



Building Enterprise Applications with Spring and JPA

Robert Berger
bBerger Technologies, Inc.
<http://www.bberger.net>
bob@bberger.net

1

Problems with EJB (through EJB 2.1)



- EJB framework is **intrusive**
- Application classes are tightly coupled to the framework
 - Application classes must:
 - Extend framework defined classes
 - Call framework defined lookup methods
- Application components cannot be created outside EJB container
 - Servlet container
 - Standalone process
 - Desktop application
 - Unit test

2

Problems with EJB (through EJB 2.1)



- No support for inheritance and polymorphism in Entity Beans
 - Two of the cornerstones of Object Oriented Design!

3


J2EE Design Patterns



- Many are work-arounds for J2EE framework problems
- Service Locator Pattern
 - Consolidate framework lookup calls in a service locator object called by the application components
- Data Transfer Object Pattern
 - Used to copy data to UI code that does not execute within a transaction
 - Requires 2 Java classes for every domain model class
 - Entity bean class
 - DTO class

4

Loose Coupling between Framework and Application Objects



“Persistent domain models often represent the core intellectual property of an application, and the most valuable product of business analysis. To preserve this investment, they should ideally be independent of the means used to persist them.”

“The service layer also represents a valuable investment that -- with good analysis -- should have a long lifetime. Thus it should also be as independent as possible of the technologies that enable it to function as part of a working application.”

Professional Java Development with the Spring Framework
Rod Johnson, et al. 2005

5

POJO Frameworks



- POJO: Plain Old Java Object
- No need to:
 - Extend framework defined classes
 - Call framework defined lookup methods
 - Throw/catch framework defined exceptions
- POJO framework combines POJO classes with metadata to achieve desired behavior

6

Hibernate



- Persistence (Object Relational Mapping)
- Analogous to Entity Beans in EJB
- Support for inheritance and polymorphism
- Detached objects- no need for separate DTO classes

7

Spring



Expert One-on-One J2EE Design and Development
Rod Johnson 2002

Expert One-on-One J2EE Development without EJB
Rod Johnson and Juergen Hoeller 2004

Professional Java Development with the Spring Framework
Rod Johnson, et al. 2005

8

Spring



- Inversion of Control
- DAO Interface Pattern
- Aspect Oriented Programming
- Transactions
- Acegi security framework
- Remote Execution
- Spring-WS
- Spring MVC
- Spring Web Flow

9

Inversion of Control



- Dependency Injection
 - Provides application objects with objects they depend on (dependencies)
 - Other application objects
 - Resources
 - JDBC data sources
 - JMS queues and topics
 - Transaction managers
 - Loggers
 - Connections to legacy systems

10

Dependency Injection



- Hollywood Principle
 - “Don’t call me, I’ll call you.”
 - Application classes provide property setter methods for dependencies.
 - Framework is configured to call the application classes’ property setters to inject dependencies.
 - Constructor injection also available.

11

Dependency Injection Containers



- Spring
- HiveMind
- PicoContainer
- Google Guici
- JBoss Microcontainer
- EJB3

12

Dependency Injection

- Spring Beans
 - Objects whose life cycle is managed by Spring
 - Created by Spring
 - Configured via Dependency Injection
 - Scope
 - singleton
 - One instance per application
 - prototype
 - New instance for every request
 - request (web application)
 - session (web application)
 - global-session (portlet)

13

Dependency Injection

JDBC DataSource for single threaded application:

```

<bean id="dataSource"
  class="org.springframework.jdbc.datasource.SingleConnectionDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://localhost:3306/TimeTrack"/>
  <property name="username" value="rwb"/>
  <property name="password" value="foo"/>
</bean>

<bean id="partyDAO" class="net.bberger.party.dao.JDBCPartyDAO">
  <property name="dataSource" ref="dataSource"/>
</bean>

