Spring Framework 2.5: New and Notable

Rod Johnson, CEO, SpringSource

TS-6169

spring source

Java

Sun microsystems

Learn what's new in Spring 2.5 and why it matters to you

GOAL

Agenda

- Goals of Spring 2.5
- Support for new platforms
- Annotation based Dependency Injection
- @Component and other stereotype annotations
- Component scanning
- Spring MVC update
- The future

Background to Spring 2.5

- Spring has become de facto standard component model for enterprise Java
  - Gartner:
    - 75% of middleware vendors will provide Spring integration by 2009
  - Forrester
    - “Most enterprise Java users reported using Spring”
  - BEA
    - Most respondents to Dev2Dev survey use Spring
  - Job listings
    - Spring leads among Java component model technologies in worldwide job requirements

Goals of Spring 2.5

- To strengthen Spring's position as the de facto standard and most capable component model for enterprise Java
- To continue to deliver simplicity and power
  - Support for new platforms
  - Annotation support across the framework
  - Significant improvement in Spring MVC framework
  - Enhanced integration testing support

Support for new Platforms

New Platform support:

- Java 6 (JDK 1.6)
- Java EE 5
- OSGi

Java 6 Support

- One of the first major frameworks with dedicated support for Java 6 (JDK 1.6)
- New JDK 1.6 APIs supported:
  - JDBC 4.0
  - JMX MXBeans
  - JDK ServiceLoader API
- JDK 1.4 and 1.5 still fully supported
- JDK 1.3 no longer supported
  - Declared end-of-life by Sun a year ago

Improved JDBC support

- JDBC 4.0 feature support
  - Native connections (`java.sql.Wrapper`)
  - LOB Handling (`setBlob/setClob`)
  - New **`SQLException`** subclasses
- Other JDBC improvements:
  - **`SimpleJdbcTemplate`**
  - **`SimpleJdbcCall`** & **`SimpleJdbcInsert`**
  - Extended SQL error code mappings for MS SQL, MySQL, PostgreSQL and Oracle

Support for new Platforms

New Platform support:

- Java 6 (JDK 1.6)
- **Java EE 5**
- OSGi

Java EE 5 support

- Integration with Java EE 5 APIs
  - Servlet 2.5, JSP 2.1 & JSF 1.2
  - JTA 1.1, JAX-WS 2.0 & JavaMail 1.4
- J2EE 1.4 and 1.3 still fully supported
  - BEA WebLogic 8.1 or higher
  - IBM WebSphere 5.1 or higher
- Spring 2.5 component model processes annotations core to Java EE 5
  - JSR-250 injection and lifecycle annotations
    - “Common Annotations for the Java Platform”
  - EJB 3.0 annotations
    - Transactions
    - @EJB and other custom injection annotations

Java EE 5: APIs

- Support for unified expression language
- JSF 1.2: **`SpringBeanFacesELResolver`**
- JTA 1.1
  - Support new **`TransactionSynchronizationRegistry`**

Other J2EE enhancements: RAR file support

Ability to deploy Spring application as RAR file

- For J2EE 1.4 and Java EE 5 (JCA 1.5)
- For non-web deployment units driven by messages, scheduled jobs, etc.
  - Instead of headless WAR
  - Add a **META-INF/ra.xml** file that references a Spring applicationContext.xml file
  - Can access app server services like JTA TransactionManager and MBeanServer

Other J2EE enhancements: IBM WebSphere 6.x

Spring 2.5 is officially supported on IBM WAS 6.x

- Support for WebSphere-specific transaction management API
  - Including transaction suspension
    - Avoiding use of the raw JTA TransactionManager on WebSphere
  - On WebSphere 6.0.x and 6.1.x
- WebSphereUowTransactionManager
  - Enhanced replacement for standard Spring JtaTransactionManager using proprietary IBM APIs *without polluting application code*

Support for new Platforms

New Platform support:

- Java 6 (JDK 1.6)
- Java EE 5
- **OSGi**

Spring & OSGi

- **O**pen **S**ervices **G**ateway **i**nitiative
- Dynamic module system for Java – Key to future of enterprise Java
  - Clean isolation of subsystems
  - Versioning
  - Hot deployment
- *Bundle* as central packaging unit
  - Versioned JAR
  - Exports types to expose
  - Specifies types to import
  - Can be started, stopped and updated *at runtime*
- Spring 2.5 JARs now are OSGi bundles
  - Headers in MANIFEST.MF
  - Ultimate modularization for any framework
    - Load only what is needed

Spring & OSGi

- Integration with OSGi provided by the ***Spring Dynamic Modules for OSGi™ Service Platforms*** project
- **`One ApplicationContext`** per bundle
- Integration with OSGi service registry
- Combines proven Spring component model with proven OSGi module system

- SpringSource Application Platform bases its generic middleware kernel on OSGi

Spring on OSGi vs Spring on Java EE

- Spring offers consistent programming model in either case
  - No dependencies on environment: just POJOs
  - Services are Spring managed components
  - Portability between environments identified by Gartner as a key benefit of Spring

Agenda

- Goals of Spring 2.5
- Annotation based Dependency Injection (DI)
- @Component and other stereotype annotations
- Component scanning
- Spring MVC update
- The future

Annotations

- Introduced in Java 5.0
- Add metadata to source code
- Spring has offered annotations for enterprise services (transaction management and JMX) since Spring 1.2
- Comprehensive annotation support for DI introduced in Spring 2.5

Purpose of annotations for DI

- **Annotations applied to classes, methods or fields**
- **Identify injection points and help to resolve values to inject**

- Annotations on *methods* identify methods whose arguments should be injected
  - Can have multiple arguments
- Annotations on *fields* identify value that should be injected

- Optional annotations on *method arguments* provide information about how to resolve dependency
- Annotations on *classes* identify components to be managed by Spring

Annotation Based DI: Pros and Cons

- Pros
  - Annotations can reduce or eliminate external configuration
  - More concise mechanism because you specify *what* should be injected, with the location of the annotation providing *where*
- Cons
  - Annotations are per-type not per-instance
  - Doesn't work for legacy code with existing classes without annotations
  - Need to recompile Java code to modify configuration
  - Not well suited to externalizing simple types

Choices for annotation-driven DI in Spring 2.5

- @Autowired
  - Native Spring annotation syntax
  - Designed in late 2007
  - Integration of proven Spring model with experience from use of annotation-driven models
- @Resource
  - JSR-250/EJB3 model

Resolving Dependencies: @Autowired

- Injection at constructor/field/method level
- Supports multi argument methods
  - Concise
- Default behavior is Spring's traditional *autowire by type*
- Annotations make autowiring much more useful

```
@Autowired
public void createTemplates(DataSource ds,
                            ConnectionFactory cf) {
  this.jdbcTemplate = new JdbcTemplate(ds);
  this.jmsTemplate = new JmsTemplate(cf);
}
```

@Qualifier Annotation

- Autowiring by type may have too many candidates
- Provide hints using qualifiers!
  - New `@Qualifier` annotation
  - Can be used on fields / parameters or on custom annotations

Resolution of dependencies by name

```
public class JdbcOrderRepositoryImpl
  implements OrderRepository {

  @Autowired
  public void init(
  @Qualifier("myDataSource")
      orderDataSource,
      @Qualifier("otherDataSource")
  inventoryDataSource,
      MyHelper autowiredByType) {
          // ...
}
```

Resolution of dependencies by annotation

```
public class JdbcOrderRepositoryImpl
  implements OrderRepository {

  @Autowired
  public void setOrderServices(
      @Emea OrderService emea,
      @Apac OrderService apac) {
          // ...
}
```

