New in the World of Spring:

A Look at Spring 2.0 and Related Projects

Colin Sampaleanu
Interface21

Spring

About Me

- Spring Framework core developer since mid-2003

- Founder and Principal Consultant at Interface21, a unique consultancy devoted to Spring Framework and Java EE
  - Training, consulting and support
  - "From the Source"
  - http://www.interface21.com

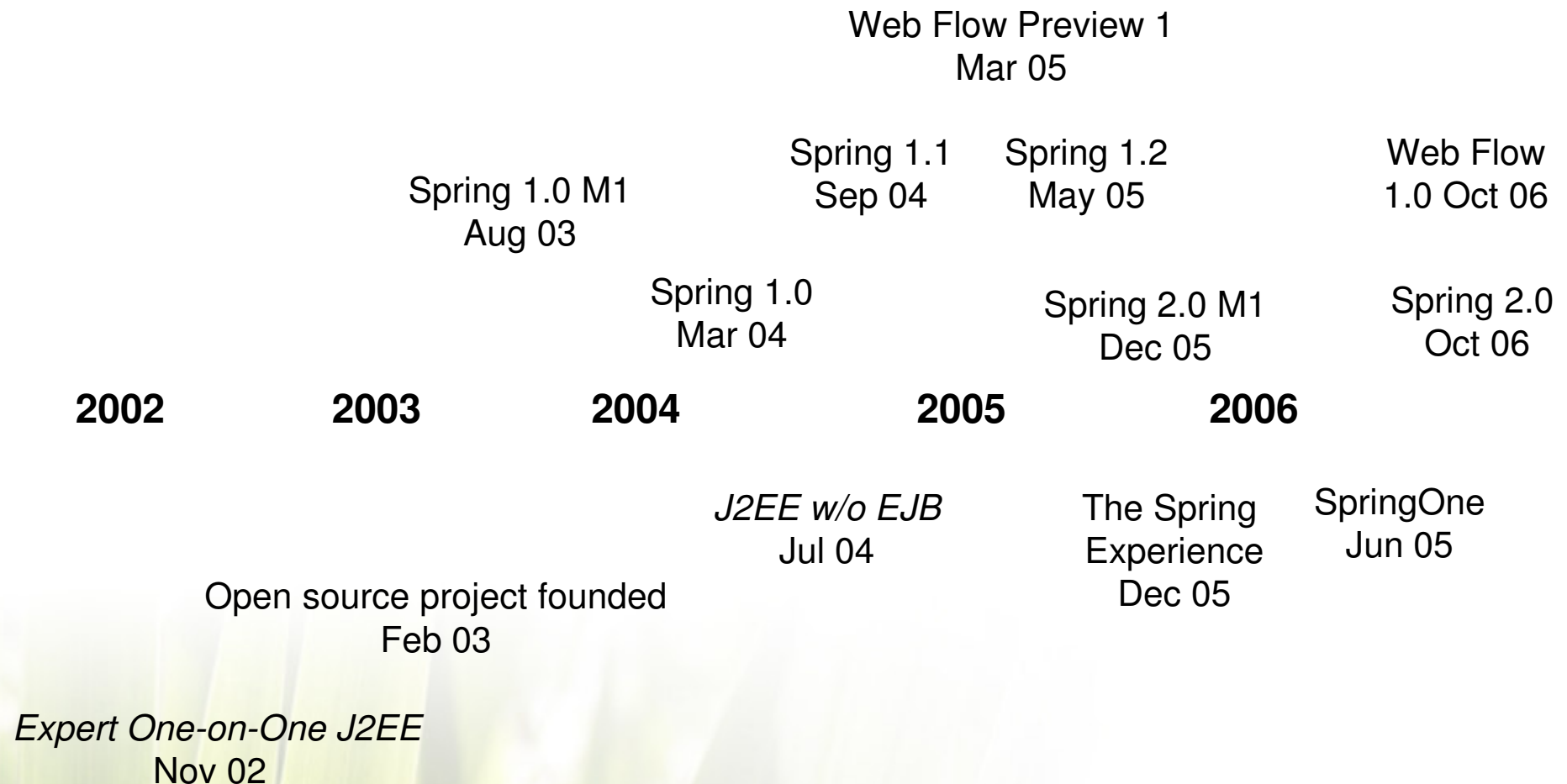# Where is Spring Today?

Interface21

Spring

Where is Spring today?

- **The favourite and most trusted Java application framework**

- **Most powerful technology for enabling POJO-based application development**

- Widely adopted across most industries and proven in many demanding applications

How did we get here?

