Experts in delivering business-driven technology solutions.

Spring + JPA + Hibernate

Agenda

- Persistence
  - JdbcTemplate
  - Hibernate
- JPA
- Spring
  - Spring 2.x
  - JPA features

J2EE 1.4 Reality Check

- Common Technology Stack
  - Spring (IoC)
  - Hibernate (Light-Weight Persistence)
  - Stateless EJB

JPA – Java Persistence API

- JEE 5 / EJB3 Persistence
- Provides an ORM framework similar to Hibernate / JDO

- Good Bye Entity Beans!!!

Spring Persistence

- Spring JDBC
- Spring Hibernate
- Spring JPA
- Spring iBatis

Issues with SQL

- SQL isn't hard... just tedious
- redundant... repeating code

```
Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;
String sql = "select * from book where id=?";

try {
    con = ds.getConnection();
    ps = con.prepareStatement(sql);
    ps.setInt(1,i);
    rs = ps.executeQuery();
    return buildBook(rs);

} catch (SQLException e) {
} finally {
    try {
        rs.close();
    } catch (SQLException e) {
    }
    try {
        ps.close();
    } catch (SQLException e) {
    }
    try {
        con.close();
    } catch (SQLException e) {
    }
}
```

Focus

- DRY - Don’t Repeat Yourself
- Testable
- Concise
- Stop forcing all the checked exceptions

JDBCTemplate

Perficient

```
public List getBooks() {

    final String sql = "select id, isbn, author, title from book ";

    RowMapper mapper = new RowMapper() {

        public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
            Book book = buildBook(rs);
            return book;
        }
    };

    return jdbcTemplate.query(sql, mapper);

}
```

Remaining Challenges?

- Testability...
  - in-memory DB
  - HSQLDB vs. Oracle
  - The code is tied to a dialect!

ORM - The Good

- Object Relational Mapping
  - Makes the Dialect configurable!
  - Testable
  - Used to increase time to market

ORM - The Good

- Issues / Warnings
  - Forces compromises in the relational datastore
    - primary keys
    - triggers
    - ...
  - Lazy vs. Eager decisions
  - As the project grows the ORM pain grows

And the winner is... Hibernate

Perficient

Hibernate was the clear winner in the ORM race...

However it wasn't a standard...

Spring Provides Hibernate Support


```
<beans>

  <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <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>

  <bean id="mySessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"/>
    <property name="mappingResources">
      <list>
        <value>product.hbm.xml</value>
      </list>
    </property>
    <property name="hibernateProperties">
      <value>
        hibernate.dialect=org.hibernate.dialect.HSQLDialect
      </value>
    </property>
  </bean>

</beans>
```

Spring Hibernate Template

Perficient

```
public class ProductDaoImpl implements ProductDao {

    private HibernateTemplate hibernateTemplate;

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

    public Collection loadProductsByCategory(String category) throws DataAccessException {
        return this.hibernateTemplate.find("from test.Product product where product.category=?"
    }
}
```

Spring HibernateDaoSupport


```
public class HibernateProductDao extends HibernateDaoSupport implements ProductDao {

    public Collection loadProductsByCategory(String category) throws DataAccessException, MyException {
        Session session = getSession(false);
        try {
            Query query = session.createQuery("from test.Product product where product.category=?");
            query.setString(0, category);
            List result = query.list();
            if (result == null) {
                throw new MyException("No search results.");
            }
            return result;
        }
        catch (HibernateException ex) {
            throw convertHibernateAccessException(ex);
        }
    }
}
```

Spring HibernateTransactionManager


```
<beans>

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

  <bean id="myProductService" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces" value="product.ProductService"/>
    <property name="target">
        <bean class="product.DefaultProductService">
            <property name="productDao" ref="myProductDao"/>
        </bean>
    </property>
    <property name="interceptorNames">
      <list>
        <value>myTxInterceptor</value> <!-- the transaction interceptor (configured elsewhere) -->
      </list>
    </property>
  </bean>

</beans>
```

Hibernate Consequences

- XML focused
  - at least at the time
- Not standard
- Alternatives: JDO
  - Focused on ubiquitous data access instead of relational

