EJB 3, Spring and Hibernate
A Comparative Analysis

Spring? Hibernate?
EJB 3?
What the #$*% do I do?

Reza Rahman
Author, EJB 3 in Action
Chief Software Architect, Tripod Technologies

Philadelphia Java User Group
September 2007
The Continuing Conundrum: What are the Cool Kids Wearing Today?

• EJB 2.x, the architect’s darling in the bubble years, is now officially as un-cool as grandpa in Speedos (and rightfully so!):
  • Made you write a bunch of redundant artifacts.
  • Gator wrestling with deployment descriptors.
  • Entity beans are half-baked and non-portable.

• Enter Spring and Hibernate, the cool new kids on the block:
  • POJO programming.
  • Give you 90+% of what EJB does.
  • And then some.

• EJB 3 to the rescue? Does granddad know something the cool kids don’t after all?
EJB 3: EJB Reinvented

- EJB 3 is truly a different beast from EJB 2.x. It is an overhaul of the magnitude its pioneers would not have foreseen:
  - EJB 3 embraces POJO programming through annotations.
  - The verbose XML deployment descriptor has been made optional.
  - The Entity Bean concept is no longer managed by the container.
  - Instead, EJB 3 adopts JPA, an API based paradigm similar to Hibernate, TopLink and JDO.
  - Object relational mapping and object queries have been completely defined instead of being left up to container vendors to sort out.
  - EJB 3 makes heavy use of “intelligent defaulting” whenever possible. This is a similar idea to “convention over configuration” in the Rails world.

- These changes make EJB 3 a viable development choice on its own right. It just may be the easiest full stack enterprise platform to work with.

- However, the question remains as to why developers should opt for EJB 3 instead of simply sticking with Spring and Hibernate.
EJB 3, Spring and Hibernate: The Bottom Line

- **Use EJB 3 if:**
  - You like annotations and dislike XML configuration.
  - You prefer a tightly integrated solution stack that makes sensible default choices for you and keeps configuration to a bare minimum.
  - Your application is very stateful.
  - You use JSF and are considering using Seam.
  - Standardization is an important consideration.

- **Use Spring if:**
  - Your application requires fine-grained control at the container level.
  - Your application requires a lot of configuration beyond gluing together components and resources.
  - You need to build your own solution stack (such as with iBATIS, Quartz or Acegi).
  - You need advanced AOP features.

- **Use Hibernate if:**
  - You like the Hibernate API better than JPA.
  - You are very unlikely to switch persistence engines.
EJB 3 and Spring

"...And no hitting below the belt!"
## EJB 3 and Spring Feature Sets

<table>
<thead>
<tr>
<th>Feature</th>
<th>EJB 3</th>
<th>Spring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dependency Injection</strong></td>
<td>Can inject anything in the container including EJBs, data sources, JMS resources, JPA resources and web services endpoints</td>
<td>Can inject almost anything including lists, maps, properties and JNDI resources</td>
</tr>
<tr>
<td><strong>Transaction management</strong></td>
<td>Works right out of the box, but only JTA is supported</td>
<td>Have to configure it to make it work, but supports a number of strategies including JTA, JDBC and Hibernate</td>
</tr>
<tr>
<td><strong>Persistence</strong></td>
<td>Tightly integrated through JPA</td>
<td>Framework support for JPA, JDO, Hibernate, JDBC, iBatis</td>
</tr>
<tr>
<td><strong>State management</strong></td>
<td>Robust support through Stateful Session Beans and Extended Persistence Context</td>
<td>Indirect support dependent on web container session management</td>
</tr>
<tr>
<td><strong>Web services</strong></td>
<td>Seamless support for JAX-WS 2.0</td>
<td>Poor direct support, best integration available is via configuring XFire for registered beans.</td>
</tr>
<tr>
<td><strong>Messaging</strong></td>
<td>Robust support out of the box through Message Driven Beans</td>
<td>Need to add configuration for each message listener. Listeners are not thread-safe.</td>
</tr>
<tr>
<td><strong>Remoting</strong></td>
<td>Integrated support through Session Bean remote interfaces. Supports distributed transactions and security.</td>
<td>Remoting support may be added via configuration. Remote transactions and security are not supported. However protocols other than RMI such as HTTP, Hessian and Burlap are supported.</td>
</tr>
<tr>
<td><strong>AOP</strong></td>
<td>Simple but limited support through interceptors.</td>
<td>Robust support through AspectJ.</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Integrated support for declarative and programmatic security through JAAS.</td>
<td>Must add and configure Acegi security. However, support beyond JAAS is possible through Acegi.</td>
</tr>
<tr>
<td><strong>Scheduling</strong></td>
<td>Simple scheduling possible through EJB Timer service.</td>
<td>Must configure Quartz for scheduling.</td>
</tr>
<tr>
<td><strong>Clustering</strong></td>
<td>Most containers have built-in support.</td>
<td>Complex third-party solutions available.</td>
</tr>
</tbody>
</table>
@Stateless
public class PlaceBidBean implements PlaceBid {
    @PersistenceContext
    private EntityManager entityManager;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void addBid(Bid bid) {
        entityManager.persist(bid);
    }
}

