An Introduction to the Spring Framework

What is the Spring Framework?

- Spring is a Lightweight ***Application*** Framework
- Where 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 == J2EE Application Server?

- Spring is NOT a J2EE application server
- 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

Before Struts

- Before Struts, everyone wrote their own front controllers or put their controller logic in JSP
- After Struts, the custom front controllers could be thrown out
  - Developers focus on solving business problems
  - Productivity Gain!
- But with Struts (and most of the other web frameworks) you still have to write your own business delegates or service layers…

Spring Can Help!

- Spring brings a consistent structure to your entire application
- Spring provides a consistent way to glue your whole application together
- Spring provides elegant integration points with standard interfaces
- Hibernate
- JDO
- TopLink
- EJB
- RMI
- JNDI
- JMS
- Web Services
- Struts etc.

Cont

- Just as Struts did on the web tier, we can realize huge productivity gains by not having to write the common integration points across your application

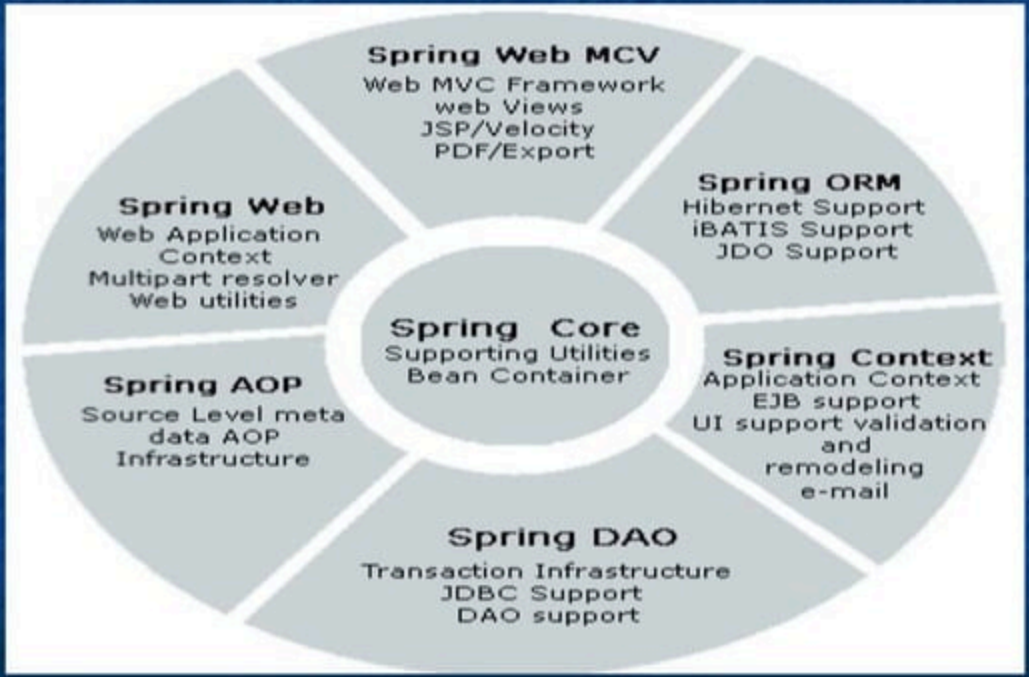
The Spring Framework Mission Statement

- 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.
- Testability is essential, and a framework such as Spring should help make your code easier to test.

Spring Framework Mission Statement (continued)

- Your application code should **not** depend on Spring APIs
- Spring should not compete with good existing solutions, but Used for integration. (For example, JDO and Hibernate are great O/R mapping solutions. We don't need to develop another one.)

Spring Architecture



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**

Spring Core

At it's core, Spring provides:

An Inversion of Control Container

- Also known as Dependency Injection
  - Setter Injection (Injection via Java Bean Setters)
  - Constructor Injection (Injection via Constructor arguments)

An AOP Framework

- Spring provides a proxy-based AOP framework
- You can alternatively integrate with AspectJ or AspectWerkz

- A Service Abstraction Layer
  - Consistent integration with various standard and 3rd party APIs

These together enable you to write powerful, scalable applications using POJOs.

Spring Core (Continue)

- Spring at it's core, is a framework for wiring up your entire application
- **BeanFactories** are the heart of Spring

