ORACLE

http://www.oracle.com/technology/jpa

Using JPA in Spring

Mike Keith michael.keith@oracle.com

Shaun Smith shaun.smith@oracle.com

About Mike

- Co-spec Lead of EJB 3.0 (JSR 220)
- Java EE 5 (JSR 244) expert group member
- Co-author of "Pro EJB 3: Java Persistence API"
- Architect for Oracle TopLink and EJB Container in OracleAS OC4J
- 15+ years experience in OO persistence and numerous persistence implementations
- Frequent presenter at conferences and events

About Shaun

- Co-Lead of Eclipse Dali JPA Tools Project
- Product Manager for Oracle TopLink
- OO programmer for almost 20 years—10 years experience with OO persistence
- Frequent presenter at conferences and events
- Previously a consultant building enterprise systems and an agile software development coach

About You

- How many people know of or have already used EJB 3.0 Java Persistence API?
- How many people are using any one of:
  - JBoss Hibernate
  - Oracle TopLink
  - BEA Kodo
- How many *think* you will be using the EJB 3.0 Java Persistence API in the future?

Goal

Learn some of the basic concepts and practices for using the JPA as your persistence layer in Spring 2.0

Agenda

- The Basics
- Spring as JPA *Consumer*
  - Creating a JPA DAO
  - Using the JPA API
- Spring as a JPA *Container*
- Summary

Spring and Persistence

- Most people need/use a persistence layer
  - Traditional Spring + Hibernate combination
- Spring supports lots of persistence options:
  - JDBC, Hibernate, TopLink, JDO, iBATIS, OJB
- Some of the problems:
  - Proprietary persistence APIs and metadata
  - Session/resource management
  - Coupled persistence layer
  - Different exceptions from different vendors/dbs
  - Varied transaction models and APIs

Spring and Persistence

- Solve some of the coupling problems using DAO design pattern and templates for API abstraction
- Exception translation to normalized Spring DataAccessException hierarchy
- Generic PlatformTransactionManager for generalized transaction mechanism
- Injection of resources (session factory, data source, etc.) into DAO for easier management

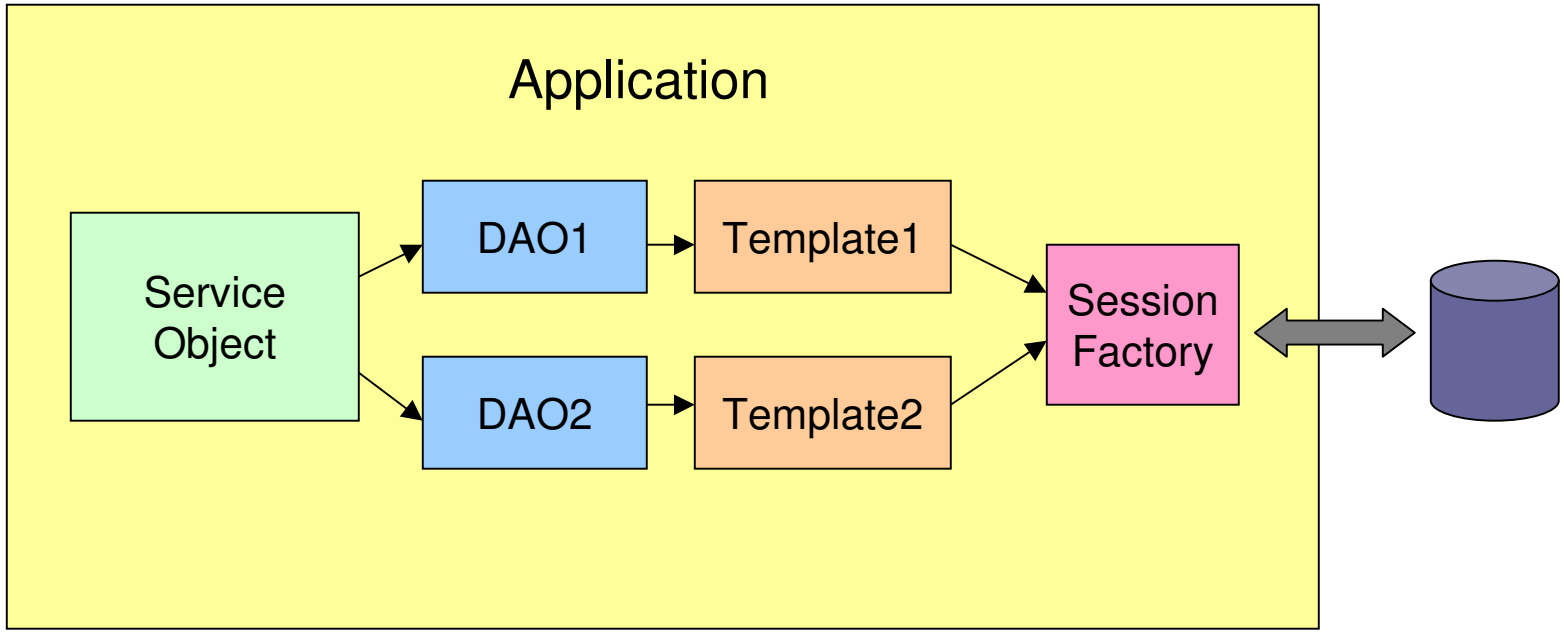
Java Persistence API (JPA)

