Spring with JPA

Tim Cunnell

July 2008

INGRES

Outline

What's coming up?

- **Introduction**
- **Why Spring?**
- **What is Spring?**
- **Why JPA?**
- **What is JPA?**
- **How does JPA fit into Spring?**
- **Example Application Architecture**
- **Spring/JPA Best Practices**

- **Questions?**

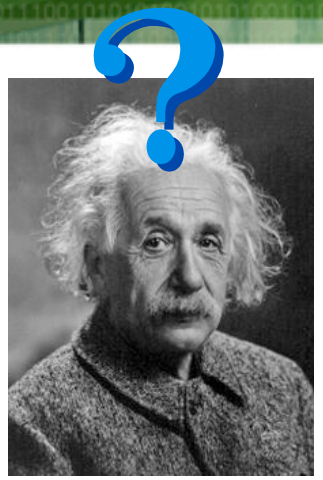
Introduction

- Object Oriented design patterns – GoF
- C++ and Smalltalk, then…
- Java and J2EE
- J2EE + wise beardy people = J2EE Design Patterns
- Business Delegate, Session Façade, DAO
- But how do I glue it all together??

Spring Framework

INGRES

Why Spring?



- J2EE: Enterprise scale applications:
- Enterprise scale complexity

- Common issue: how to wire up different elements?
- Frameworks developed to address this problem, known as ‘lightweight containers’
- PicoContainer: Thoughtworks
- Spring: Interface21

What is Spring?

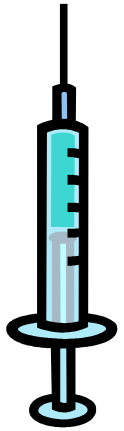
Inversion Of Control (IoC)

- **What aspect of control is being inverted?!**
- **Early UI comprised of a sequence of events:**
  - ‘Enter name: ‘
  - ‘Enter address: ‘ etc.
- **GUI: main control of your program is inverted and moved to the framework**
- **BUT IoC is common to frameworks, too generic a term**
- **Hence ‘Dependency Injection’**

Spring

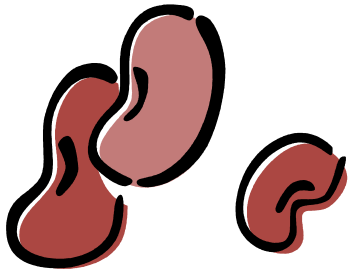
Dependency Injection

- **Setter Injection**
- **Constructor Injection**
- **Spring supports both but Setter Injection tends to be preferred by developers**
  - Generally advocated by Spring
  - Large number of constructor arguments can get unwieldy
- **Promotes decoupling**



Spring

Containers and Beans



- **Beans – the backbone of your application**
- **Instantiated, assembled, managed by the IoC container**

- **`org.springframework.beans.factory.BeanFactory` – is the actual Spring IoC container**
- **Instantiates, configures, assembles dependencies**

Spring

A Bean Factory in action

Maps to the 'setBeanTwo' method

```
<beans>
    <bean id="exampleBean" class="examples.ExampleBean">
        <!-- setter injection using the 'ref' attribute -->
        <property name="beanTwo" ref="yetAnotherBean"/>
        <property name="integerProperty" value="1"/>
    </bean>
    <bean id="anotherExampleBean" class="examples.AnotherBean"/>
    <bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
</beans>
```

Fully qualified class name

```
public class ExampleBean {
    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;
    public void setBeanOne(AnotherBean beanOne) {
        this.beanOne = beanOne;
    }
    public void setBeanTwo(YetAnotherBean beanTwo) {
        this.beanTwo = beanTwo;
    }
    public void setIntegerProperty(int i) {
        this.i = i;
```


