EJB 3.0

Rajesh Patel
rpatel@harpoontech.com
Software Developer
Partner Harpoon Technologies

St. Louis Java SIG
November 10, 2005

Background

- Java Developer for 4 years
- 8 years in software industry.
- Mentoring Java Developers
- Hibernate Consultant For Jboss
- Never done EJB 2.1

EJB 2.1

- EJB 2.1 too complicated
- Too Many interfaces to implement
- Too Many difficult XML Descriptors

EJB 3.0 To the Rescue

- Everything is a POJO(Plain old java object)
- Simplify programming model
- Hibernate like ORM(Object Relational Mapping)
- Dependency injection
- Annotations(Requires Java 5)
- Facilitate Test Driven Development
- All Exceptions are Runtime

EJB3-JSR 220

- Available at http://java.sun.com/products/ejb/docs.html
- EJB 3.0 Simplified(50 pages)
- EJB 3.0 Persistence API(191 pages)
- EJB 2.1 Core(646 pages)
- EJB 3.0 spec is 1/3 size of EJB 2.1
  - That is if you leave out EJB 2.1

EJB Types

- Stateless Session Bean
  - Don't maintain state between calls
- Statefull Session Bean
  - Store state between calls – like a shopping cart
- Entity Bean
  - Used for beans that need to be persisted
- Message Driven Bean
  - A bean that responds to messages(ie JMS/SMTP)

Remote vs. Local

- Local interface is a direct java invocation
- Remote is a call to another VM via RMI
- Original EJB designs had everything remote
- Led to the EJB being perceived as slow
- Colocate as much as possible
  - Web app and EJB should be in same cont.

Session Beans

- Don't have to Implement any large interface
- Does not throw RemoteException
  - In fact no EJB throws RemoteException

Stateless Session Bean Example

```
@Local public interface CalculatorLocal {
        public int add(int x, int y);
        public int subtract(int x, int y);
}
@Remote public interface CalculatorRemote  {
        public int add(int x, int y);  // Look Ma no Remote Exception
        public int subtract(int x, int y);
}
@Stateless public class CalculatorBean implements CalculatorRemote,
CalculatorLocal{
        public int add(int x, int y) {
                return x + y;
        }
        public int subtract(int x, int y) {
                Return x – y;
        }
}
```

Stateful Session Bean

- Created on Lookup
- @Remove – Bean is removed after method call

```
@Local public interface Incrementer  {
        public int next();
        public void remove();
}
@Stateful public class IncrementerBean implements Incrementer {
        int count=0;
        public int next() {
                count++;
                return count;
        }
        @Remove public void remove(){}
}
```

Transactions

- @TransactionAttribute defines the transaction boundary for a session bean method.

```
@Stateful public class ShoppingCartBean implements
ShoppingCart {

    @Remove
    @TransactionAttribute(REQUIRED)
    public void checkout()  {
        …
    }
}
```

Transaction Attributes

- REQUIRED – Start new transaction if no current tx
  - Assumed if using CMT and no attribute specified
  - “Intelligent default”
- REQUIRESNEW – Always start new TX, suspend old
- SUPPORTS – Don't start new tx, but use existing
- MANDATORY – Require the caller to have started TX
- NEVER – Forbid the caller to have started TX
- NOTSUPPORTED – Suspend existing TX.

Security

- Annotations Defined by JSR-250
  - JSR-250 defines common annotations for java 5

```
@Stateful public class ShoppingCartBean implements
ShoppingCart {
    @MethodPermission({“valid_customer”})
    public void checkout()  {
        …
    }
}
```

Dependency Injection

- @EJB for ejbs
- @ Resource for resources

```
@Stateful
public class FibonacciBean implements Fibonacci {
    int x=0,y=1;
    @EJB private Calculator calculator;
    public int next() {  // Mathematicians: This alg not right...
        int next = calculator.add(x,y);
        x=y;y=next;
        return next;
    }
```

Interceptors

- Allow custom code to be applied to an EJB
- Simply add @AroundInvoke to a method
- Introduces AOP to EJB

```
@Stateless
public class CalculatorBean implements Calculator {
        public int add(int x, int y) {
                return x + y;
        }
          @AroundInvoke
          public Object calcInterceptor(InvocationContext ctx) throws Exception
          {
            System.out.println("*** Intercepting call to EmailMDB.mdbInterceptor()");
            return ctx.proceed();
          }
}
```

Interceptors

- A custom interceptor class can be defined

```
@Stateless
@Interceptors ({"com.my.TracingInterceptor"})
public class EmailSystemBean
{
   ...
}
```