- Separate document bundled as part of EJB 3.0 specification
- Merging of expertise from persistence vendors and communities
- Standardization of current persistence practices
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container

Spring as a JPA Consumer

- Uses JPA "bootstrap API" in non-EE runtime
- Intermediate application layer similar to other existing Spring persistence solutions
- Not particularly integrated with the persistence provider
- Configured using LocalEntityManagerFactoryBean
- JDBC connection parameters specified as properties in persistence.xml file
- JpaTransactionManager in Spring to match RESOURCE_LOCAL tx type in persistence.xml

Spring DAO Persistence



Using JPA DAO Support Classes

- Similar to DAO support for other persistence types in Spring
- Preconfigured **JpaDaoSupport** and **JpaTemplate** framework classes
- Inject EntityManagerFactory bean class
- EntityManager created and accessed through the template

JPA DAO Template Example

```
public class JpaTemplateClinic extends
  JpaDaoSupport implements Clinic {

  public Collection getVets() throws
      DataAccessException {

      return getJpaTemplate().find(
            "SELECT vet FROM Vet vet ORDER BY
            vet.lastName, vet.firstName");
  }
…
```

JPA DAO Template Config

```
…
<bean id="entityManagerFactory"
   class="org.springframework.orm.jpa.LocalEntityManager
   FactoryBean">
       <property name="persistenceUnitName"
             value="PetClinic"/>
</bean>

<bean id="clinic"
   class="org.springframework.samples.petclinic.jpa.JpaT
   emplateClinic">
       <property name="entityManagerFactory"
       ref="entityManagerFactory"/>
</bean>
…
```

JpaTemplate API

- JpaTemplate
  - contains
  - find
  - findByNamedQuery
  - flush
  - merge
  - persist
  - refresh
  - remove
  - doInJPA(…)—Executing random JPA code

JPA API

- Spring Beans use JPA API directly
  - no template classes
- EntityManager injection through JPA annotations
  - same annotations as JPA spec
  - Spring implements subset of EJB 3.0 container functionality

Using the JPA API

- EntityManager Injection
  - @PersistenceContext (JPA spec)
  - PersistenceAnnotationBeanPostProcessor
- Exception Translation
  - @Repository (Spring defined,non-JPA spec)
  - PersistenceAnnotationBeanPostProcessor

JPA API Example

```
@Repository
public class EntityManagerClinic implements Clinic {
   private EntityManager em;

   @PersistenceContext
   public void setEntityManager(EntityManager em) {
       this.em = em;
   }
   public Collection<Vet> getVets() {
       return em.createQuery(
              "SELECT vet FROM Vet vet
              ORDER BY vet.lastName, vet.firstName")
       .getResultList();
   }
```