Association of injection target with annotation: By annotation

```
@Emea
public class EmeaOrderService
       implements OrderService {

       ...
}
```

```
@Qualifier
@Component
public @interface Emea {
}
```

```
@Apac
public class ApacOrderService
       implements OrderService {
       ...
}
```

```
@Qualifier
@Component
public @interface Apac{
}
```

Association of injection target with annotation: XML

```
<bean class="example.EmeaOrderService">
    <qualifier type="example.Emea"/>
    <!-
        ...
        EmeaOrderService need not be annotated
     -->
</bean>
```

```
<bean class="example.ApacOrderService">
    <qualifier type="example.Apac"/>
    <!-- inject any dependencies required by
  this bean -->
</bean>
```

@Autowired pros and cons

- Pros
  - Capable model
  - Simple, concise, yet powerful
  - Allows avoidance of Spring-specific annotations on injection targets
- Cons
  - Spring specific mechanism for injectees
    - …but no runtime dependence on Spring

@Resource for injection

- `@Resource`
  - Identifies injection point
  - Resolves to a single component
  - Spring does not require that the component comes from JNDI, although Spring can transparently resolve JNDI references

@Resource Example

```
public class DefaultAccountService
                    implements AccountService {

   @Resource
   private AccountDAO jdbcAccountDAO;
   ...
}


public class JdbcAccountDAO implements AccountDAO {

   @PostConstruct
   public void init() {...} ...
}
```

@Resource Pros and Cons

- Pros
  - Supports Java EE 5 configuration style
  - May help portability
- Cons
  - Limited power
    - @Resource style is not as powerful as Spring @Autowired approach
      - Can only resolve a single reference
      - No support for “qualifiers” or annotation resolution

JSR-250 lifecycle annotations

- `@PostConstruct`
  - Similar to `InitializingBean#afterPropertiesSet()`
- `@PreDestroy`
  - Similar to `DisposableBean#destroy()`
- Best practice
- Simple but valuable functionality to standardize
- Not Spring specific
- **We recommend using these annotations in place of Spring init-method or InitializingBean interfaces**

Agenda

- Goals of Spring 2.5
- Annotation based Dependency Injection
- @Component and other stereotype annotations
- Component scanning
- Spring MVC update
- The future

The @Component meta-annotation

- *Meta-annotations*
  - Annotations can annotate other annotations
  - Allow extensibility
    - Similar to Java inheritance
- Spring *stereotypes*
  - Concept introduced in Spring 2.0, but more stereotypes added later
  - Identify classes with a particular purpose
  - Help to build a strong semantic model of application
  - Not Spring-specific
    - Common identifiers for regular code
  - Valuable targets for AspectJ pointcuts, useable directly in Spring since Spring 2.0

Out-of-the-box *stereotype* annotations

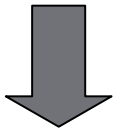
- @Service
  - Identifies a stateless service
- @Repository
  - Identifies a repository (DAO)
- @Aspect
  - @AspectJ aspect
- @Controller
  - Spring MVC controller

- Can define your own…
- @Component
  - *Meta-annotation*
    - Annotate your own annotation with @Component and your classes get picked up by scanning
  - @Emea example earlier

Component Scanning

- Scans the classpath for annotated classes
- Removes the need for XML definitions unless you want to need to do something you can't do in annotations

```
@Service
public class DefaultAccountService { ...
```



```
<bean id="defaultAccountService"
                    class="DefaultAccountService"/>
```

Component Scan Usage

- Use Spring core *context* namespace
- Specify package(s) to pick up
- Can coexist with XML bean definitions and namespaces

```
<context:component-scan
      base-package="com.mycompany.myapp"/>
```

More advanced component scanning usage

- Not limited to annotations
  - Can use type or other checks
- Highly customizable, as you expect from Spring
  - Can even work without using annotations

```
<context:component-scan base-package="blog"
                        use-default-filters="false">
    <context:include-filter type="annotation"
      expression="org.sf.stereotype.Component"/>
    <context:include-filter type="regex"
      expression="blog\.Stub.*"/>
    <context:exclude-filter type="assignable"
      expression="blog.JdbcMessageRepository"/>
</context:component-scan>
```

