

Spring Framework Overview (using MyPetstore)

MSIS 531 – Spring 2006

Some slides from a presentation to
the Chicago Java Users Group by
Ron Lamber (Shawk, Inc.)

2

What is the Spring Framework?

- Spring is a Lightweight **Application** Framework
- Where JSF Struts, WebWork and others can be considered **Web** frameworks, Spring addresses all tiers of an application
- Spring provides the plumbing so that you don't have to!
- Spring is NOT a J2EE application server
 - JBoss, WebLogic, WebSphere
 - Spring can integrate nicely with J2EE application servers (or any Java environment)
 - Spring can, in many cases, elegantly replace services traditionally provided by J2EE application servers

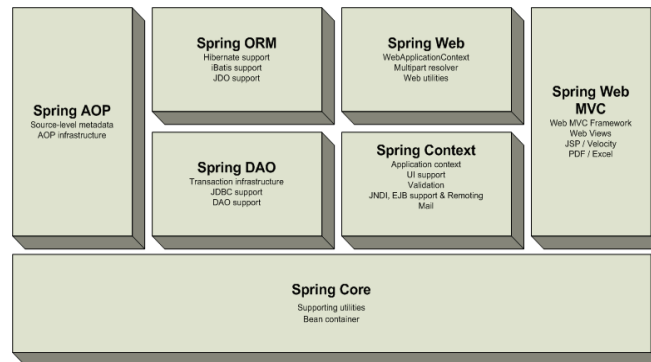
Spring Benefits

- Spring brings a consistent structure to your entire application
- Spring provides a consistent way to glue applications together
- Spring provides integration points with standard and defacto-standard interfaces: Hibernate, JDO, TopLink, EJB, RMI, JNDI, JMS, Web Services, Struts
 - Huge productivity gains possible through not having to write the common integration points

The Spring Framework Mission Statement

- J2EE should be easier to use
- It's best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero
- JavaBeans offer a great way of configuring applications
- OO design is more important than any implementation technology, such as J2EE
- Checked exceptions are overused. A framework shouldn't force you to catch

Spring Overview



Note: Spring comes as one big jar file and alternatively as a series of smaller jars broken out along the above lines (so you can include only what you need)

Aside: Ant and JUnit

- Two very nice, open source tools to aid in Java development and testing:
 1. **Ant** (ant.apache.org): Java build tool, created out of frustration with existing tools (e.g. make)
 - Used everywhere, for everything (and note this is not always a good thing)
 2. **JUnit** (JUnit.org): Testing framework
 - Outcome of so-called “test-driven development”: we’re done coding when all tests pass, and tests get written before code
- We’ve used Ant all along (in Netbeans), but the builds are now separated out, and JUnit is in the same tool

Spring is Non-Invasive

What does that mean?

- You are not forced to import or extend any Spring APIs
- An invasive API takes over your code.
- Anti-patterns:
 - EJB forces you to use JNDI
 - Struts forces you to extend **Action**

Invasive frameworks are inherently difficult to test. You have to stub the *runtime* that is supplied by the application server

Example: Ch. 1 Lab 1

Spring Services

At its core, Spring provides:

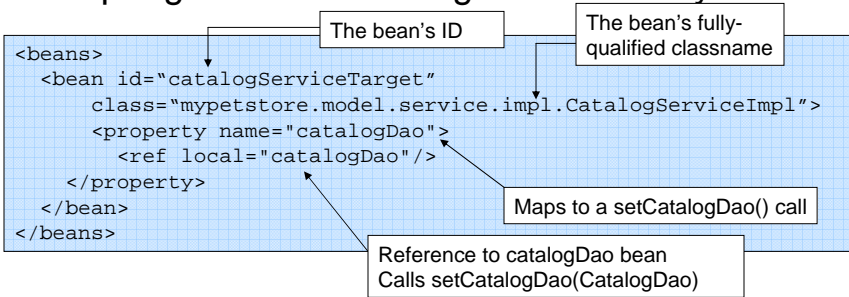
- An Inversion of Control Container
 - Also known as Dependency Injection
- An AOP Framework
 - Spring provides a proxy-based AOP framework
 - Integrates with AspectJ or AspectWerkz
- A Service Abstraction Layer
 - Consistent integration with standard, 3rd party APIs
- In summary: a framework for “wiring up” (connecting) your entire application
 - Key component: **Bean Factories**

BeanFactories

- A **BeanFactory** is typically configured in an XML file with the root element: <beans>
- The XML contains one or more <bean> elements
 - id (or name) attribute to identify the bean
 - class attribute to specify the fully qualified class

BeanFactories

- The magic of Spring (and IOC) is below
- Beans are automatically instantiated and referenced to other beans (“wiring”)
- The Spring container manages all this for you



Dependency Injection (Inversion of Control)

- Complicated sounding terms for a fairly simple concept
- The Hollywood Principle: “Don’t call me, I’ll call you”
- Dependencies used from within a bean aren’t asked for outwardly, but are injected into the bean by the container
- Eliminates lookup code from within your application
- Allows for pluggability and hot swapping
- Promotes good OO design
- Enables reuse of existing code
- Makes your application extremely testable

Example: Ch. 1 Lab 4

The Spring ApplicationContext

- An **ApplicationContext** is a **BeanFactory**, but adds “framework” features such as:
 - I18n (localization) messages
 - Event notifications
- Essentially a registry of beans available in the current context (web or other)
- For MyPetstore, the context supplies Hibernate services
 - Created as a servlet in web.xml
 - Available anywhere in web application as a result
 - Accessed in mypetstore.view.bean.ServiceLocatorBean
 - Few MyPetstore classes (and no JSFs) know about Spring

AOP (Aspect-Oriented Programming)

- AOP decomposes a system into **concerns**, instead of objects
 - Concern: something that you are “concerned” about
- Deals with "aspects" that **cross-cut** across the code and can be difficult to modularize with OOP
- Common example: logging
 - Logging code typically scattered all over a system
 - With AOP, you can declare, for example, that a system should write a log record at the beginning and end of all method invocations
- Other examples: system security, database transactions
 - Spring delivers logging, security, declarative transactions; others can be added by developers

Service Abstraction Layers

Spring provides abstraction for:

- Transaction Management
 - JTA, JDBC, others
- Data Access
 - JDBC, Hibernate, JDO, TopLink, iBatis
- Email
- Remoting
 - EJB, Web Services, RMI, Hessian/Burlap

Service Abstraction Layers

Benefits:

- No implicit contracts (connections) with JNDI, etc.
- Insulates you from the underlying APIs
- Greater reusability
- Spring abstractions are always interfaces
 - Makes testing simpler
- For data access, Spring uses a generic transaction infrastructure and DAO exception hierarchy that is common across all supported platforms

Spring on the Web Tier (Spring MVC)

- Spring integrates with JSF, Tapestry, Velocity, and other web frameworks
- Spring provides a web framework, Spring MVC
 - Simple interface-based infrastructure for handling web-based interaction
 - MVC components are first-class Spring beans
 - Other Spring beans can be injected into Spring MVC components
 - Easy to test

Example: Ch. 2 Lab 3

MyPetstore – Deep Dive

- We'll use the development notes from Session 4 to take a closer look at what's going on
 - Explicit goal: kick the tires
 - Implicit goal: treasure hunt for A2 answers
- Primary use for Spring in MyPetstore is as a glue layer between JSF and Hibernate
 - Design goal was to emulate J2EE Petstore app, a “see all the nice things we can do” application
 - Possibly more complicated than needed