JPA API Config

```
<bean id="entityManagerFactory"
   class="org.springframework.orm.jpa.LocalEntityManagerFactor
   yBean">
        <property name="persistenceUnitName"
        value="PetClinic"/>
</bean>

<bean id="clinic"
   class="org.springframework.samples.petclinic.jpa.EntityMana
   gerClinic"/>

<bean
   class="org.springframework.orm.jpa.support.PersistenceAnnot
   ationBeanPostProcessor" />
<bean
   class="org.springframework.dao.annotation.PersistenceExcept
   ionTranslationPostProcessor" />
```

JPA Container SPI

- JPA Container – Provider API
- Facilitates pluggable persistence providers
- Container passes provider config and runtime info for each persistence unit
  - Persistence unit info in persistence.xml
  - Root location of persistence unit
  - API for provider to register for class transformation
  - Disposable classloader that provider can use to examine classes without having to load them in the "real" classloader

Spring as a JPA Container

- Spring implements the Container SPI and acts as host JPA container
- Enables using container-managed EntityManagers in environments when normally not possible
  - non-EJB 3.0 servers
  - standard Java SE runtime
- Reads persistence.xml and passes persistence unit information to the provider
- When running in a server environment the weaver must be integrated with server loaders

'LocalContainer' Configuration

- Things to configure:
  - Persistence Unit Info
  - Database Connection/Data Source
  - Transactional Semantics
  - JPA Provider Properties
- With Spring JPA you have (at least) two config files:
  - Persistence.xml
  - ApplicationContext.xml

Basic Config Layout

- Persistence.xml
  - Persistence Unit
  - Provider Properties
- ApplicationContext.xml
  - Data Source
  - *Weaver Config*
  - *Persistence Unit Manager*
  - Transactional Semantics

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
   version="1.0">
   <persistence-unit name="PetClinic"
        transaction-type="RESOURCE_LOCAL">
     <properties>
       <property name="toplink.logging.level" value="FINEST"/>
        <property name="toplink.logging.timestamp" value="false"/>
        <property name="toplink.logging.thread" value="false"/>
        <property name="toplink.logging.session" value="false"/>
        <property name="toplink.throw.orm.exceptions"
               value="true"/>
     </properties>
   </persistence-unit>
</persistence>
```

ApplicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
   "http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>
   <bean id="dataSource"
   class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>
        <property name="username" value="sa" />
        <property name="password" value="" />
   </bean>
…
```

ApplicationContext.xml (2)

```
<bean id="entityManagerFactory"
   class="org.springframework.orm.jpa.LocalContainerEntityManagerFact
   oryBean">
   <property name="dataSource" ref="dataSource"/>
   <property name="jpaVendorAdapter">
      <bean
        class="org.springframework.orm.jpa.vendor.TopLinkJpaVendorAda
   pter">
         <property name="databasePlatform"
   value="org.springframework.samples.petclinic.toplink.EssentialsHSQ
   LPlatformWithNativeSequence"/>
         <property name="generateDdl" value="false"/>
         <property name="showSql" value="true" />
      </bean>
   </property>
</bean>
```

ApplicationContext.xml (3)

```
<bean id="transactionManager"
   class="org.springframework.orm.jpa.JpaTransactionManager">
   <property name="entityManagerFactory"
        ref="entityManagerFactory"/>
</bean>
```

Weavers

- Byte code weaving is used by some JPA providers to add persistence to POJO Entities.
  - e.g., for lazy loading of one-to-one and many-to-one relationships

```
@Entity
public class Pet extends NamedEntity {

    @ManyToOne(fetch=FetchType.LAZY)
    private Owner owner;
```

Weavers (cont.)

