public long create(Item item) {
     em.persist(item);
     return item.getId();
  }

  public Item findById(long id) {
     return (Item) em.find(Item.class, id);
  }

  public void merge(Item item) {
     em.merge(item);
  }
}
```

- Item found with primary key
- Detached from persistent storage at tx completion
- Can be serialized like any other object

EntityManager

```
@Stateless public class ItemDAOImpl implements ItemDAORemote {

  @Inject private EntityManager em;

  public long create(Item item) {
     em.persist(item);
     return item.getId();
  }

  public Item findById(long id) {
     return (Item) em.find(Item.class, id);
  }

  public void merge(Item item) {
     em.merge(item);
  }
}
```

- Item can be updated remotely and merged back to persistent storage
- Item instance is reattached to storage with merge() call

Query API

- Queries may be expressed as EJBQL strings
  - ✓ Embedded in code
  - ✓ Externalized to metadata (named queries)
- Invoke via Query interface
  - ✓ Named parameter binding

```
@Session public class ItemDAOImpl {
   ...
  public List findByDescription(String description, int page) {
          return em.createQuery("from Item i where i.description like :d")
            .setParameter("d", description)
            .setMaxResults(50)
            .setFirstResult(page*50)
            .listResults();
      }

      ...
```

EJB QL 3.0

- EJBQL 3.0 is very similar to HQL (Hibernate Query Language)
- Aggregation, projection
  - ✓ `select max(b.amount) from Bid b where b.item = :id`
  - ✓ `select new Name(c.first, c.last) from Customer c`
- Fetching
  - ✓ `from Item i left join fetch i.bids`
- Subselects
  - ✓ `from Item i join i.bids bid where bid.amount = (select max(b.amount) from i.bids b)`
- Group By, Having, Joins

Inheritance

- Persistence mapping supports inheritance
  - ✓ Single table per hierarchy – SINGLE_TABLE
  - ✓ Join table per subclass – JOINED
  - ✓ Distinct table per subclass – UNION
- Queries on class hierarchy are polymorphic

Inheritance – SINGLE_TABLE

```
@Entity
@Table(name="Animal")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE")
public class Animal {
  @Id private int id;
  @Column(name="AVG_WEIGHT")
  private int averageWeight;
...
}

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public class Dog extends Animal{
  @Column(name="BREED")
  private String breed;
...
}
```

Inheritance – SINGLE_TABLE

```
create table Animal
(
ID Number,
TYPE varchar(255),
AVG_WEIGHT Number,
BREED varchar(255)
);
```

Inheritance – JOINED

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="TYPE")
@Table(name="Animal")
public class Animal{
  @Id private int id;
  @Column(name="AVG_WEIGHT")
  private int averageWeight;
...
}

@Entity
@InheritanceJoinColumn(name="DOGGY_ID")
@Table(name="Doggy")
public class Dog extends Animal{
  @Column(name="BREED")
  private String breed;
...
}
```

Inheritance – JOINED

```
create table Animal
(
ID Number,
TYPE varchar(255),
AVG_WEIGHT Number
);

create table Doggy
(
DOGGY_ID Number,
BREED varchar(255)
);
```

Inheritance – UNION

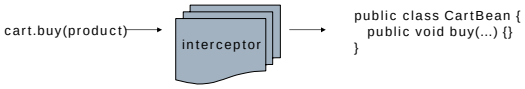
```
create table Kitty
(
ID Number,
TYPE varchar(255),
AVG_WEIGHT Number
BREED varchar(255),
LIVES Number
);

create table Doggy
(
DOGGY_ID Number,
TYPE varchar(255),
AVG_WEIGHT Number
BREED varchar(255)
);
```

New Features

EDR2 functionality

Interceptors

- Interceptors intercept calls
- Interceptors sit between caller and a session bean
- Analogous to servlet filters
- Can only be used with session and message driven beans
- Precursor to full aspect-oriented programming



Why Interceptors

- Tracing
- Pluggable auditing
- Custom security
- Generic exception handling

Interceptors

- Interceptor is a plain Java class
- A method can be designated as the interceptor method
  - ✓ @AroundInvoke
- That method must return Object and throw Throwable
- That method must also accept an InvocationContext
- InvocationContext hold information about the request
- Request can be aborted with an exception
- Exceptions can be caught from the bean class and suppressed
- Return objects can be changed
- Arguments can be modified

Interceptors

```
public class RuleBasedSecurityInterceptor {

  boolean checkRule(...) { ...}

  @AroundInvoke
  public Object customSecurity(InvocationContext ctx) throws Exception {

     if (checkRule(...) == false) {
        throw new SecurityException("Custom check failed");
     }

     return ctx.proceed();
  }
}
```

55

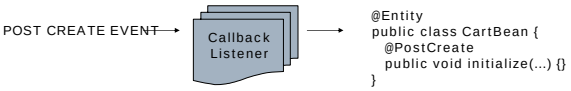
Exception handling

```
public class OracleExceptionHandlerInterceptor {

  public final static int ORACLE_DEADLOCK = ...;

  @AroundInvoke
  public Object customSecurity(InvocationContext ctx) throws Exception {

     try {
        return ctx.proceed();
     } catch (SQLException ex) {
        switch (ex.getErrorCode()) {
          case ORACLE_DEADLOCK:
             throw new DeadlockException(ex);
          ...
        }
     }
  }
}
```

56

Callback Listeners

- Similar to interceptors
- Intercept EJB callback methods in a separate class
- Can be attached to entities, sessions, or MDBs



57

Callback Listeners

```
public class CallbackTracer {

  @PostCreate
  public void tracePosCreate() {
     log.trace("postcreate");
  }

  @PreUpdate
  public void tracePreUpdate {
     log.trace("preupdate");
  }

  ...


}
```

58

JBoss Inc.

- EJB 3.0 Preview Available NOW!
  - ✓ Download at www.jboss.org
  - ✓ Tutorial with code examples
  - ✓ Mostly functional

59