Object **Partners** Inc.

Service Layers in Spring

Twin Cities JUG

Panel Discussion

June 2005

Overview

- Recommended service layer is a Spring bean (interface/implementation)
- Panel to cover
  - Spring and Component Based Development
  - Spring and J2EE
  - Spring Remoting and AOP
  - Spring Templates and ORM integration
  - Lots of discussion!!

Presenters

- Casey Bowman
  - President - SojoSoft
- Brian Repko
  - Technical Project Lead – OPI
- Chris Rosenquest
  - Senior Consultant – OPI
- Adam Waldal
  - Senior Consultant/Technical Lead - OPI

About me: Casey Bowman

- Java, since 1996
- SoJoSoft, Inc
  - 1997-1999
- SPS Commerce – Software Architect
  - 1999-2004
- Wells Fargo – Consultant
  - 2004-2005
- SoJoSoft, LLC
  - 2005-
  - Specializing in component-based development
  - Working on a project with Trelligence, Inc., in Houston, TX.

Spring Framework

- BeanFactory – getBean(“beanName”)
  - JavaBean
  - *singleton* or *prototype*
- Dependency Injection
  - Spring populates each bean with other beans
  - bean has setters
  - configuration lies outside bean's “class-space”
  - bean need only know interfaces for other beans
- Application needs an entry point to access these beans
  - Spring MVC – via beanName
  - Standalone application – via ?

Getting a bean

- BeanFactory.getBean(“beanName”)

DANGER!!

- smells of global variables
- dependencies on Spring everywhere!!
- Dependency Injection vs. Lookup
  - use lookup sparingly; use injection primarily
  - injection displays relationships between beans (perhaps even visually!!)
  - injection is flexible, easily editable for testing

What about the “top” bean?

Getting the “top” bean

One strategy...

- Main class calls getBean(“beanName”)
  - Spring dependency restricted to Main
  - Command-line arguments may be used for setting bean properties

What problem might arise?

- The bean tree could get huge and unwieldy.

What do we do before a source code tree gets out of control?

Components

Before a source code tree becomes huge and unwieldy, break the code down into components.

- Limit dependencies.
- Increase abstraction, trustworthiness, and reuse.

Define interface contracts
Unit of deployment: *jar*
Testing
Versioning

Components and Spring

Rod Johnson (2004, June) wrote that he didn't “like to view components as distinct from objects” (J2EE Development without EJB, p. 88) He felt “it's unclear exactly what value viewing applications as built of components adds over viewing them as built of objects” (p. 86).

Rod Johnson (2004, November) may be coming around on this issue according to the email thread “Creating reusable component packages with Spring”.

In the world of Spring beans...

Bean names are floating around.

Smells like global variables.

Why not limit bean names to each component?

That way, other components won't break if there is a bean name conflict or a bean name changes.

Dependencies then can continue to be defined by class (not bean-name) dependencies.

Component Scope Design Pattern (in the making)

- Bean names are component-scoped

Components may then use other components without having to bother about the dependency injection going on inside those other components.

Examples of code aiming to providing Component Scope

- Proprietary code – Summer 2004
  - Simple approach, single classloader, nice vista
  - Wells Fargo would not release to open-source
- Open-source code – Summer 2005 ??
  - Must use a different approach
  - ComponentLoader extending ClassLoader ??
  - Suggestions welcome!

References

- Spring
  - Walls & Breidenbach (2005) *Spring in Action*
  - Tate & Gehtland (2005) *Spring: A Developer's Notebook*
  - Harrop & Machacek (2005) *Pro Spring*
  - Johnson (2004) *J2EE Development without EJB*
- Components
  - Apperly et al. (2003) *Service- and Component-based Development*
  - Szyperski (2002) *Component Software*
  - Cheesman & Daniels (2001) *UML Components*
  - Lakos (1996) *Large-Scale C++ Software Design*

Questions and Answers

Spring and J2EE

- Service Object Architecture
  - Threading and Configuration
- Enterprise Services
  - Transactions, Security, Naming and Management
- Resource Management
  - JDBC, JavaMail and JMS
- Discussion

Service Object Architecture

- Spring recommends Spring Bean as Service object
  - Can support Session EJB as service object if needed
  - Eliminates some of EJB restrictions
  - Can have lifecycle methods and can be factory-aware
- Spring Beans support multiple threading models
  - Multi-threaded objects with no instance variables
  - Multi-threaded objects using synchronized / concurrency
  - Instance-per-Client (ThreadLocal or pooled)
- Spring Beans can make use of dependency injection or lookup for Enterprise Services, Resources or other Business Services
  - So you can test outside the container

Service Object Architecture/2

- ApplicationContext
  - Hierarchical naming service and IoC container
- WebApplicationContext
  - AppContext for a specific WebApplication (ServletContext)
  - May have child AppContexts
- Typical Uses
  - One WebAppContext (w/ or w/o children)
  - Multiple independent WebAppContexts (w/ or w/o children)
  - One AppContext (EAR) with WebAppContexts underneath
- Use built-in WebApplicationContextListener