INGRES

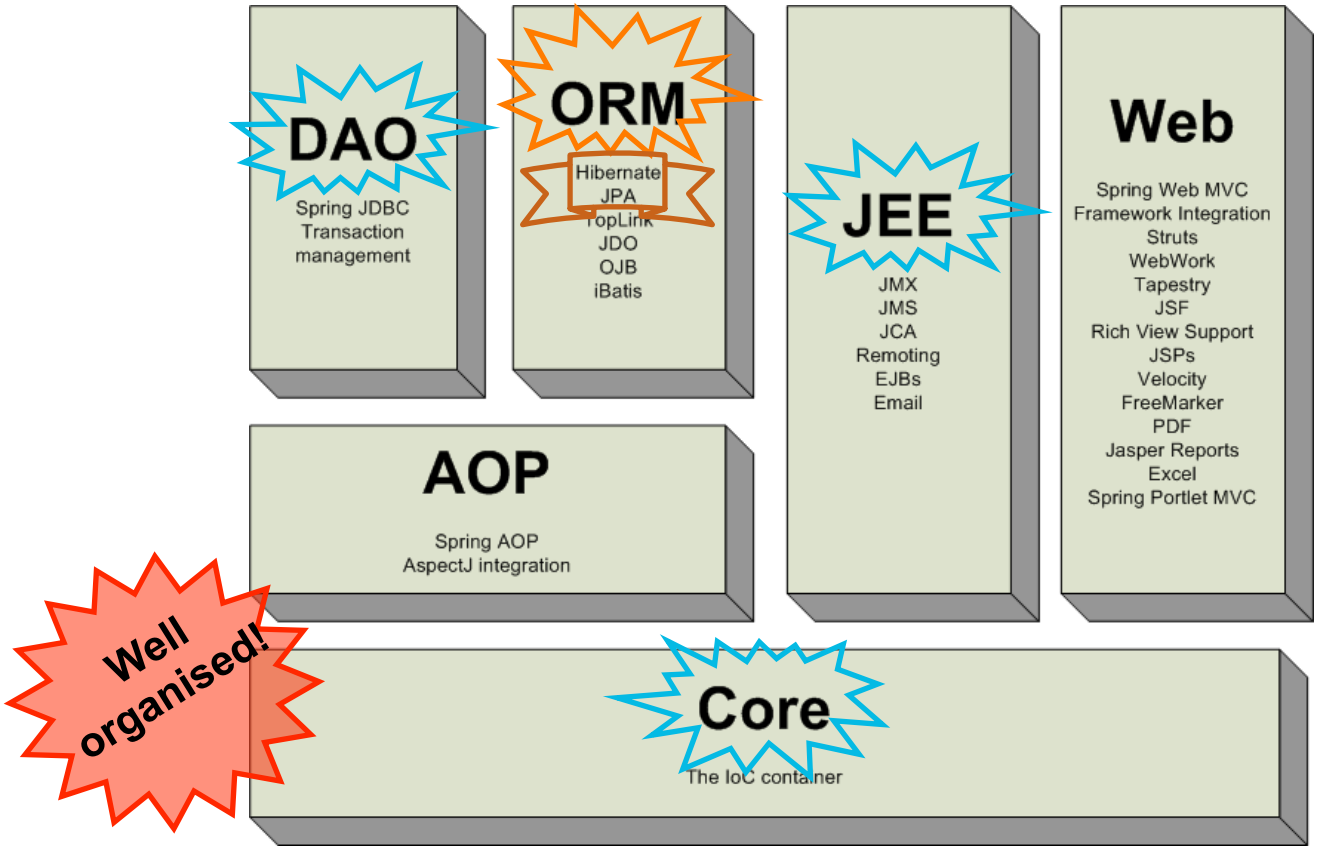
Spring

BeanFactory or ApplicationContext?

- BeanFactory – just instantiates and configures beans
- ApplicationContext is a sub-interface of BeanFactory
- ApplicationContext also provides infrastructure for enterprise-specific features
- e.g. transactions and AOP
- So…
- Favour the use of ApplicationContext over BeanFactories

Spring

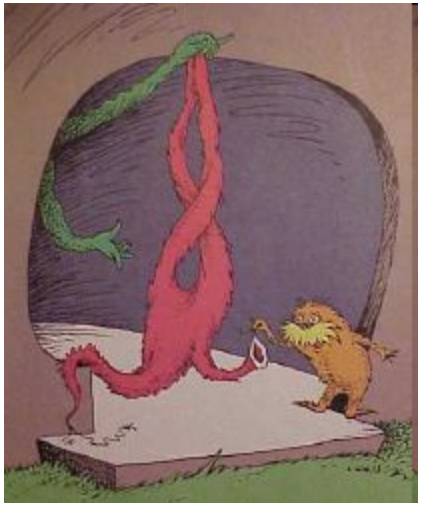
The Spring Framework (from springframework.org):



Spring

Non-invasive IoC Container

- **Won't invade your code with dependency on its APIs – you don't have to import/extend Spring APIs**
- **Business objects can therefore potentially be run in different Dependency Injection frameworks without code changes**
- **Almost any POJO can become a component in a Spring bean factory**



A Thneed's a Fine-Something-That-All-People-Need!

Java Persistence API

INGRES

Java Persistence API – why?

- **Developed by the EJB 3.0 expert group (as part of JSR220)**
- **JPA – but why?**
- **Provides ease of development through:**
  - Simplification of the development of Java EE and Java SE applications using data persistence
  - Single, standard persistence API
  - Standardizes object-relational mapping
- **That sounds fantastic! But what is it??**

What is JPA?