Interceptor Class

```
public class TracingInterceptor {

        @AroundInvoke
        public Object log(InvocationContext ctx) throws Exception
        {
          System.out.println("*** TracingInterceptor intercepting");
          long start = System.currentTimeMillis();
          try
          {
            return ctx.proceed();
          }
          catch(Exception e)
          {
            throw e;
          }
          finally
          {
            long time = System.currentTimeMillis() - start;
            String method = ctx.getBean().getClass().getName() +
            "." + ctx.getMethod().getName() + "()";
            System.out.println("*** TracingInterceptor invocation of "
                    + method + " took " + time + "ms");
          }
        }
      }
```

Entity Beans

- No interfaces to implement everything is POJO
  - Notice the trend :)
- All types of relationships supported: one-to-many, many-to-one, one-to-one, many-to-many
- Inheritance
- Polymorphic Queries
- Based on Hibernate

Entity Beans

- Only 2 annotations needed to get started
- @Entity and @Id

Entity Bean Example

```
@Entity
public class Product { // table name defaults to product
    private int id;
    String product;
    @Id(generate = GeneratorType.AUTO)
    public int getId() {return id;}  // column defaults to id
    public void setId(int id) {this.id = id;}

    public String getProduct() { // columnt default to product
        return product;
    }
    public void setProduct(String product) {
        this.product = product;
    }
}
```

Entity Bean Example

```
@Entity
@Table(name="vendor_product")
public class Product {
        private int id;
        String product;
        @Id(generate = GeneratorType.AUTO)
        @Column(name="product_id")
        public int getId() {return id;}
        public void setId(int id) {this.id = id;}
        // column defaults to product
        public String getProduct() {
                return product;
        }
        public void setProduct(String product) {
                this.product = product;
        }
```

Entity Lifecyle

- Transient – Newly created and not saved
- Persistent – Managed by hibernate (loaded/saved)
- Detached
  - Serialized out of VM
  - Persistence Context Ends

Entity Lifecycle

```
EntityManager em = (EntityManager)
getInitialContext().lookup("java:/EntityManagers/custdb");

// Bean Managed Transaction
TransactionManager tm = (TransactionManager)
getInitialContext().lookup("java:/TransactionManager");
tm.begin();
Product product = new Product();  // product is transient
em.persist(product);                    // product is persisted
tm.commit();                            // product is detached

tm.begin();
em.merge(product);              // Detached product becomes
persisted
em.remove(product);             // Persisted product becomes transient
tm.commit();
```

Persistence Context

- Container Managed Transaction Scoped
- Container Managed Extended
- Application Managed Transaction Scoped
- Application Managed Extended

```
@PersistenceContext(type=EXTENDED)
EntityManager em;
```

Relationships

- @ManyToOne
- @OneToMany
- @OneToOne
- @ManyToMany

Many To One

```
  @Entity
public class Customer implements Serializable
{
...
  private Collection<Order> orders = new HashSet();
...

        @OneToMany(cascade = CascadeType.PERSIST)
        public Collection<Order> getOrders() {
                return orders;
        }

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

Cascading

- Persist - **CascadeType.PERSIST**
- Remove - **CascadeType.DELETE**
- Merge - **CascadeType.MERGE**

Customer

```
Customer cust = new Customer();
cust.setName("Bill2");

Order order = new Order();
order.setName("Cold Coffee");

Order order2 = new Order();
order2.setName("Hot Coffee");

cust.getOrders().add(order);
cust.getOrders().add(order2);

em.persist(cust);  // Orders are automatically saved via a cascade
```

EJBQL

- Very similar to HQL
- HQL very similar to SQL

```
// a single object – notice the implice join to the
// shipping address table
SELECT o
FROM Order o
WHERE o.shippingAddress.state = 'CA'

// The collection order is agressively fetch
// using a Fetch Join – used for performance
// optimizations.
SELECT DISTINCT c
FROM Customer c JOIN FETCH c.orders
WHERE c.address.state = 'CA'
```

JBoss Embeddable EJB3

- Standalone EJB3 based on JBoss microkernel
- Allows easy unit test of EJBs!

```
public static void main(String str[]){
        EJB3StandaloneBootstrap.boot(null);
        EJB3StandaloneBootstrap.scanClasspath();

        InitialContext ctx = getInitialContext();
        Calculator calculator = (Calculator) ctx.lookup(Calculator.class.getName());
        calculator.add(2,2);
 }
```

Vendors

- Jboss
- BEA – Bought Solarmetric for EJB3 Perst.
- Resin
- Oracle AS

Spec Finalization

- Spec is expected not to change much from current form
- Hopefully it will be finalized by April.

Questions?