```

14

Dependency Injection

JDBC DataSource with connection pooling for multi-threaded application:

```

<bean id="dataSource"
  class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://localhost:3306/TimeTrack"/>
  <property name="username" value="rwb"/>
  <property name="password" value="foo"/>
  <property name="initialSize" value="5"/>
  <property name="maxActive" value="10"/>
</bean>

<bean id="partyDAO" class="net.bberger.party.dao.JDBCPartyDAO">
  <property name="dataSource" ref="dataSource"/>
</bean>

```

15

POJO Frameworks- Spring

Dependency Injection



JDBC DataSource from JEE application server:

```
<jee:jndi-lookup id="dataSource" jndi-name="TimeTrackDataSource"/>


<bean id="partyDAO" class="net.bberger.party.dao.JDBCPartyDAO">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

16

POJO Frameworks- Spring

Dependency Injection

Manual Wiring



```
<bean id="partyService"
  class="net.bberger.party.service.PartyService">
  <property name="zipDAO"
    ref="zipDAO" />
  <property name="stateDAO"
    ref="stateDAO" />
  <property name="addressDAO"
    ref="addressDAO" />
  <property name="personDAO"
    ref="personDAO" />
  <property name="partyDAO"
    ref="partyDAO" />
  <property name="organizationDAO"
    ref="organizationDAO" />
</bean>
```

17

POJO Frameworks- Spring

Dependency Injection

Auto Wiring



```
<beans default-autowire="byType">

<bean id="partyService"
  class="net.bberger.party.service.PartyService"/>
```

18

Dependency Injection



- Dependency Injection for object instances not created directly by Spring
- Introduced in Spring 2.0
- Implemented using AspectJ's load-time code weaving
- @Configurable annotation

19

Application Context



Standalone application:

```
new ClassPathXmlApplicationContext("applicationContext.xml");
```

web.xml:

```
<context-param>  
  <param-name>contextConfigLocation</param-name>  
  <param-value>/WEB-INF/tomcat_applicationContext.xml</param-value>  
</context-param>  
<listener>  
  <listener-class>  
    org.springframework.web.context.ContextLoaderListener  
  </listener-class>  
</listener>  
<listener>  
  <listener-class>  
    org.springframework.web.context.request.RequestContextListener  
  </listener-class>  
</listener>
```

20

Integration with Web Applications



Servlet:

```
ApplicationContext appContext =  
  WebApplicationContextUtils.getWebApplicationContext(  
    getServletContext());
```

Alternative: use AspectJ based Dependency Injection.

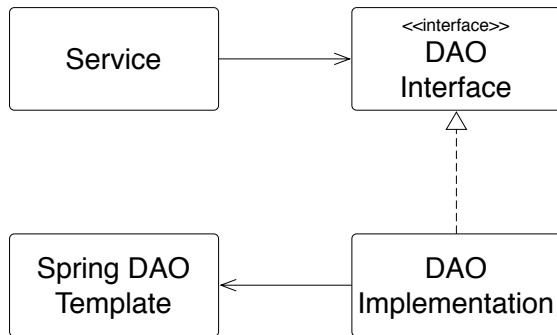
JSF (faces-config.xml)

```
<application>  
  <variable-resolver>  
    org.springframework.web.jsf.DelegatingVariableResolver  
  </variable-resolver>  
</application>
```

Makes Spring Beans available to JSF expression language.

21

POJO Frameworks- Spring
DAO Interface Pattern



22

POJO Frameworks- Spring
Spring DAO Templates

- Setup
- Cleanup
- Error handling
- Can be used independently of rest of Spring-- no framework tie-in

23

POJO Frameworks- Spring
Spring DAO Templates

- JDBC
- iBATIC SQL Maps
- Hibernate
- Toplink
- JDO
- JPA

24

POJO Frameworks- Spring
DAO Exceptions



JDBC

```
catch (SQLException e) {  
    switch (e.getErrorCode()) {  
  
    }  
}
```

Spring

```
catch (EmptyResultSetDataAccessException e) {  
  
}
```

25

POJO Frameworks- Spring
DAO Exceptions



- Unified across drivers and persistence frameworks
- Unchecked
 - Components that can't fix a problem don't have to catch/throw the exception.

26

POJO Frameworks- Spring
JDBC Query
(without Spring JDBC Template)