Web Flow Preview 1
Mar 05

Spring 1.1   Spring 1.2   Web Flow
Spring 1.0 M1   Sep 04   May 05   1.0 Oct 06
Aug 03

Spring 1.0   Spring 2.0 M1   Spring 2.0
Mar 04   Dec 05   Oct 06

2002   **2003**   **2004**   **2005**   **2006**

J2EE w/o EJB   The Spring   SpringOne
Jul 04   Experience   Jun 05
Dec 05

Open source project founded
Feb 03

Expert One-on-One J2EE
Nov 02

Spring

Who's using Spring?

- Everybody! - Spring has become ubiquitous for Java application development
  - Very well established in the enterprise space, including:
    - Banking / Financials, Telecom, Insurance, Government, Airline industry, Defence
  - The middle tier companies
  - …all the way down to mom & pop development shops

- It's known & trusted
- It's used
- It adds value

- Analyst conclusion (Forrester – *Health of Open Source*)
  - *A majority of [enterprise Java] users interviewed by Forrester use Spring*

Spring

Sample enterprise user: Voca

- Part of UK's Critical National Infrastructure
  - They process Direct Debits, Direct Credits and Standing Orders to move money between banks
  - Over 5 billion transactions worth €4.5 trillion in 2005
  - Some 15% of Europe's Direct Debits and Direct Credits are handled by Voca
  - Over 70% of the UK population use Direct Debits to pay household bills; Direct Credits are used to pay over 90% of UK salaries
  - Over 72 million items on a peak day
  - They have never lost a payment

- Have rolled out Spring-based implementation of this infrastructure to replace legacy version!
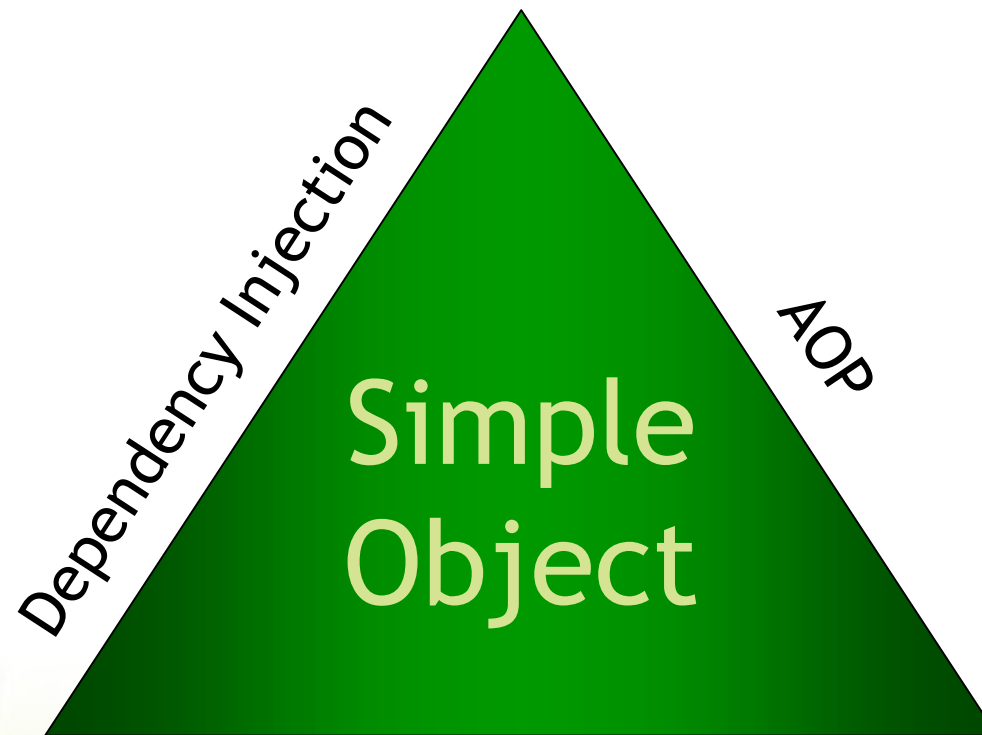
Spring 2.0 Preview

Interface21

Spring

Spring has always delivered great value based on 3 enabling technologies…

… working together to enable POJO-based software development to be agile, fun, and productive

… there's a bit of magic here, it's adds up to more than the sum of the parts

Enabling technologies

Dependency Injection

AOP

Simple Object

Portable Service Abstractions

INTERFACE21

Spring

Spring 2.0 – taking it to the next level

- Builds on the solid base established by Spring 1.x
- Pursues vision of POJO-based development
- Adds new capabilities and makes many tasks more elegant
  - make Spring more **powerful**
  - … while **simplifying** common tasks