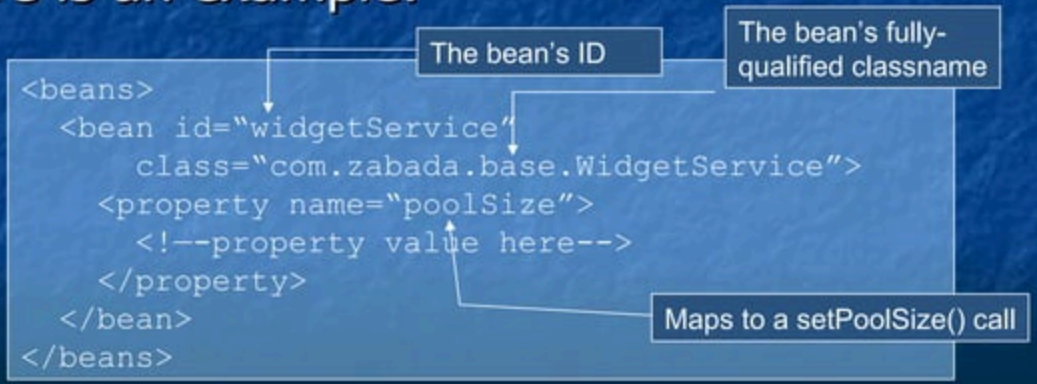
BeanFactories

- 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

- By default, beans are treated as singletons
- Can also be prototypes

Here is an example:



Property Values for BeanFactories

- Strings and Numbers

```
<property name="size"><value>42</value></property>
```

```
<property name="name"><value>Jim</value></property>
```

- Arrays and Collections

```
<property name="hobbies">
  <list>
    <value>Basket Weaving</value>
    <value>Break Dancing</value>
  </list>
</property>
```

Property Values for BeanFactories (continued)

The real magic comes in when you can set

a property on a bean that refers to another bean in the configuration.

```
<bean name="widgetService" class="com.zabada.base.WidgetServiceImpl">
  <property name="widgetDAO">
    <ref bean="myWidgetDAO"/>
  </property>
</bean>
```

calls **setWidgetDAO(myWidgetDAO)** where **myWidgetDAO** is another bean defined in the configuration

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

Dependency Injection (Inversion of Control)

- Eliminates lookup code from within your application
- Allows for pluggablity and hot swapping
- Promotes good OO design
- Enables reuse of existing code
- Makes your application extremely testable

A Very Special BeanFactory: the ApplicationContext

- An **ApplicationContext** is a **BeanFactory**, but adds "framework" features such as:
  - i18n messages
  - Event notifications
- This is what you will probably most often use in your Spring applications

AOP (Aspect-Oriented Programming)

- AOP decomposes a system into concerns, instead of objects.
- Deals with "aspects" that **cross-cut** across the code and can be difficult or impossible to modularize with OOP
- The most common example given is logging
  - Code for doing logging typically must be 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.

AOP (Aspect-Oriented Programming)

AOP enables the delivery of services to POJOs

- Spring provides pre-packaged AOP services:
  - Declarative Transaction Management
  - Security
  - Logging
- You can write custom AOP services for:
  - Auditing
  - Caching
  - Custom security

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 with JNDI, etc.
- Insulates you from the underlying APIs
- Greater reusability
- Spring abstractions always consist of interfaces
- This 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 integrates nicely with Struts, WebWork, JSF, Tapestry, Velocity and other web frameworks
- Spring also provides it's own web framework, Spring Web MVC

Spring on the Web Tier – Spring MVC

- MVC web application framework built on core Spring functionality
- MVC dispatcher framework
- is highly configurable via strategy interfaces and accommodates multiple view technologies Like
- JSP, Tiles, Velocity, FreeMarker, iText
- MVC comes in a Servlet edition working with the underlying environment

Spring on the Web Tier – Spring MVC (Cont)

- The Spring MVC Framework offers a simple interface based infrastructure for handing web MVC architectures
- Spring MVC components are treated as first-class Spring beans
  - Other Spring beans can easily be injected into Spring MVC components
  - Spring MVC components are easy to test

Spring MVC – Key Interfaces

- **Controller** (`org.springframework.web.servlet.mvc.Controller`)
  - Must implement `ModelAndView handleRequest(request,response)`
  - *This is the base controller interface, comparable to the notion of a Struts Action.*
- **View** (`org.springframework.web.servlet.mvc.View`)
  - Must implement `void render( model, request, response)`
  - *This is the MVC view for a web interaction. Implementations are responsible for rendering content, and exposing the model.*
- **Model**
  - To complete the MVC trio, note that the model is typically handled as a `java.util.Map` which is returned with the view
  - the values of the model are available, for example in a JSP, using a `<jsp:useBean/>` where the **id** corresponds to the key value in the Map

Spring on the Web Tier: Integration with Other Frameworks

- Spring integrates nicely with other web frameworks with two methodologies:
  - Look up Spring beans within Controllers/Actions via the convenience static method:
    ```
    WebApplicationContextUtils.getWebApplicationContext( servletContext)
    .getBean("beanName")
    ```
  - Configure the Controllers/Actions for the web framework in a Spring BeanFactory and then use Spring provided proxies in the actual web framework configuration
    - When available, this methodology is preferred
    - This approach lets you design your Controllers/Actions with dependency injection and makes your Controller/Actions more testable

Spring ORM

- ORM Module in Spring Relates With the Database access
- Provides Integration Layer for Popular Object Relating Mapping API's including JDO , Hibernate etc

Spring DAO

- Spring DAO (Data access object) module supports for standardizing data access work using Technologies like
  JDBC
  Hibernate
  JDO, etc