JPA

JPA Benefits

- Standards-Based
- No Descriptors necessary
- Annotated POJOs
- Detached Object Support
  - Reduce Overhead of DTO / VO
- Improve Testability

JPA - Specification

- Packaging
- Entities
- Entity Operations
- Queries
- Metadata
- Life-cycle Model
- Callbacks

Persistence.xml

- In the classpath under the META-INF directory.

```
<persistence-unit name="unit1" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
                <property name="hibernate.hbm2ddl.auto" value="create"/>
                <property name="hibernate.ejb.autodetection" value="class"/>
                <property name="hibernate.connection.url"
                        value="jdbc:hsqldb:hsql://localhost:1234/employee"/>
                <property name="hibernate.connection.driver_class"
                        value="org.hsqldb.jdbcDriver"/>
                <property name="hibernate.connection.username" value="sa"/>
                <property name="hibernate.connection.password" value=""/>
        </properties>
  </persistence-unit>
</persistence>
```

Entity Requirements

- Must be annotated an Entity
- Public or Protected No-arg Constructor
- Must not be final
  - No final methods or variables
- Must be Serializable to be detached and serialized…

Persistent Fields

- Primitives and Strings
  - automatically become columns in the database
- Object fields
  - must be mapped by joins and foreign key relationships
- Fields marked ***transient*** are not persisted
- Fields annotated @Transient are not persisted

Customer Entity (from spec)

Perficient

```
@Entity(access=FIELD)
public class Customer {
  @Id(generate=AUTO) Long id;
  @Version protected int version;
  @ManyToOne Address address;
  @Basic String description;
  @OneToMany(targetEntity=com.acme.Order.class,
  mappedBy="customer")
  Collection orders = new Vector();

  @ManyToMany(mappedBy="customers")
  Set<DeliveryService> serviceOptions = new HashSet();
  public Customer() {}

  public Collection getOrders() { return orders; }
  public Set<DeliveryService> getServiceOptions() {
```

POGO for Exceptional Terseness

Perficient

```
@Entity
class Book {

    @Id
    @GeneratedValue
    Long Id

    @NotNull
    String title


    String author
    String ISBN
}
```

JPA Persistence Interfaces

- EntityManager
  - Interface to interact with persistence context.
  - @PersistenceContext
- EntityManagerFactory
  - Creates an EntityManager
  - @PersistenceUnit

Entity Manager


```
void persist(Object entity);
<T> T merge(T entity);
void remove(Object entity);
<T> T find(Class<T> entityClass, Object primaryKey);
<T> T getReference(Class<T> entityClass, Object
  primaryKey);
void flush();
void refresh(Object entity);
boolean contains(Object entity);
void close();
boolean isOpen();
EntityTransaction getTransaction();
```

Acquiring a Manager

- Injection in Stateless Bean

```
@PersistenceContext
public EntityManager em;
```

- OR

```
@PersistenceContext(unitName="order")
EntityManager em;
```

- From Java Application

```
EntityManagerFactory emf =     Persistence.createEntityManagerFactory
("unit1");
EntityManager em = emf.createEntityManager();
```

JPA Query

- **JPQL**
- **Example:**

```
public List<Session>
   findSessionByCatagory(String name) {
        return entityManager.createQuery(
         "from Session session where
           session.catagory.name=:name")
        .setParameter("name", name).getResultList();
}
```