- … with full backwards compatibility

Spring 2.0: Backwards compatible

- Huge user base demands full backwards compatibility
  - Old code still works

- Runs on existing infrastructure
  - Java 1.3, Java 1.4, Java 5
  - A myriad of Java EE app server versions, old and new (WebSphere, WebLogic, JBoss, OC4J, etc.).
  - Simple Servlet engines (TomCat, Resin, etc.)
  - Standalone app (Swing/RCP) environments

Spring

Spring 2.0: New Features (1)

- ## Major enhancements and new features across the board, especially…
  - Simpler, more extensible XML configuration
  - Enhanced AOP functionality and integration with AspectJ
    - Leveraging new XML config

Spring

Spring 2.0: New Features (2)

- Additional scoping options for beans
  - Backed by HttpSession etc.
  - Pluggable backing store
    - Not tied to web tier
- Ability to define any named bean in a scripting language such as Groovy or JRuby
  - Named bean conceals both configuration and implementation language
  - Allows for DI, AOP and dynamic reloading

- JPA (Java Persistence Architecture) Integration
- JdbcTemplate simplification for Java 5

Spring 2.0 Enhancements

- MVC Simplification: Intelligent defaulting
- New JSP form tags
- Spring Portlet MVC, an MVC framework for JSR-168 Portlets

- Asynchronous JMS facilities enabling *message-driven POJOs*
- Customizable task execution framework for asynchronous task execution
- CommonJ TimerManager implementation
  - Great for WebLogic/Websphere users

XML

XML Configuration in Spring 2.0

- Ability to define new XML tags to produce one or more Spring bean definitions

- Tags out of the box for common configuration tasks

- Problem-specific configuration
  - Easier to write and to maintain

- XML schema validation
  - Better out of the box tool support
  - Code completion for free

- Exploits the full power of XML
  - Namespaces, schema, tooling

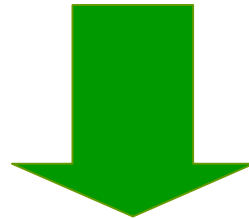# XML Configuration in Spring 2.0

- Backward compatibility
  - Full support for <beans> DTD
  - Complete interoperability between classic and extended configuration

- Choices
  - Flexible, generic configuration
  - Targeted, problem-specific configuration

- Ideal for third parties developing Spring components, and very large projects

Spring

XML Configuration in Spring 2.0

A JNDI lookup...

```
<bean id="dataSource" class="...JndiObjectFactoryBean">
  <property name="jndiName" value="jdbc/StockData"/>
  <property name="resourceRef" value="true"/>
</bean>
```

```
<jee:jndi-lookup id="dataSource"
    jndiName="jdbc/StockData" resourceRef="true"/>
```

Code Completion in IDE

Spring

XML Configuration in Spring 2.0

Loading a properties file…

```
<bean id="properties" class="...PropertiesFactoryBean">
  <property name="location" value="jdbc.properties"/>
</bean>
```



```
<util:properties id="properties"
                 location="jdbc.properties"/>
```

Out of the box namespaces

- `<jee:*/>`
  - JEE related configuration
- `<util:*/>`
  - Load Properties instances, create constants, etc.
- `<lang:*/>`
  - Dynamic scripting support
- `<aop:*/>`
  - Simplified standard AOP configuration
  - Expose new AspectJ-style advice
- `<tx:*/>`
  - Transaction simplification - DSL for concise definitions
  - Improved tool support (code-asist) for transactional advice
  - Enable annotation-driven transactions in a single line
    - `<tx:annotation-driven />`

Authoring Custom Extensions: Step 1

- Write an XSD to define element content
  - Allows sophisticated validation, well beyond DTD
  - Amenable to tool support during development
  - Author with XML tools
    - XML Spy

Spring

Authoring Custom Extensions: Step 2

- Implement a NamespaceHandler to generate Spring BeanDefinitions from element content
- Helper classes such as BeanDefinitionBuilder to make this easy

```java
public interface NamespaceHandler {

    BeanDefinitionParser findParserForElement(
                                Element element);
    BeanDefinitionDecorator findDecoratorForElement(
                                Element element);
}
```

```java
public interface BeanDefinitionParser {

    void parse(Element element,
               BeanDefinitionRegistry registry);
}
```