```
public int countParties() {  
    Connection conn = null;  
    PreparedStatement stmt = null;  
    ResultSet rs = null;  
  
    try {  
        conn = dataSource.getConnection();  
        stmt = conn.prepareStatement(COUNT_QUERY);  
        rs = stmt.executeQuery();  
        rs.first();  
        return rs.getInt(1);  
    } catch (SQLException e) {  
        throw new RuntimeException(e);  
    } finally {  
        try {  
            if (rs != null) {  
                rs.close();  
            }  
            if (stmt != null) {  
                stmt.close();  
            }  
            if (conn != null) {  
                conn.close();  
            }  
        } catch (SQLException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```

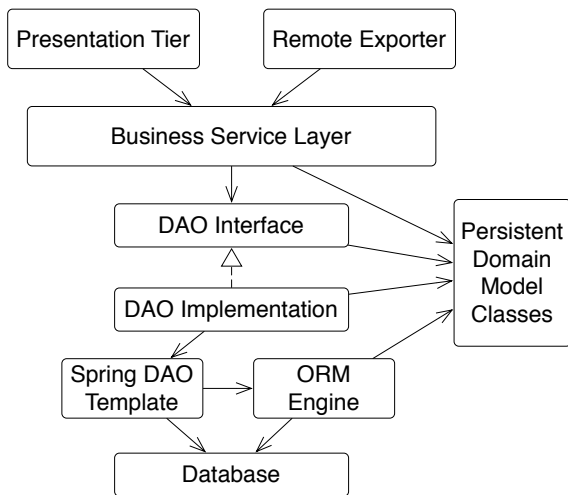
27

POJO Frameworks- Spring
Spring JDBC Template

```
public int countParties() {  
    return jdbcTemplate.queryForInt(COUNT_QUERY);  
}
```

28

Typical Spring Application

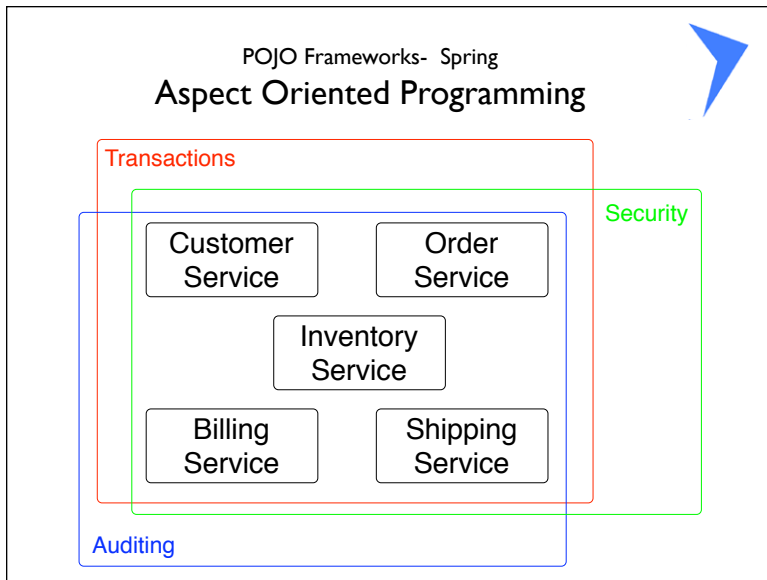


29

POJO Frameworks- Spring
Aspect Oriented Programming

- Separation of concerns
- Cross-cutting concern
 - Applies to many methods in many objects
 - Logging
 - Security
 - Transactional Integrity
 - Remote Execution