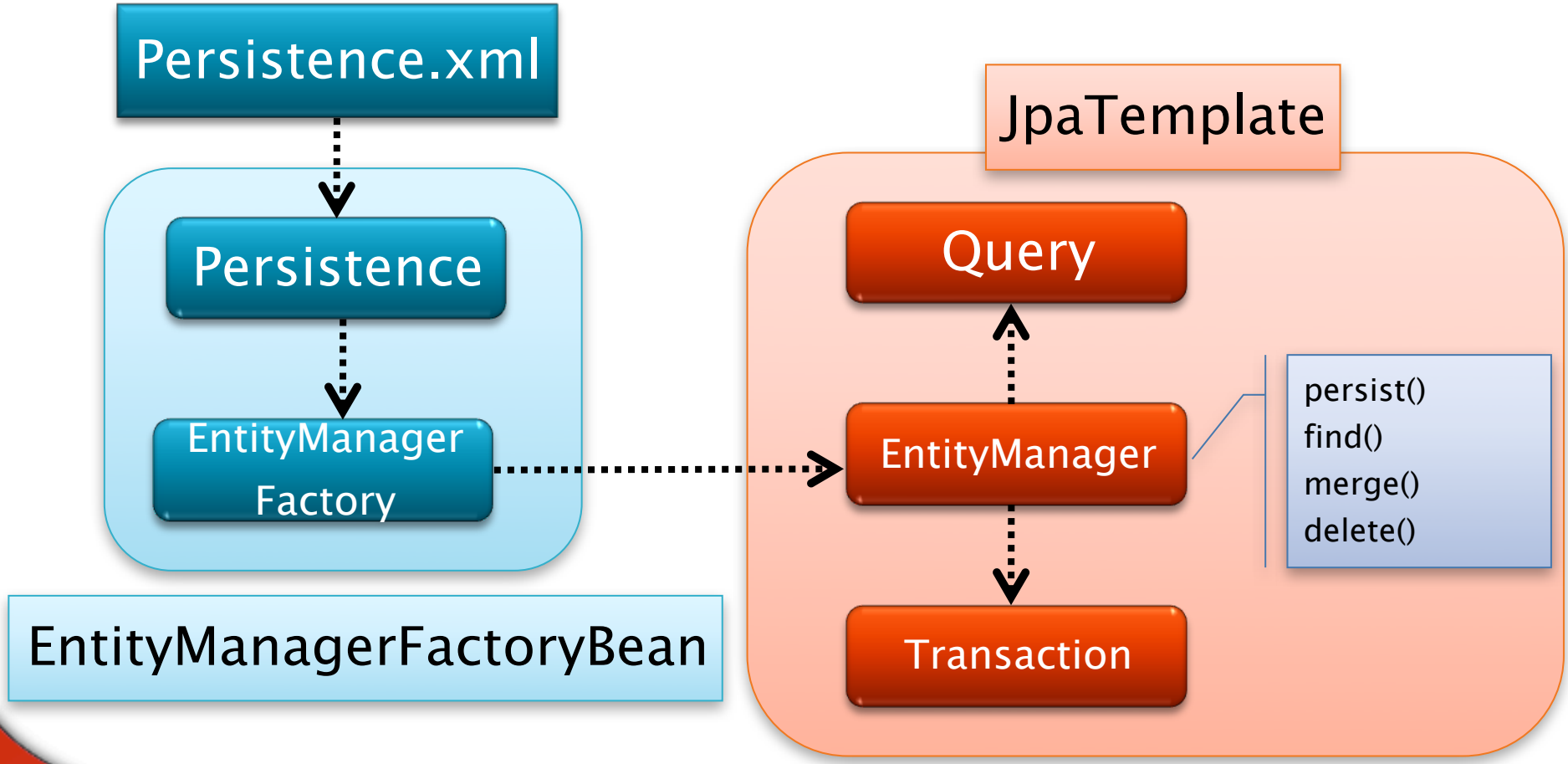
JPA Challenges

- 2 Programming Models
  - Standalone application
  - container managed
- Bootstrapping

Spring 2.x

Spring 2 introduces JPA support

Spring - JPA Relationship



Spring 2 JPA Support

- org.springframework.orm.jpa package
  - Contains subset of the JPA container
- JpaDaoSupport
  - similar to other DAO support classes like HibernateDaoSupport
- LocalEntityManagerFactoryBean
  - Provides resource bootstrapping for non-jndi lookups

Spring / JPA Approaches

- JpaDaoSupport Approach
  - Not preferred approach
  - Similar to HibernateDaoSupport
  - Requires Spring Configuration of the EntityManager
- Pure JPA Approach
  - Preferred approach
  - No spring references necessary in the code
    - with the exception of @Transactional