Authoring Custom Extensions: Step 3

- Add a mapping in META-INF/spring.handlers
- Can add or hide handlers

```
http\://www.springframework.org/schema/util=org.springfram
    ework.beans.factory.xml.UtilNamespaceHandler
http\://www.springframework.org/schema/aop=org.springframe
    work.aop.config.AopNamespaceHandler
http\://www.springframework.org/schema/jndi=org.springfram
    ework.jndi.config.JndiNamespaceHandler
http\://www.springframework.org/schema/tx=org.springframew
    ork.transaction.config.TxNamespaceHandler
http\://www.springframework.org/schema/mvc=org.springframe
    work.web.servlet.config.MvcNamespaceHandler
```

Using custom extensions

- Import relevant XSD
- Use the new elements

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"

    xsi:schemaLocation="http://www.springframework.org/schema
/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-
aop.xsd">
```

XML Configuration Best Practices

- ## Standard `<bean>` tags
  - Still a great solution
  - General configuration tasks
  - Application-specific components
    - DAOs, Services, Web Tier

- ## Custom tags
  - Infrastructure tasks
    - JNDI, Properties, AOP, Transactions
    - Supplied with Spring
  - 3rd party namespaces
    - Whatever **YOU** or **YOUR USERS** need

AOP

AOP in Spring 2.0

- AOP is important…
  - How do we make Spring AOP better?

- Simplified XML configuration using <aop:*/> tags
- Closer AspectJ integration
  - Pointcut expression language
  - AspectJ-style aspects in Spring AOP
  - @AspectJ-style aspects in Spring AOP
    - Fully interoperable with ajc compiled aspects

- Spring ships with AspectJ aspects for Spring/AspectJ users
  - Dependency injection on any object even if it isn't constructed by the Spring IoC container

Spring 2.0 aims for Spring AOP

- **Build on strengths, eliminate weaknesses**
- Preserve ease of adoption
  - Still zero impact on development process, deployment
  - Easier to adopt
- Benefit from the power of AspectJ
- Provide a comprehensive AOP roadmap for Spring users, spanning
  - Spring AOP
  - AspectJ

- Remains fully backwards compatible with Spring 1.x releases

Spring 2.0 and AspectJ

- Spring and AspectJ are still distinct projects

- Spring just uses the AspectJ pointcut parsing and matching APIs
  - using AspectJ as a library, not as a weaving engine

- Gives the same syntax and semantics across Spring AOP and AspectJ
  - perfect if you are going to use both
  - or start out with Spring AOP, and then want to introduce AspectJ at some point

Spring

Pointcut Expressions

- Spring can use AspectJ pointcut expressions
  - In Spring XML
  - In @AspectJ aspects
  - In Java code (with Spring `ProxyFactory`)
- New `AspectJExpressionPointcut` will become the most used Spring AOP Pointcut implementation

Spring

What's so good about AspectJ pointcut expressions?

- Go far beyond simple wildcarding
- AspectJ views pointcuts as first-class language constructs
  - Can compose pointcuts into expressions
  - Can reference named pointcuts, enabling reuse
  - Can perform argument binding…
  - Can express complex matching logic *concisely*

- Well documented in many books/articles

Spring

AOP is *about* pointcuts

- Pointcuts give us the tool to think about program structure in a different way to OOP

- Without a true pointcut model we have only trivial interception
  - Does not achieve aim of modularizing crosscutting logic
  - **DRY (Don't repeat yourself)**

- Spring AOP has always had true pointcuts
  - But now they are dramatically improved

@AspectJ-style Aspects

```java
@Aspect
public class MyLoggingAspect {

    @Pointcut("execution(* *..Account.*(..))")
    public void callsToAccount(){}


    @Before("callsToAccount()")
    public void before(JoinPoint jp) {
        System.out.println("Before [" +
                        jp.toShortString() + "].");
    }


    @AfterReturning("callsToAccount()")
    public void after() {
        System.out.println("After.");
    }
}
```

@AspectJ-style Aspects

<!– tell spring-aop to treat any beans that are @AspectJ aspects as such -->

`<aop:aspectj-autoproxy/>`

`<bean id="account" class="demo.Account"/>`

<!– define a bean that is an @AspectJ aspect, DI as normal… -->

`<bean id="aspect" class="demo.ataspectj.AjLoggingAspect"/>`

Argument binding for convenience and type safety

```
@Aspect
public class MakeLockable {

    @DeclareParents(value = "org.springframework..*",
                defaultImpl=DefaultLockable.class)
    public static Lockable mixin;