Component Scan Pros

- No need for XML unless you need the greater sophistication it allows
- Changes are picked up automatically
  - Great during development
- Works great with Annotation Driven Injection
  - picking up further dependencies with @Autowired
- Highly configurable

Component Scan Cons

- Not a 100% solution
  - Can't do everything with annotations
- Requires classes to be annotated
- Need to take care not to scan an excessive number of classes, using Spring's filtering mechanism
- Don't get the valuable application structure blueprints you get with XML configuration
  - Although Spring IDE can unify all Spring component definitions

Mix and Match

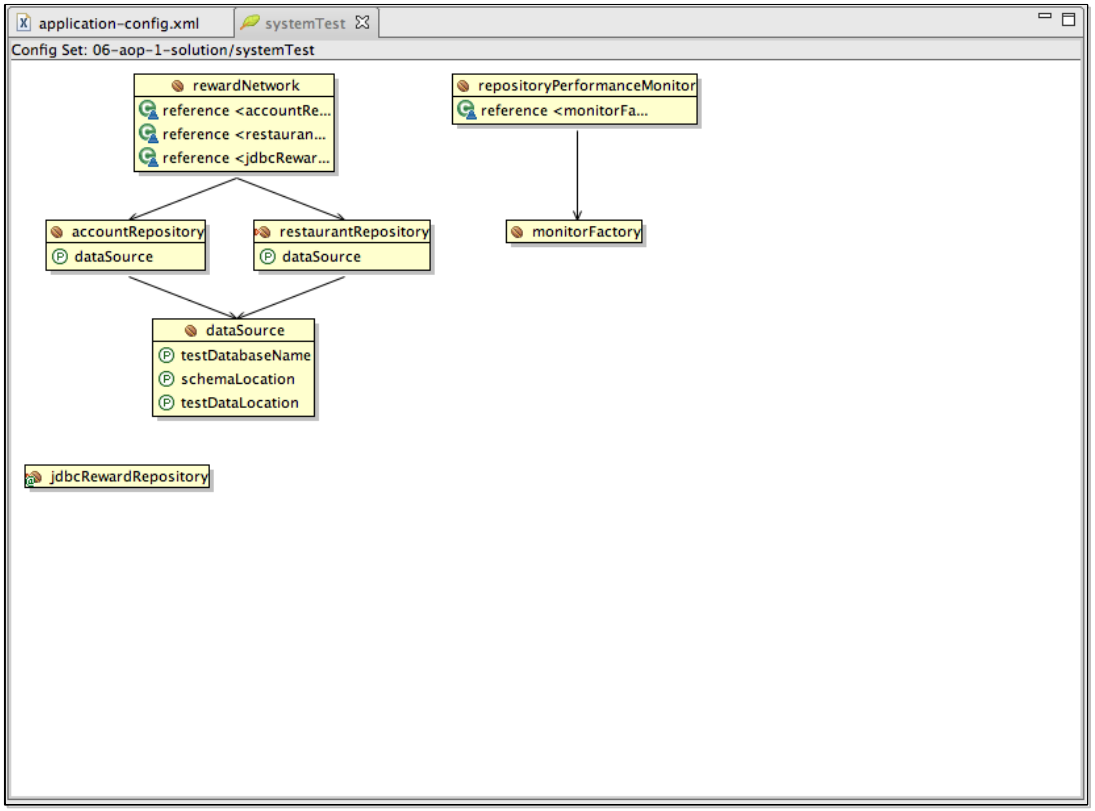
- All Spring metadata in the end
- Spring component model is independent of metadata
- One approach does not exclude others
- Can have multiple *contributions* to a single context