@Remote
@WebService
public interface PlaceBid {
    void addBid(Bid bid);
}
public class PlaceBidBean implements PlaceBid {
    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional(propagation=Propagation.REQUIRED)
    public void addBid(Bid bid) {
        sessionFactory.getCurrentSession().save(bid);
    }
}

public interface PlaceBid {
    void addBid(Bid bid);
}
A Quick Look at Code: Spring XML Configuration

```xml
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="hibernateProperties">
        <value>hibernate.dialect=org.hibernate.dialect.HSQLDialect</value>
    </property>
</bean>

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

<bean id="placeBid" class="PlaceBidBean">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>

<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
    <property name="serviceName" value="placeBid"/>
    <property name="service" ref="placeBid"/>
    <property name="serviceInterface" value="PlaceBid"/>
    <property name="registryPort" value="1199"/>
</bean>

<bean id="placeBidService" class="org.codehaus.xfire.spring.remoting.XFireExporter">
    <property name="serviceFactory" ref="xfire.serviceFactory"/>
    <property name="xfire" ref="xfire"/>
    <property name="serviceBean" ref="placeBid"/>
    <property name="serviceClass" value="PlaceBid"/>
</bean>

<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="urlMap">
        <map>
            <entry key="/PlaceBidService"><ref bean="placeBidService"/></entry>
        </map>
    </property>
</bean>
```
@Stateful
public class BidderAccountCreatorBean implements BidderAccountCreator {
    @PersistenceContext(type=PersistenceContextType.EXTENDED)
    private EntityManager entityManager;

    public void addLoginInfo(LoginInfo loginInfo) {
        entityManager.persist(loginInfo);
    }

    public void addBiographicalInfo(BiographicalInfo biographicalInfo) {
        entityManager.persist(biographicalInfo);
    }

    public void addBillingInfo(BillingInfo billingInfo) {
        entityManager.persist(billingInfo);
    }

    @Remove
    public void cancelAccountCreation() {
        entityManager.clear();
    }

    @Remove
    public void createAccount() {
        entityManager.flush();
    }
}
Spring Dependency Injection Range

```xml
<bean id="placeBid" class="PlaceBidBean">
  <property name="bidDao" ref="bidDao"/>  
  <property name="concurrencySensitivity" value="1"/>
  <property name="adminEmails">
    <props>
      <prop key="administrator">
        administrator@somecompany.org
      </prop>
      <prop key="support">
        support@somecompany.org
      </prop>
      <prop key="development">
        development@somecompany.org
      </prop>
    </props>
  </property>
  <property name="dataSource">
    <jee:jndi-lookup jndi-name="jdbc/ActionBazaarDB"/>
  </property>
</bean>
```
@Stateless
class DIExampleBean implements DIExample {
    @Resource
    private SessionContext context;

    @PersistenceContext
    private EntityManager entityManager;

    @Resource(name="jdbc/TurtleDS")
    private DataSource dataSource;