    @Before(value="execution(void set*(*)) && this(mixin)",
                                    argNames="mixin")
    public void checkNotLocked(
        Lockable mixin)    // Bind to arg
    {
        if (mixin.locked()) {
            throw new IllegalStateException();
        }
    }

}
```

POJO Methods as Advice

```
public class JavaBeanPropertyMonitor {

   private int getterCount = 0;
   private int setterCount = 0;

   public void beforeGetter() {
      this.getterCount++;
   }

   public void afterSetter() {
      this.setterCount++;
   }

   ...
```

Spring

Applying Pointcuts: Via XML

INTERFACE21

```xml
<aop:config>
  <aop:aspect bean="javaBeanMonitor">
    <aop:before
        pointcut=
          "execution(public !void get*())"
        method="beforeGetter"/>
    <aop:after
        pointcut=
          "execution(public void set*(*))"
        method="afterSetter"/>
  </aop:aspect>
</aop:config>
<!- just a regular bean -->
<bean id="javaBeanMonitor" class="..."/>
```

Spring

Code Break

- Let's contrast this to the old approach…

Use of Old Interceptors

```
<aop:config>
  <aop:advisor advice-ref="advice"
          pointcut="execution(* *..ITestBean.*(..))"/>
</aop:config>

<bean id="advice"
    class="org.springframework.aop...DebugInterceptor"/>
```

Domain Objects and DI

- What if a domain object needs access to a business service or DAO?

  - how does it get hold of a reference?

- A lookup introduces unwanted coupling

  - use dependency injection

- But domain objects are created outside of Spring's control???

In the Domain Object (1)

```
@Configurable("accountBean")
public class Account {
  private TransferService tService;

  public void setTransferService(
    TransferService aService) {
    this.tService = aService();
  }
  …
}

Account acc = new Account(…);
```

In the Domain Object (2)

```
@Configurable(autowire=Autowire.BY_TYPE)
public class Account {
  private TransferService tService;

  public void setTransferService(
    TransferService aService) {
    this.tService = aService();
  }
  …
}

Account acc = new Account(…);
```
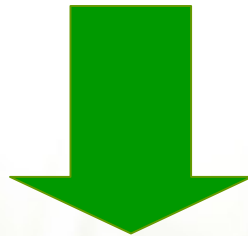
Configuring the aspect

- Simple aspect config allows Spring to configure the domain object immediately after construction

```
<bean class="..beans.factory.aspectj.BeanConfigurer"
       factory-method="aspectOf"/>
```

<aop:spring-configured />

"But I don't want an annotation in my classes..."

- Introducing framework annotations is a questionable practice in core business objects
- Often just not possible if you have existing code
- Solution
  - Just use an AspectJ pointcut
  - AspectJ is ideal for capturing concepts such as any method in a domain object
  - execution(*
    com.mycompany.domain..*.*(..))

Other Container Enhancements

Bean Scopes

- Spring 1.2 supports two scopes for beans managed by Spring, configured via the `singleton` attribute
  - singleton=true
    - Singleton scope
    - Only one instance of the bean ever created
    - Managed by container even after initialization
  - singleton=false
    - Prototype scope
    - Spring creates new bean each time it's asked for or injected
    - Not managed by the container after initialization

- Spring 2.0 adds additional built-in scopes, and user extensible scoping

Bean Scopes

- Scopes in Spring 2.0:
  - singleton (same as old singleton="true")
  - prototype (same as old singleton="false")
  - request
  - session
  - global session

- Add your own custom scope if needed…

- AOP optionally adds dynamic lookups

```
<bean id="requestScoped1" scope="request" class="…TestBean">
  <property name="name" value="Rob Harrop"/>
</bean>
<bean id="requestScoped2" scope="request" class="…TestBean">
  <aop:scoped-proxy/>
  <property name="name" value="Rob Harrop"/>
</bean>
```

Spring

@Required annotation

- @Required annotation may eliminate the need for init methods
  - Requires Java 5
  - Framework specific annotation in your code!