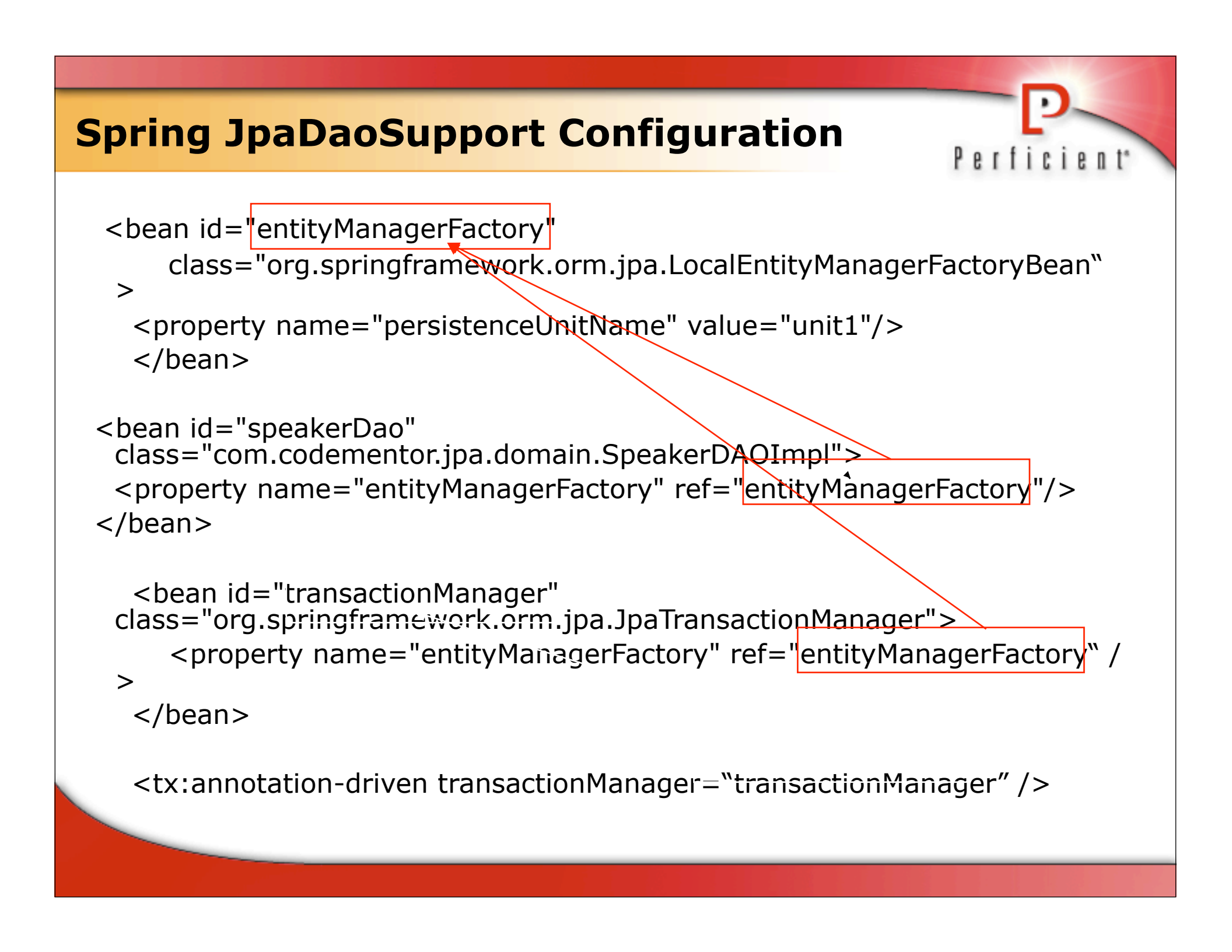
Approach 1: JpaDaoSupport

- Provides great support with **JpaDaoSupport** with **JpaTemplate** to simplify common code
  - very familiar to hibernate developers
- Consequences:
  - import of spring framework
    - not exactly POJO
  - requires spring configuration of entitymanager