    @EJB
    private PlaceBid placeBid;

    @WebServiceRef(wsdlLocation="http://www.ripoffcreditprocessing.com/jaxws-webservice/BillingService?WSDL")
    private BillingService billingService;
}
**Spring AOP through AspectJ**

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Auditable {
    Boolean value() default true;
}

@Aspect
public class AuditAspect {
    @Before("execution(public * *(..)) && @annotation(Auditable)")
    public void audit(JoinPoint joinPoint) {
        System.out.println("Entering: " + joinPoint);
        System.out.println(" with args: " + joinPoint.getArgs());
    }
}

public class PlaceBidBean implements PlaceBid {
    private SessionFactory sessionFactory;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    @Auditable
    public void addBid(Bid bid) {
        sessionFactory.getCurrentSession().save(bid);
    }
}

<bean id="auditAspect" class="AuditAspect"/>
<bean id="placeBid" class="PlaceBidBean">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
Lightweight EJB 3 AOP through Interceptors

```java
public class AuditInterceptor {

    @AroundInvoke
    public Object audit(InvocationContext context) throws Exception {
        System.out.println("Entering: ");
        + context.getMethod().getName();
        System.out.println(" with args: ");
        + context.getParameters();
        return context.proceed();
    }
}

@Interceptors({AuditInterceptor.class})
@Stateless
public class PlaceBidBean implements PlaceBid {
    @PersistenceContext
    private EntityManager entityManager;

    public void addBid(Bid bid) {
        entityManager.persist(bid);
    }
}
```
Spring and EJB 3 Integration

• Using EJB 3 in a Spring application:
  • Very easy to do since Spring can be deployed inside a Java EE container.
  • Anything in JNDI can be injected into the Spring application context.
  • Especially easy using the jee schema: `<jee:jndi-lookup/>`, `<jee:local-slsb/>` and `<jee:remote-slsb/>` tags.
  • Spring provides excellent integrated support for JPA, very similar to Spring-Hibernate integration.
  • Spring natively supports EJB 3 features through the Pitchfork project.

• Using Spring in an EJB 3 application:
  • Insert Spring beans into EJB 3 using Interceptors. JBoss provides excellent support for doing this using the `@Spring` annotation and the JBoss Spring deployer. You can easily implement similar functionality in other Java EE containers yourself.
EJB 3 (JPA) and Hibernate
## JPA and Hibernate Feature Sets

<table>
<thead>
<tr>
<th></th>
<th>JPA</th>
<th>Hibernate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORM mapping</td>
<td>Supported through annotations and XML.</td>
<td>Supported through annotations and XML. Annotations are compatible with EJB 3 standard.</td>
</tr>
<tr>
<td>Mapping relations</td>
<td>Allows mapping of one-to-one, one-to-many, many-to-one and many-to-many relations</td>
<td>Allows mapping of one-to-one, one-to-many, many-to-one and many-to-many relations</td>
</tr>
<tr>
<td>Mapping complex types</td>
<td>Mapping complex types such as collections, composite keys, embedded classes and dates possible</td>
<td>Mapping complex types such as collections, composite keys, embedded classes and dates possible</td>
</tr>
<tr>
<td>Primary key generation</td>
<td>Various useful strategies for generating primary keys.</td>
<td>Various useful strategies for generating primary keys.</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Full support for inheritance</td>
<td>Full support for inheritance</td>
</tr>
<tr>
<td>Object queries</td>
<td>Full support of object queries through JPQL</td>
<td>Full support of object queries through HQL</td>
</tr>
<tr>
<td>Native queries</td>
<td>Robust support for native queries</td>
<td>Robust support for native queries</td>
</tr>
<tr>
<td>API syntax</td>
<td>API defined by EJB 3 standard. Slightly smaller API</td>
<td>Proprietary API. Relatively larger API</td>
</tr>
<tr>
<td>Query syntax</td>
<td>More strict and SQL like</td>
<td>Relatively more flexible, but may be confusing for an SQL developer</td>
</tr>
<tr>
<td>Query API</td>
<td>Query API supports simple primary key based lookup and JQPL</td>
<td>Query API supports primary query based lookup and HQL. In addition, Query-by-Criteria/Query-by-Example is supported</td>
</tr>
</tbody>
</table>
A Quick Look at Code: The JPA EntityManager

```java
public class ItemManager {
    private EntityManagerFactory factory;
    private EntityManager entityManager;

    public ItemManager() {
        factory = Persistence.createEntityManagerFactory("ActionBazaar");
        entityManager = factory.createEntityManager();
    }

    public void addItem(Item item) {
        EntityTransaction transaction = entityManager.getTransaction();
        entityManager.persist(item);
        transaction.commit();
    }

    public void updateItem(Item item) {
        EntityTransaction transaction = entityManager.getTransaction();
        entityManager.merge(item);
        transaction.commit();
    }

    public void deleteItem(Item item) {
        EntityTransaction transaction = entityManager.getTransaction();
        entityManager.remove(entityManager.merge(item));
        transaction.commit();
    }

    public Item getItemByName(String name) {
        Query query = entityManager.createQuery("SELECT i FROM Item i WHERE i.name = :itemName");
        query.setParameter("itemName", name);
        return (Item)query.getSingleResult();
    }

    public List getItems() {
        return entityManager.createQuery("SELECT i FROM Item i").getResultList();
    }

    public void close() {
        entityManager.close();
        factory.close();
    }
}
```
public class ItemManager {
    private SessionFactory factory;
    private Session session;