```
Public class EmailService {
    @Required
    public setAdminAdress(String addr) {
        this.adminAddress = addr;
    }
}
```

Dynamic Scripting Support

```xml
<lang:groovy id="messenger"
    refresh-check-delay="5000" <!-- switches refreshing on with 5
 seconds between checks -->
    script-source="classpath:Messenger.groovy">
  <lang:property name="message" value="I Can Do The Frug" />
</lang:groovy>

<bean id="bookingService" class="x.y.DefaultBookingService">
  <property name="messenger" ref="messenger" />
</bean>
```

Spring

Dynamic Scripting Support (2)

```xml
<lang:groovy id="messenger">
   <lang:property name="message" value="I Can Do The Frug" />
   <lang:inline-script>
package org.springframework.scripting.groovy;

import org.springframework.scripting.Messenger

class GroovyMessenger implements Messenger {

   @Property String message;
}
   </lang:inline-script>
</lang:groovy>
```

Utility

Task Executor Abstraction (1)

- Central strategy interface: TaskExecutor
  - for submitting asynchronous work
  - execute(Runnable)
- SimpleAsyncTaskExecutor
  - simple asynchronous execution in new Thread
- ThreadPoolTaskExecutor
  - uses the Java 5 ThreadPoolExecutor
- TimerTaskExecutor
  - uses the JDK Timer
- WorkManagerTaskExecutor
  - uses CommonJ WorkManager from JNDI

Spring

Task Executor Abstraction (2)

- **CommonJ WorkManager**
  - joint BEA/IBM specification
  - supported by WebLogic 9 & WebSphere 6

- **WorkManager available as JNDI resource**
  - configured through administration console
  - fully managed application server threads!

- **Delegate custom asynchronous work to the application server**
  - no custom thread creation!
  - properly managed application server threads
  - closes important gap in J2EE specification

Task Executor Abstraction (3)

- TaskExecutor is available for direct use in application code
  - simply receive a TaskExecutor via dependency injection
  - pass custom Runnables to it as needed
- TaskExecutor is also used within the framework
  - execution of application event listeners
    - configurable ApplicationEventMulticaster
    - synchronous or asynchronous
  - asynchronous reception of JMS messages
    - message listener loops execute as scheduled tasks
    - can participate in shared thread pool

Middle Tier

Code Break – transactions the old way

- Let's see how Spring 1.2 handles transactional wrapping via XML and annotations…

INTERFACE21

Spring

New Transaction Configuration: XML

Using Spring 2.0's concise "tx" namespace:

```
<aop:config>
   <aop:advisor pointcut="execution(* *..OrderService+.*(..))"
                advice-ref="txAdvice"/>
</aop:config>


<tx:advice id="txAdvice">
   <tx:attributes>
      <tx:method name="get*"/>
      <tx:method name="process*" propagation="REQUIRES_NEW"/>
   </tx:attributes>
</tx:advice>

<bean id="transactionManager"
    class="org.springframework.transaction.jta.JtaTransactionManager"/>

<bean id="orderService" class="foo.OrderServiceImpl"/>
```

More Powerful

More Concise

Less Chance of Error (Code-Completion in IDE)

Transaction configuration via Annotations

Code example:

```
@Transactional
public class OrderServiceImpl implements OrderService {

    @Transactional(readOnly=true)
    public List getOrdersForCustomer(Customer customer) {…}

    public void processOrder() {…}
}
```

Spring 2.0 configuration with "tx" namespace:

```
<tx:annotation-driven/>
```

That's it!...

Spring will automatically detect @Transactional annotations

Spring

Spring 2.0: SimpleJdbcTemplate

- Motivations
  - Use new Java 5 features that can simplify usage
    - Varargs
    - Autoboxing
    - Parameterized methods
  - Offer only methods that are commonly used
    - Most commonly used callbacks
    - Fewer overloaded methods

- Offers access to a wrapped `JdbcTemplate` for more advanced operations

- Provides the `SimpleJdbcDaoSupport` class

Varargs and Autoboxing

```
jdbcTemplate.queryForInt("SELECT COUNT(0) FROM
    T_CLIENT WHERE TYPE=? AND CURRENCY=?",
    new Object[] { new Integer(13), "GBP" }
);
```

**JdbcTemplate, <=
Java 1.4**

```
jdbcTemplate.queryForInt("SELECT COUNT(0) FROM
    T_CLIENT WHERE TYPE=? AND CURRENCY=?",
    new Object[] { 13, "GBP" }
);
```

**JdbcTemplate,
autoboxing**

```
simpleJdbcTemplate.queryForInt("SELECT COUNT(0) FROM
    T_CLIENT WHERE TYPE=? AND CURRENCY=?",
    13, "GBP"
);
```

**SimpleJdbcTemplate,
available on Java 5**

Spring

Generics

- Generics make signatures clearer and eliminate casts