Enterprise Services

- J2EE exposes JTA UserTransaction, however
  - Lookup is via JNDI and Exceptions are caught but not within same hierarchy
  - Spring is not just for J2EE!
- Transaction support in Spring completely redesigned
  - Not just JTA-based, want the same semantics for local transactions as well
  - Programmatic and Declarative transaction management
- The interfaces…
  - TransactionStatus
    - Wrapper for transaction (global or various local)
  - PlatformTransactionManager
    - Access to TransactionStatus (get, commit and rollback)
    - Implementations based on JTA, DataSource, JDO, Hibernate
  - TransactionDefinition / TransactionAttribute
    - Defines behavior (Required, Supports, etc.), isolation level, timeout
    - Can be optimized (read-only)
  - Transactions and exception handling
    - TransactionAttribute has “rollbackOn(Exception e)”
    - Declarative transactions can define exceptions for commit or rollback
- TransactionTemplate for wrapping up begin/commit/rollback flow
- TransactionInterceptor or Attributes for declarative transactions

Enterprise Services/2

- Security – nothing built in
  - ACEGI Security Framework
  - Data Validation Framework
- Naming – use ApplicationContext
  - Use JndiObjectFactoryBean for real JNDI object
- Management
  - Spring can register bean with JMX (Standard MBean), create a proxy for an MBean or create a bean representing an MBeanServer

Resource Mgmt / JDBC

- Define resource factory (DataSource) as a bean
- Define your DataSource to be either
  - Basic factory (DriverManagerDataSource)
  - Based on connection pool (DBCP, C3PO, XAPool)
  - A JNDI DataSource (full name or resource reference)
- JdbcTemplates do typical JDBC code
  - Methods for batchUpdate, call, execute, query, queryFor*, update
  - Lots of interfaces for Callbacks, Creators and Extractors
- DataAccessExceptions built from SQLExceptions
  - Converts code to subclass
- JDBC package has Objects from ResultSet code
  - Interface21 codebase (pre-Spring)

Resource Mgmt / JavaMail

- Spring works with outbound email only
- Supports both JavaMail and COS mail
  - COS is com.oreilley.servlet – simpler API
- No resource factory – use MailSender or JavaMailSender
  - SimpleMailMessage and MimeMailMessage based respectively
  - Each interface has JavaMail and COS implementations
  - Properties to connect to the SMTP server
- SimpleMailMessage
  - Basic data / Copy Constructor for thread safety
- MimeMessageHelper
  - Used to add embedded images or attachments
  - Used with MimeMessagePreparator to create message
    - Can do HTML and Plain text alternative
    - Can be used with Velocity to do message templates

Resource Mgmt / JMS

- Supports both JMS 1.0.2 and 1.1
- Define resource factory (ConnectionFactory) as a bean
- Can define destinations as beans as well (optional)
  - ActiveMQ supported very well
- JmsTemplate and JmsTemplate102
  - Lots of properties – can also do DestinationResolver
  - Methods for sending and receiving (with selection)
- MessageConverter for Object-Message conversion
  - Methods for convertAndSend and receiveAndConvert
- Working on asynchronous invocation w/o MDB
  - Implemention just uses MessageListener
  - Is single-threaded on a managed thread

Questions and Answers

Spring Framework: AOP - Abilities

- Dynamic AOP – Logic applied at runtime.
- Two Implementations
  - JDK Dynamic Proxy
  - CGLIB Proxy
- Spring applies advice to proxy objects
  - ProxyFactory controls AOP configuration.
  - configure Advisors, configure ProxyFactoryBean, inject Advisors into ProxyFactoryBean
  - Allows Advisor injection with regex

Spring Framework: AOP – Let me give you some advice

- Before
  - Occurs before the joinpoint execution
  - Advisor will implement BeforeAdvice
- Around
  - Occurs before and after method invocation
  - Advisor will implement MethodInterceptor
- After Returning
  - Occurs after returning from the joinpoint execution
  - Advisor will implement AfterReturningAdvice
- Introduction
  - Occurs during invocation of an Object
  - “Introduce” new functionality to an object at runtime
  - Object Locking and Modification Detection
- Throws
  - Occurs after method invocation return but only if that return throws an exception
  - Ability to specify subclass for throws advice

Spring Framework: AOP – Introducing...