- JPA is a POJO persistence API for object/relational mapping
- Contains a full ORM specification
- Supports a rich, SQL-like query language for static and dynamic queries
- Supports the use of pluggable persistence providers, e.g. Hibernate, TopLink, JDO
- Identify entities and define relationships between them with annotations

Entities

- Entity: POJO – replaces the EJB1-2.x Entity bean
- Lightweight persistence domain object, maps to db table
- Row = entity instance
- Fields are persisted (unless marked `@Transient` or `transient`) but
- Preferable to follow JavaBeans conventions
- Looks like this:

Entities - example

I'm an entity

I'm Serializable

```
@Entity
public class Artist implements Serializable {

    private static final long serialVersionUID = -6872825805935710407L;
    private Integer id;
    private String name;
    private String genre;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Integer getId() {
        return id;
    }


    protected void setId(Integer id) {
        this.id = id;
    }
```

This is my primary key

Id generation strategy – SEQUENCE is default for Ingres

(code omitted for clarity)

Relationships

- Entities have relationships with each other, annotations are:
  - `OneToOne, OneToMany, ManyToOne, ManyToMany`
- The code looks like this:

```
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
public List<Track> getTracks() {
    return tracks;
}
public void setTracks(List<Track> tracks) {
    this.tracks = tracks;
}
.
.
```

JPA

What are the advantages?

- Fewer classes and interfaces
- Deployment descriptor relief through annotations
- Cleaner, easier, standardized ORM
- No need for lookup code
- Support for inheritance, polymorphism and polymorphic queries
- Easier to test outside an EJB container
- but…

JPA

What are the disadvantages?

- Lack of control over generated SQL
- Proprietary query languages are often not sophisticated enough
- Less visibility of the efficiency of the generated SQL

Spring and JPA

INGRES

ORM Data Access with Spring

- **Integration with JPA, Hibernate, Toplink etc. for resource management, DAO implementation support, tx strategies**
- **2 integration styles:**
  - Spring's DAO 'templates' or
  - Coding DAOs against plain JPA/Hibernate etc. APIs
- **Dependency Injection for both and participation in Spring's resource and tx management**

Spring and JPA

Configuration

- **Spring JPA: `org.springframework.orm.jpa`**
- **3 ways to set up JPA EntityManagerFactory:**
  1. **`LocalEntityManagerFactoryBean`**
     - Simplest, most limited form: use only in simple deployments
  2. **`EntityManagerFactory`** from JNDI
     - Use in a JEE environment
  3. **`LocalContainerEntityManagerFactoryBean`**
     - Full control over **`EntityManageFactory`**
     - Most powerful JPA config
- **But what is load time weaving?!**

Spring and JPA

Load-time Weaving

- Weaving: used with AOP, adding code to binary class files
- Compile-time, Post-compile, load-time weaving
- Requires weaving class loader(s)
- LoadTimeWeaver interface allows JPA ClassTransformer instance to be plugged
- Spring provides implementations for Tomcat, Glassfish etc. classloaders

Spring and JPA

`JpaTemplate` and `JpaDaoSupport`

- DAOs receive EntityManagerFactory through dependency injection
- Config (from applicationContext.xml):

```
<beans>
  <bean id="musicDAO" class="uk.co.luminary.dao.MusicDAO">
  <property name="entityManagerFactory"
            ref="entityManagerFactory" />
  </bean>
</beans>
```

- Code…

Spring and JPA

- **DAO code fragment:**

```
public class MusicDAOJpaImpl extends JpaDaoSupport
                implements MusicDAO {


    public Track getTrackByIndex(Record record, int index) {
        List<Track> results = getJpaTemplate().findByNamedQuery(
         "track.byIndex",
          new Object[] { Integer.valueOf(index), record.getId()});
          return results == null || results.size() < 1 ? null :
                        (Track) results.get(0);
    }
}
```

Named query to execute

Spring and JPA

- Where `track.byIndex` named query is defined in the Track entity as:

```
@Entity
@NamedQuery(name="track.byIndex",
            query="SELECT t FROM Track t WHERE
    t.trackNumber = ? AND t.record.id = ?")
public class Track implements Serializable {
.
.
.
}
```