    public ItemManager() {
        factory = new Configuration().configure().buildSessionFactory();
        session = factory.getCurrentSession();
    }

    public void addItem(Item item) {
        Transaction transaction = session.beginTransaction();
        session.save(item);
        transaction().commit();
    }

    public void updateItem(Item item) {
        Transaction transaction = session.beginTransaction();
        session.update(item);
        transaction.commit();
    }

    public void deleteItem(Item item) {
        Transaction transaction = session.beginTransaction();
        session.delete(item);
        transaction.commit();
    }

    public Item getItemByName(String name) {
        Item item;
        Transaction transaction = session.beginTransaction();
        Query query = session.createQuery("from Item i where i.name = :itemName");
        query.setParameter("itemName", name);
        item = (Item) query.uniqueResult();
        transaction.commit();
        return item;
    }

    public List getItems() {
        List items;
        Transaction transaction = session.beginTransaction();
        items = session.createQuery("from Item").list();
        transaction.commit();
        return items;
    }

    public void close() {
        session.close();
        factory.close();
    }
}
Annotated Entity Example (Hibernate or EJB 3)

```java
@Entity
@Table(name="ITEMS")
public class Item implements Serializable {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @Column(name="ITEM_ID", nullable=false)
public Long itemId;

  @Column(name="ITEM_NAME")
public String title;

  @Column(name="BID_END_DATE")
public Timestamp bidEndDate;

  @Column(name="BID_START_DATE")
public Timestamp bidStartDate;

  @Column(name="INITIAL_PRICE")
public Double initialPrice;

  @OneToMany(mappedBy="item", cascade=CascadeType.ALL)
public Set<Bid> bids;
}
```
Query-by-Criteria/Query-by-Example

List items = session.createCriteria(Item.class)
    .add(Restrictions.like("name", "%Useless Junk\%"))
    .add(Restrictions.between("initialPrice", cheapest, cheapo))
    .list();

Item junk = new Item();
junk.setBidStartDate(tomorrow);
junk.setBidEndDate(dayAfterTomorrow);
List results = session.createCriteria(Item.class)
    .add(Example.create(junk))
    .list();
The Matrix: EJB3, Spring and Hibernate

- Robust AOP
- Heavy configuration
- Fine-grained control
- Stateful application
- Minimize configuration
- Standardization
- Annotations

EJB 3 is the true path...right!?
References


• **Make the Right Decision with Our Side-by-Side Comparison of Spring and EJB 3.0**, Rod Coffin, [http://www.devx.com/Java/Article/32314/0/page/1](http://www.devx.com/Java/Article/32314/0/page/1).

Shameless plug alert!