- Configuration is in application-config.xml

```
<bean id="entityManagerFactory"
   class="org.springframework.orm.jpa.LocalContainerEntityManagerFact
   oryBean">
   <property name="dataSource" ref="dataSource"/>
   <property name="loadTimeWeaver">
        <bean
        class="org.springframework.instrument.classloading.Instrument
   ationLoadTimeWeaver"/>
   </property>
…
</bean>
```

PersistenceUnitManager

- Spring's PersistenceUnitManager provides a mechanism for identifying persistence.xml files without a classpath scan.
- persistence.xml files need not be called "persistence.xml"
- LocalContainerEntityManagerFactoryBean has a DefaultPersistenceUnitManager but you can define your own and inject

ApplicationContext.xml (4)

```
<bean id="persistenceUnitManager"
   class="org.springframework.orm.jpa.persistenceunit.DefaultPersistenc
   eUnitManager">
     <property name="persistenceXmlLocations">
        <list><value>classpath*:META-INF/persistence.xml</value></list>
     </property>
     <property name="defaultDataSource" ref="dataSource" />
     <property name="loadTimeWeaver">
        <bean
        class="org.springframework.instrument.classloading.Instrumentat
   ionLoadTimeWeaver"/>
     </property>
</bean>
<bean id="entityManagerFactory"
   class="org.springframework.orm.jpa.LocalContainerEntityManagerFactor
   yBean">
 <property name="persistenceUnitManager"ref="persistenceUnitManager"/>
```

JPA/Spring Data Sources

- In Persistence.xml:

```
<persistence-unit name="PetClinic"
   transaction-type="RESOURCE_LOCAL">
   <non-jta-data-source>jdbc/PetClinic</non-jta-data-source>
```

- In ApplicationContext.xml:

```
<bean id="persistenceUnitManager"
   class="org.springframework.orm.jpa.persistenceunit.DefaultP
   ersistenceUnitManager">
   <property name="dataSources">
      <map>
         <entry key="jdbc/PetClinic" value-ref="dataSource"/>
      </map>
   </property>
```

Summary

- ✓ A lot of resources have been invested to provide advanced container-level JPA support in Spring 2.0
- ✓ Spring 2.0 supports pluggable JPA vendor implementations
- ✓ JPA is becoming the preferred Spring persistence API and a means of harmonizing the existing proprietary persistence layers
- ✓ Spring/TopLink Essentials provides a complete open source application development framework
  - ➢ TopLink Essentials ships with Spring 2.0!

Links

- TopLink Essentials—JPA Reference Implementation

  http://glassfish.dev.java.net/

- Dali JPA Tools project (a WTP sub-project) provides an open source tools for JPA development

  http://www.eclipse.org/dali

Links

- JPA Specification

  http://www.jcp.org/en/jsr/detail?id=220

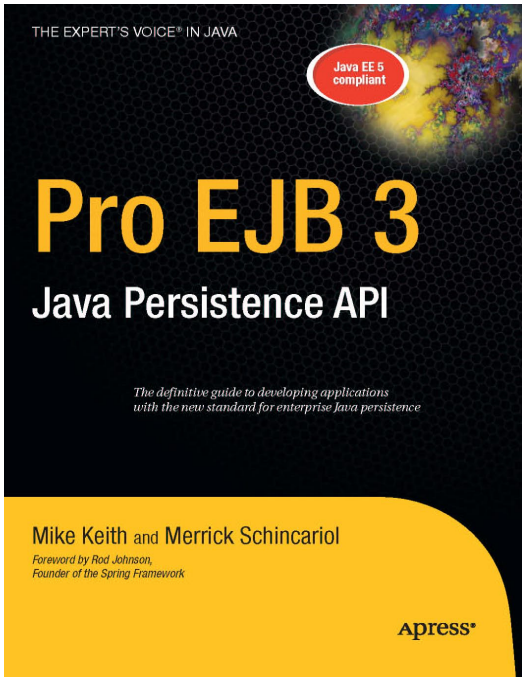
- JPA white papers, tutorials and other learning resources

  http://otn.oracle.com/jpa

Learning JPA

- Pro EJB 3: Java Persistence API

  Mike Keith & Merrick Schincariol
  (Foreword by Rod Johnson)



JPA Reference Implementation

- TopLink Essentials RI
  - Open source project on java.net
  - Shipped with Spring 2.0!
- Production-quality RI based on a proven product with a decade of Fortune 100 user base
- Works with any JPA-compliant container that implements the pluggability SPI
- Works in any Java SE JVM
- Works on virtually all known databases