```
public Map<String, Object>
   queryForMap(String sql, Object... args)
         throws DataAccessException


public List<Map<String, Object>>
   queryForList(String sql, Object ... args)
         throws DataAccessException
```

Spring

Spring 2.0: JPA Integration

- JPA integration is consistent with other ORM integrations in Spring
  - "common concepts, common approach"
- Error handling, resource acquisition & release:
  - **JpaTemplate**
- Provide resources to templates:
  - **LocalEntityManagerFactoryBean** or JNDI-bound **EntityManagerFactory**
- Transaction Management:
  - **JpaTransactionManager** or JTA
- Dao Superclass:
  - **JpaDaoSupport**

JMS in Spring

- Spring 1.2 focused on synchronous JMS
  - JmsTemplate
  - Boilerplate reduction
  - Abstract 1.0.2 and 1.1 differences
  - Transaction integration

- High value, but people still had to code async functionality themselves, or use EJB MDBs
  - A clear need for async functionality built into Spring
    - "Message Driven POJOs"

Spring

Asynchronous JMS with Spring (1)

- Spring's new message listener container
  - full-fledged support for asynchronous message listener callbacks
    - to POJO implementing standard JMS MessageListener interface
    - or Spring's SessionAwareMessageListener
    - or use delegating adapter to drive messages to other classes

- Various flavors
  - DefaultMessageListenerContainer
  - SimpleMessageListenerContainer
  - ServerSessionMessageListenerContainer

MessageListener Interface

- Standard JMS MessageListener interface
  - onMessage(Message)
  - to be implemented by application classes
  - called for each incoming message that applies

```
public interface MessageListener {

  void onMessage(Message message);
}
```

Spring

```java
public class Listener implements MessageListener {

    public void onMessage(Message message) {

        if (message instanceof TextMessage) {

            try {

                String text = ((TextMessage) message).getText();

                log.info("Received: " + text);

            }

            catch (JMSException e) {

                e.printStackTrace();

            }

        }

    }

}
```

Spring

An Enhanced MessageListener Interface

- Spring's SessionAwareMessageListener interface
  - onMessage(Message, Session) callback
  - alternative to the standard JMS MessageListener interface
    - which has a Message argument only
  - allows for performing work on the active JMS Session, as passed in
    - sending reply messages

```
public interface SessionAwareMessageListener {

  void onMessage(Message message, Session session)
    throws JMSException;
}
```

Message Listener Container (1)

- DefaultMessageListenerContainer
  - async MessageListener support based on MessageConsumer receive facility
    - configurable number of concurrent consumers
    - using a loop with receive timeouts
  - threads managed by the listener container
    - through Spring 2.0's TaskExecutor abstraction
  - works within J2EE environments as well as standalone
    - only uses J2EE-compatible portions of JMS API
  - supports XA message reception!

Message Listener Container (2)

- Simple Spring bean definition example for DefaultMessageListenerContainer
  - pass in MessageListener reference
  - highly configurable, many defaults

```
<bean id="myMessageListenerContainer" class=
  "org.springframework.jms.listener.DefaultMessageListenerContainer"/>

  <property name="connectionFactory" ref="myConnectionFactory"/>
  <property name="destinationName" value="myQueue"/>
  <property name="messageListener" ref="myMessageListener"/>
</bean>

<bean id="myMessageListener" class="example.Listener">
  <property name="someDao" ref="myDao"/>
</bean>
```

Message Listener Container (3)

- DefaultMessageListenerContainer
  with transactional message reception

```xml
<bean id="myMessageListenerContainer" class=
  "org.springframework.jms.listener.DefaultMessageListenerContainer"/>

  <property name="connectionFactory" ref="myConnectionFactory"/>
  <property name="destinationName" value="myQueue"/>
  <property name="messageListener" ref="myMessageListener"/>
  <property name="transactionManager" ref="myTransactionManager"/>
</bean>

<bean id="transactionManager" class=
  "org.springframework.transaction.jta.JtaTransactionManager"/>

<bean id="myMessageListener" class="example.MyMessageListenerImpl">
  <property name="someDao" ref="myDao"/>
</bean>
```

Spring

Message Listener Container (4)

- Choosing a message listener container
  - use DefaultMessageListenerContainer for typical needs
    - J2EE-compatible, with or without XA transactions
    - fixed number of concurrent consumers
  - use ServerSessionMessageListenerContainer for dynamic needs
    - dynamic number of concurrent consumers
    - no XA transaction capability
  - use SimpleMessageListenerContainer for raw JMS provider usage
    - leverage JMS provider's built-in listener mechanism
    - no XA transaction capability

Web

New JSP Tags in Spring 2.0

- Spring MVC has always supported very rich data binding and validation functionality
  - Bind form data automatically to domain or form objects
  - Track binding and validation errors

- Only Velocity/FreeMarker views supported macros which both generated HTML elements and accessed this binding/validation status
- JSP pages forced to use lower level <spring:bind tag> and output HTML manually

New JSP Tags in Spring 2.0 (2)

- Spring 1.2:

```
<spring:bind path="employee.address.zip">
  ZIP
  <input type="text"
         name="${status.expression}"
         value="${status.value}"/>
</spring:bind>
```

- Spring 2.0 adds a rich JSP tag library:

```
<form:input path="address.zip" />
```

Portlet Support

- Spring 2.0 adds Spring Porlet MVC

  - Mirrors Spring MVC
    - DispatcherPortlet
    - Handler Mappings
    - Handler Interceptors
    - Controller Hierarchy
    - View Resolvers
    - Pluggable Views
    - Data Binding and Validation to command/form objects

  - Servlet and Portlet lifecycle differences do lead to some differences in the frameworks

Spring 2.0 - Summary

- Builds on the solid base established by Spring 1.x
- Pursues vision of POJO-based development
- Adds new capabilities and makes many tasks more elegant
  - make Spring more **powerful**
  - … while **simplifying** common tasks
- … with full backwards compatibility

- Provides a new foundation for future work
  - Most capable and flexible container on the market
  - The road ahead is exciting, with much more to come!

INTERFACE21

ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Hot on the Heels of Spring 2.0...

Spring Web Flow 1.0 is here

Spring

INTERFACE21

Spring Web Flow 1.0

- A full spring subproject

- Part of Spring's web stack

- Capture a logical flow of your web application as a self contained module, at a **higher level**

  - In a declarative fashion

  - Essentially a black box, including sub flows

  - Representing a user conversation

  - Introduces new scope: Flow Scope

  - Highly portable across lower level UI frameworks

  - Highly manageable

- Available at http://www.springframework.org

Spring

Spring-LDAP

Spring-LDAP 1.1

- A full spring subproject
- Simplifies LDAP operations, based on the pattern of Spring's JdbcTemplate
- Encapsulates nasty boilerplate plumbing code traditionally required for LDAP
  - Resource management and cleanup
  - Exception handling
  - Iterating

- Lower level access to data
- Higher level mapping of domain objects

- Available at http://www.springframework.org/ldap

Spring-OSGi

Spring

OSGi, What is it?

- Industry driven framework specification, with multiple implementations
- Dynamic component model, based around the idea of *bundles*

- Classloading (for isolation & versioning)
- Lifecycle control and definition
- Service Registry
- Standard Services
- Security Model

Spring

Why do we Need Spring-OSGi?

- Spring-OSGi is an integration library for Spring in OSGi environments

- *For those that need it*, allows a more powerful component programming model

  - Without Spring having to re-invent the wheel

  - ApplicationContexts become bundles, able to import and export services, with full isolation, integrated into OSGi lifecycles

- Project moving along rapidly, with large amount of interest and involvement from vendors such as BEA, Oracle, IBM, members of the OSGi foundation, and the general public

- Find it at http://www.springframework.org/osgi

Spring Web Services

Spring

Spring Web Services

- A full Spring subproject

- Focused on creating document-driven Web services
  - facilitate contract-first SOAP service development
  - allows for the creation of flexible web services using one of many ways to manipulate XML payloads

- Three major modules:
  - a **flexible Object/XML Mapping abstraction** with support for JAXB, XMLBeans, Castor, and JiBX
  - a **Web service framework** that resembles Spring MVC,
  - a **WS-Security module that integrates with Acegi Security**

- Currently pre-release (in 1.0 Milestone stage)
- Available at http://www.springframework.org/spring-ws

Spring

The Spring Experience

Spring

The Spring Experience 2006
December 7th – 10th, Hollywood Florida
by Interface21 and NoFluffJustStuff Java Symposiums

- World-class technical conference for the Spring community
- Experience 3 full days, 55 sessions across 5 tracks
    1. Core Spring 2.0
    2. Core Enterprise 2.0
    3. Core Web 2.0
    4. Domain-Driven Design
    5. Just Plain Cool
- Enjoy five-star beach resort and amenities
- Converse with core Spring team and industry experts
    – Rod Johnson, Adrian Colyer, Ramnivas Laddad, Juergen Hoeller
    – Eric Evans, Luke Hohmann, Eamon McManus
- **Register at http://www.thespringexperience.com**

Code Break & Q & A

- Questions?