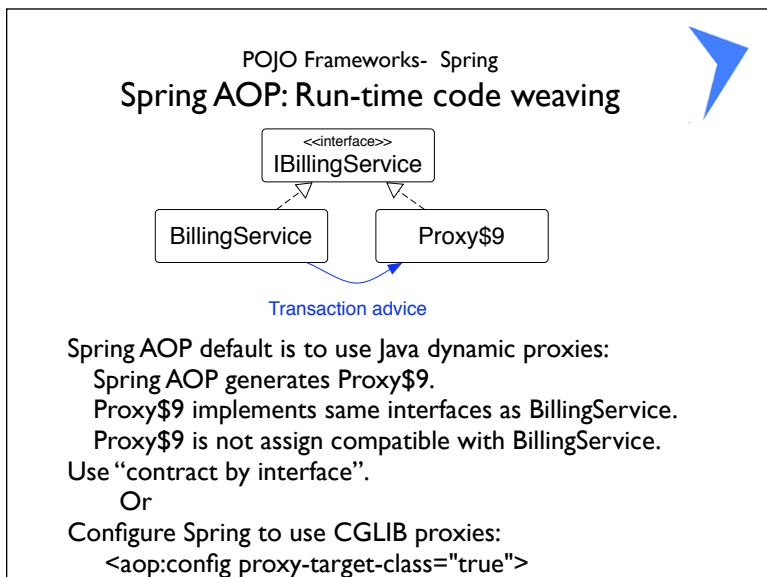
30



31

- POJO Frameworks- Spring
- ## Aspect Oriented Programming
- Code weaving
 - Merges advice behavior into target object
 - Compile-time (AspectJ)
 - Load-time (AspectJ)
 - byte code modification
 - Run-time (Spring AOP)
 - Proxy objects

32



33

Spring AOP



```
<tx:advice id="txAdvice"
  transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="*" />
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="serviceOperation"
    expression=
      "execution(* net.bberger.*.service.*Service.*(..))"/>
  <aop:advisor advice-ref="txAdvice"
    pointcut-ref="serviceOperation"/>
</aop:config>
```

34

Acegi Security Framework



- Declarative, role based security
- Web URL's
- JSP content
 - <authz:authorized ifAnyGranted="ROLE_ADMIN,ROLE_MANAGER">
- Method invocation
 - Service object methods
 - Persistent entity operations (CRUD)
- Integration with other security technologies
 - JAAS
 - LDAP
 - X509 Certificates

35

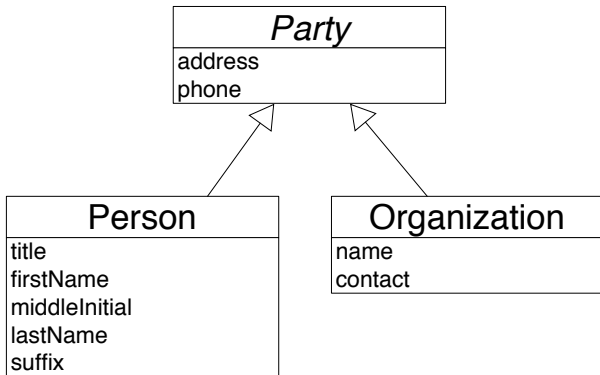
JPA



- Java Persistence API
- Defined as part of EJB3 (Entity Beans)
- Usable outside of EJB container
- Providers:
 - Hibernate (Redhat/JBoss)
 - OpenJPA (Apache)
 - Toplink (Oracle)
 - EJB3 containers (JEE 5 application servers)

36

JPA Inheritance and Polymorphism



37

JPA Inheritance and Polymorphism

`InheritanceType.SINGLE_TABLE`

One table per type hierarchy
Table contains union of properties of all classes in the hierarchy.

Pros:
Fast access: single row per entity.

Cons:
Adding new subclasses modifies existing table definition.
Cannot use null constraints on subclass fields.

38

JPA Inheritance and Polymorphism

`InheritanceType.JOINED`

One table per class.
Table contains fields from single class.

Pros:
Normalized tables.
Can use null constraint on subclass fields.

Cons:
Retrieval requires a join or multiple fetches.
Save requires multiple writes.

39

JPA

Inheritance and Polymorphism



InheritanceType.TABLE_PER_CLASS

One table per outermost concrete class.
Table contains union of fields from outermost class and it's superclasses.

Pros:
Fast access: single row per entity.
Can use null constraint on subclass fields.