Spring and JPA

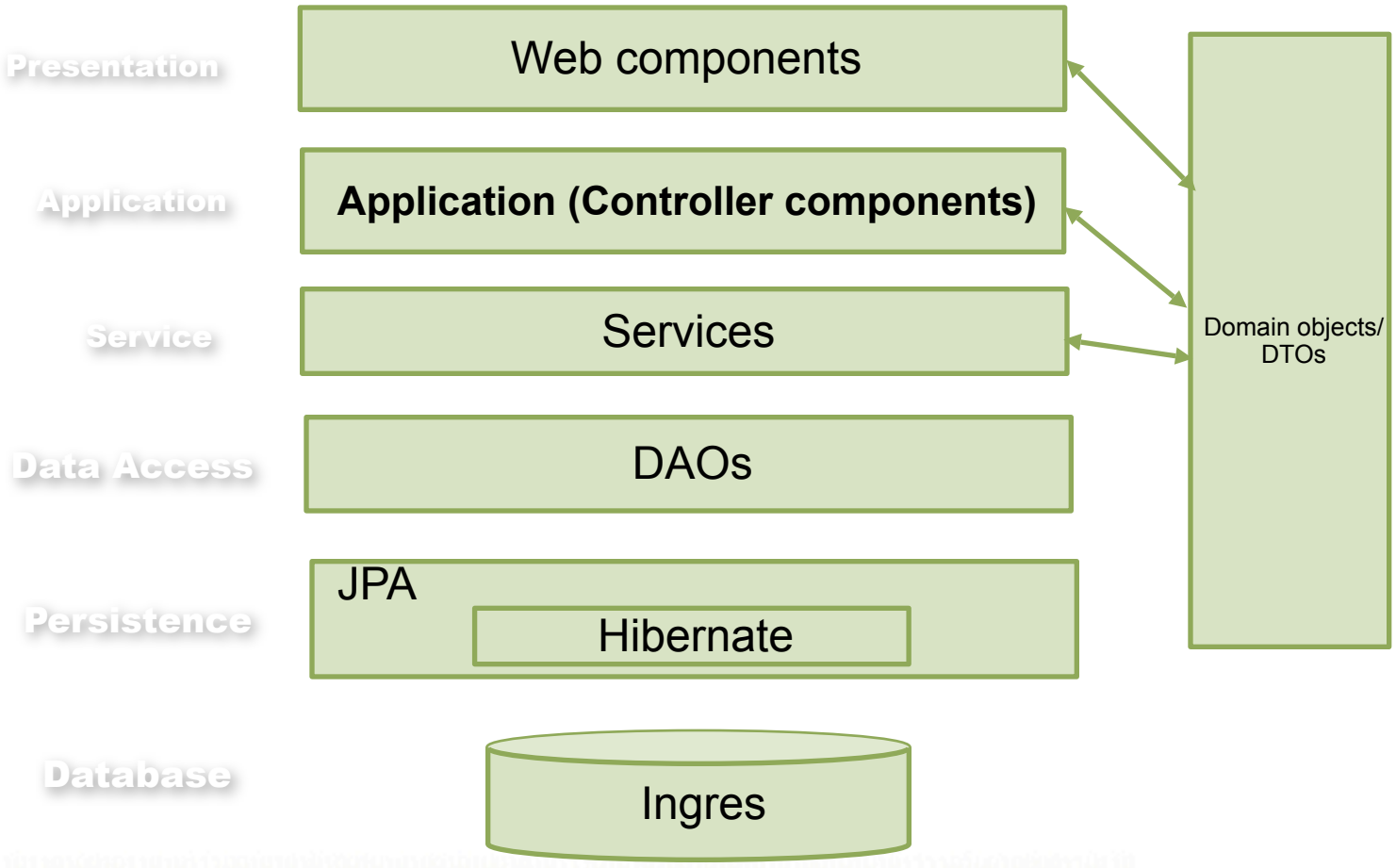
DAOs based on plain JPA

- Use a injected (shared) EntityManagerFactory and `@PersistenceContext` annotation
- Create an EntityManagerFactory like this:

```
EntityManager em = this.emf.createEntityManager();
```

- Then create and execute your query against the EntityManagerFactory
- Add the `@Repository` annotation to apply Spring's exception translation transparently

Example architecture



Spring - more than JPA

- **JDBC Core Classes – JDBCTemplate**
- **Connection handling**
- **SQLException to DataAccessException hierarchy**
- **JTA and JDBC transactions**
- **Also supports**
  - named parameters
  - database connection control
  - modelling JDBC ops as Java objects
  - stored procedures, functions etc.

Spring – JPA Best Practises

A few best practices:

- **Use only exact datatypes for primary keys**
- **Design JPA code for both Java SE and Java EE**
- **Use JPA code in the appropriate layer**
- **Favour self-describing code over metadata and comments**
- **Apply naming conventions for more readable JPA code**
- **Use Java EE 5 and EJB 3.0 best practices**
- **Use Spring's `@Repository` annotation**

Links

- **JPA:**

http://java.sun.com/javaee/technologies/persistence.jsp

- **Spring:**

http://www.springframework.org/documentation

- **Dependency Injection:**

http://martinfowler.com/articles/injection.html

- **Introduction to the Spring Framework 2.5:**

http://www.theserverside.com/tt/articles/article.tss?l=IntrotoSpring25

Wrap-up

- Spring – dependency injection etc.
- JPA
- Spring JPA programming model
- Links – more links to articles etc. available on request

Any questions?

tim.cunnell@luminary.co.uk

INGRES