JpaDaoSupport Example: SpeakerDaoImpl


```
package com.nfjs.jpa;

import java.util.List;
import org.springframework.orm.jpa.support.JpaDaoSupport;

public class SpeakerDAOImpl extends JpaDaoSupport implements SpeakerDAO {

    public Speaker findSpeaker(long id) {
        return getJpaTemplate().find(Speaker.class,id);
    }

    public List<Speaker> findSpeakerByCatagory(String catagory) {
        return getJpaTemplate().find("select distinct s from Speaker s, Session
session where session.catagory.name=?1 and session.speaker.id = s.id",catagory);
    }
```

Spring JpaDaoSupport Configuration




```
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean"
>
  <property name="persistenceUnitName" value="unit1"/>
  </bean>

<bean id="speakerDao"
  class="com.codementor.jpa.domain.SpeakerDAOImpl">
  <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>

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

  <tx:annotation-driven transactionManager="transactionManager" />
```

Approach 2: Spring / Pure JPA Configuration

- Leverage the persistence.xml in classpath:/META-INF

```
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean" >
   <property name="persistenceUnitName" value="unit1"/>
</bean>
```

- DAO with no Spring references, however it contains @PersistenceContext annotated EntityManager

```
<bean id="conferenceDao"
 class="com.codementor.jpa.domain.ConferenceDAOImpl"/>
```

- Spring configuration which injects JPA annotationed EntityManager

```
<bean
 class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor‟ />
```

Pure JPA Code Example: ConferenceDaoImpl


```
package com.nfjs.jpa;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.transaction.annotation.Transactional;

public class ConferenceDAOImpl implements ConferenceDAO {

    @PersistenceContext
    private EntityManager entityManager;


    public void setEntityManager(EntityManager entityManager) {
        this.entityManager = entityManager;
    }
  ...
```

Pure JPA Spring Configuration


```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean" >
        <property name="persistenceUnitName" value="nfjs"/>
    </bean>

<bean id="conferenceDao" class="com.nfjs.jpa.ConferenceDAOImpl"/>

<bean
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"
/>


</beans>
```

No PU No Problem

- The LocalContainerEntityManagerFactoryBean can be configured with all Persistent Unit information.

```
<bean id="entityManagerFactory"
 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="jpaVendorAdapter">
   <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="showSql" value="true"/>
    <property name="generateDdl" value="true"/>
    <property name="databasePlatform"
        value="org.hibernate.dialect.HSQLDialect"/>
   </bean>
  </property>
 </bean>
```

Transactions

- XML Configuration

```
<tx:annotation-driven />
```

- Annotation

```
@Transactional(readOnly = false,
  propagation =  Propagation.REQUIRES_NEW)
Public void doSomething() {
```

** transaction manger bean id must be transactionManger or configured with the xml configuration above.

Scoping the Transactional

Perficient

```
public class BookService {

    @Autowired
    BookDAO dao;

    @Transactional
    public void save(Book book) {
        dao.save(book);

    }

}
```

Transactions are best at the level of a service class

Test JPA with Spring


```
public class SpeakerDAOTest extends AbstractJpaTests {

private SpeakerDAO speakerDao;

  public void setSpeakerDao(SpeakerDAO speakerDao) {
    this.speakerDao = speakerDao;
  }

  protected String[] getConfigLocations() {
    return new String[] {"classpath:/jpaContext.xml"};
  }

  protected void onSetUpInTransaction() throws Exception {
    jdbcTemplate.execute(
  "insert into speaker (id, name, company) values (1, 'Ken', 'CodeMentor')");
```

AbstractJpaTests Benefits

- getConfigLocations ()
  - Separates test from production configuration
  - Allows for multiple configurations
- Injected Dependencies By Type
  - field references
- Every Test
  - Starts a Transactions
  - Rolls back Transaction
- Leverage jdbcTemplate for SQL checks

Demo

JPA with Spring

References

- http://www.springframework.org/
- http://java.sun.com/developer/technicalArticles/J2EE/jpa
- http://www.hibernate.org/hib_docs/annotations/reference/en/html/entity.html

Questions

Perficient

kensipe@codementor.net

twitter: @kensipe