Cons:
Polymorphic references (to superclass) are complex.
Optional in JPA 1.0

40

JPA

Versioning



- Keep database transactions short
 - May lock rows, pages tables
 - Don't wait for user input in a transaction
- Still desirable to detect conflicting updates
 - Application level check
 - Optimistic Offline Lock Pattern
 - Patterns of Enterprise Application Architecture
Martin Fowler 2002
 - Uses a version column in table, managed by application level code

41

JPA

Versioning



- Optimistic Offline Lock Pattern
 - Split UI edit operation into 2 transactions
 - Version # property in entity (integer or timestamp)
 - Transaction 1
 - Return entity to UI layer, including current version #
 - Transaction 2
 - Throw exception if entity version # from UI does not match current version # in database
 - Increment current version # in database

42

JPA Versioning

- Optimistic Offline Lock Pattern
- Used to detect lost updates
 - Lost update: an apparently successful update is overwritten by another transaction
- Lost updates can occur when:
 - Simultaneous processes read/modify/write a value
 - Split transactions for UI input
 - Read/modify/write within a transaction, where transaction isolation level is less than `SERIALIZABLE`
- SQL:
 - update **TABLE** set **VERSION** = *NEW_VALUE* where **VERSION** = *OLD_VALUE* ...
 - Conflicting update has occurred if 0 rows were modified
- JPA:
 - `@Version` tag on a version property of the entity

43

JPA Versioning

- **BEWARE:**
 - Optimistic Offline Lock can fail to detect lost updates if transaction isolation level is higher than `READ_COMMITTED`
 - SQL **UPDATE** would see version column value from start of current transaction, and could fail to detect a conflicting update!
 - JPA assumes `READ_COMMITTED` so `@Version` can work

44

JPA Persistence Context

- Maintains set of entity instances being managed by the persistence engine.
- Each entity represented by a unique object instance.
 - Needed for transaction write-back optimization.
 - Allows database writes to be deferred until end of transaction.
- Scope
 - Transaction
 - Extended

45

JPA Extended Scope Persistence Context

- Pros:
 - Shares cached data across transaction boundaries.
- Cons:
 - Requires synchronization.
 - within JVM
 - across cluster
 - Cache data may be stale if other processes update the database.

46

JPA Detached Entities

- At end of persistence context (i.e. transaction), entity references become POJO's which are no longer managed by the persistence engine.
 - Entities automatically converted into DTO's
- `EntityManager.merge(Object entity)`
 - Integrates changes made outside of the persistence context.
- `EntityManager.persist(Object entity)`
 - Persists new object created outside of the persistence context.

47

JPA Detached Entities

- Entity relationships
 - `FetchType.EAGER`
 - Related entities always fetched with current entity
 - `FetchType.LAZY`
 - Related entities fetched when referenced
 - Automatic fetching can only happen within transaction
 - Automatic fetching outside of a transaction would violate transactional integrity
- What happens if an entity is detached, and a related object that was not yet fetched is referenced?
 - JPA does not specify

48

JPA Detached Entities



- Reference needed entities while still in transaction
- Use JPA Query Language “FETCH JOIN”

49

EJB 3



- JPA for persistence (Entity Beans)
- Session Beans configured with a limited, coarse grained form of Dependency Injection
 - Can only inject objects obtained from JNDI
 - Container managed resources
 - Other EJB's

50

MyEclipse



- Commercial plugins for Eclipse IDE (\$30/year)
- Editing of Spring configuration files with validation and auto-completion of bean ID's, class names, and property names
- Creation of JPA entity classes and Spring DAO's from existing database

EJB
JSP
JSF
Struts
Web Services

XML
HTML
Javascript
UML
Swing

51

Resources



Spring Framework

<http://www.springframework.org/>

MyEclipse

<http://www.myeclipseide.com/>

Spring in Action 2nd Edition

Craig Walls and Ryan Breidenbach August 2007

Enterprise JavaBeans 3.0 5th Edition

Bill Burke and Richard Monson-Haefel May 2006