- Introduction
  - Adding functionality at runtime.
- Locking
  - Applying the Locking interface to any object
  - Forces immutability
    Object becomes instanceof Lockable (even if the class itself doesn't extend Lockable)
- IsModified
  - Apply an interface to check for member modification.
- mixin the Introduction

Spring Framework: Remote Support

- RMI
  - Basic plumbing for RMI support
  - RemotingException handling and Remote interface support
- HTTP Invocation
  - Native to the framework
  - Secure remote services with HTTP authentication
- JAXRPC
  - Integration with Axis
- Hessian/Burlap
  - Caucho implementation creating web services
  - binary/XML

Spring Framework: Exposing your... services

- Use the Dispatcher Servlet to configure remote application access
- RMI Object (Spring imbedded)
  - Define services in remote-servlet.xml (remote BeanFactory)
  - Expose as RMI object with RmiServiceExporter
    - Name
    - Service
    - Interface
    - RegisteryPort
- Web Services
  - JAX-RPC extending ServletEndpointSupport and implementing service interface
  - Axis WebService = 4 parts
    - Servlet
    - Deployment descriptor
    - Remote interface
    - Service implementation
  - SOAP message processing handled behind the curtains
  - Auto-detection is NOT implemented for security

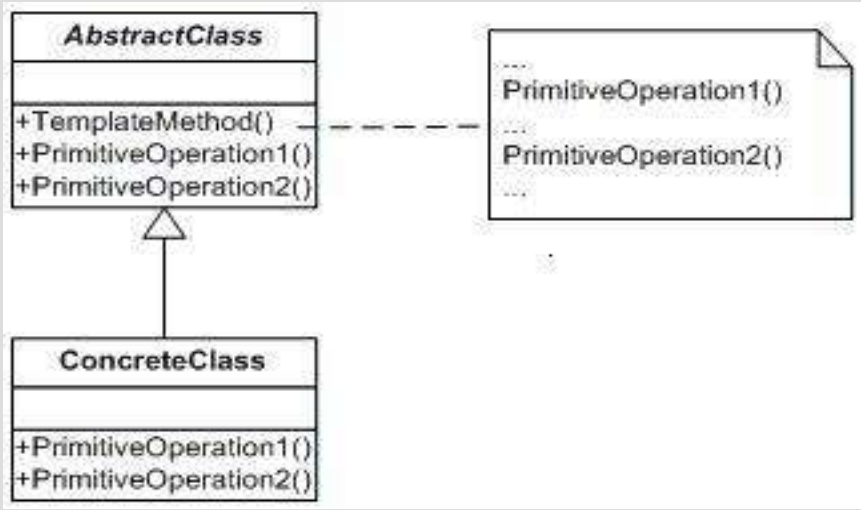
Questions and Answers

Use of Template Method Pattern in Spring/Hibernate and Modules

Template Method
Spring with Hibernate
Spring Modules

Template Methods

- “A template method defines the skeleton of an algorithm in terms of abstract operations which subclasses override to provide concrete behaviour.”
  - http://en.wikipedia.org/wiki/Template_method_pattern
  - http://www.dofactory.com/Patterns/PatternTemplate.aspx



Spring DAO Templates

JDBCTemplate
- execute() method templates all connection management code

SQLMapClientTemplate
- Custom methods for using iBATIS SqlMaps
- queryForXXX(), insert(), update(), delete()

HibernateTemplate
- doInHibernate() executes statements in a Hibernate session.

- All of these Template objects encapsulate the infrastructure code necessary to use the database, with the datasource injected.
- It also gives easy extension for use of converters and callbackHandlers for dealing with results and exceptions using Spring.

Spring has 2 ways to work with Hibernate

- HibernateTemplate
  - Use Spring's HibernateTemplate to gain access to Hibernate's SessionFactory and Session
- Hibernate Interception
  - Use Spring AOP to deal with the SessionFactory and Session using before and after or around advice.
- My recommendation is use the HibernateTemplate, because it is well thought out, and takes care of most uses for you. If you need more the AOP solution is there and can be done if necessary, but DAO code will have to change.

HibernateTemplate

```
public class HibernateClinic extends HibernateDaoSupport implements Clinic {

  public Collection getVets() throws DataAccessException {
    return getHibernateTemplate().find("from Vet vet order by vet.lastName, vet.firstName");
  }
  public Collection getPetTypes() throws DataAccessException {
    return getHibernateTemplate().find("from PetType type order by type.name");
  }
  public Collection findOwners(String lastName) throws DataAccessException {
    return getHibernateTemplate().find("from Owner owner where owner.lastName like ?", lastName +
      "%");
  }
  public Pet loadPet(int id) throws DataAccessException {
    return (Pet) getHibernateTemplate().load(Pet.class, new Integer(id));
  }
  public void storeOwner(Owner owner) throws DataAccessException {
    getHibernateTemplate().saveOrUpdate(owner);
  }
...
}
```

- Spring Petclinic Hibernate Sample

Other Templates in Spring

All these template exhibit the same structure and use a Callback interface for extending base behavior.
(Spring Distribution)

- JNDITemplate
- JMSTemplate
- TransactionTemplate

(SpringModules)

- OSWorkflowTemplate
- JSR94Template
- LuceneTemplate

Questions and Answers