Mixing and matching XML and annotation configuration

- Naming conventions
  - Provide an XML bean definition for the short name of the bean class
  - In more complex scenarios, provide a “name” for the @Component annotation
- Combine the best of both worlds
  - Never hit a wall
  - Escape to the powerful and proven Spring XML model when you hit the limitations of annotations
    - *Without the need to rework anything you’ve done…*

Spring IDE Visualization and Editing support

- Spring IDE provides sophisticated visualization and editing support for bean definitions, *however defined*
- Unified view of configuration



Agenda

- Goals of Spring 2.5
- Annotation based Dependency Injection
- @Component and other stereotype annotations
- Component scanning
- **Spring MVC update**
- The future

Spring MVC annotations

- Major enhancements in Spring MVC in Spring 2.5

- Spring MVC has always offered a flexible model
  - Extension points allowing customization of any part of the framework
    - HandlerMappings
    - HandlerAdapters
    - ViewResolvers
  - Flaws relating to concrete inheritance in typical usage
    - SimpleFormController and other base classes with numerous template methods

Annotation-driven Controllers

- Java 5 evolution of MultiActionController
  - Including form handling capabilities
- POJO-based
  - Just annotate your class
  - Works in servlet and portlet container
- Annotations offer superior programming model
  - @Controller
  - @RequestMapping
  - @RequestMethod
  - @RequestParam
  - @ModelAttribute
  - @SessionAttributes
  - @InitBinder

Example of Annotated MVC Controller

```
@Controller
@RequestMapping("/order/*")
public class OrderController {

  @Autowired
  private OrderService orderService;

  @RequestMapping("/print.*")
  public void printOrder(HttpServletRequest request,
      OutputStream responseOutputStream) {
    ...
    // write directly to the OutputStream:
    orderService.generatePdf(responseOutputStream);
  }

  @RequestMapping("/display.*")
  public String displayOrder(
      @RequestParam("id") int orderId, Model model) {
    ...
    model.addAttribute(...);
    return "displayOrder";
  }
}
```

Advanced annotation-based MVC

- Annotations for
  - Session attributes
  - Data binder initialization
  - Form lifecycle
- See the PetClinic sample application that ships with Spring
  - Compare with Spring 1.0 version to see how much simpler today's Spring is to use!

Agenda

- Goals of Spring 2.5
- Annotation based Dependency Injection
- @Component and other stereotype annotations
- Component scanning
- Spring MVC update
- The future

Beyond Spring: The Spring Portfolio

- The **Spring Portfolio** extends beyond the Spring Framework to provide an increasingly complete solution for enterprise Java requirements
- Recent string of major releases
  - Spring Security 2.0
    - Evolution of Acegi Security for Spring with massive reduction in configuration required
    - Most capable security solution for enterprise Java™
  - Spring Batch
    - Co-developed by SpringSource and Accenture
  - Spring Web Flow
  - Spring Web Services
  - Spring Integration
  - Spring Dynamic Modules for OSGi

Spring Ecosystem

- Spring ecosystem extends to commercial and other open source projects
  - SpringSource Application Platform
    - Announced last week
  - WebLogic Server
  - WebLogic Event Server
  - Gigaspaces
  - SpringSource Tool Suite
  - SpringSource Advanced Management Suite (AMS)
  - SpringSource Advanced Pack for Oracle Database
  - ..

Spring 3.0

- Q3, 2008

- Moves to Java 5+ basis
- Further improvements in Spring MVC will provide a unified programming model between Spring MVC and Spring Web Flow to handler the full range of web programming requirements
- Comprehensive REST support across Spring MVC and Spring Web Services

Summary

- Spring 2.5 makes Spring easier to use, but still more powerful
- Adds extensive support for annotations across the framework
- Spring MVC 2.5 leverages Java 5 features to provide more concise, more flexible model
- The Spring Portfolio extends beyond the Spring Framework to handle a wide range of enterprise requirements
- Spring 3.0 will continue the rapid progress of Spring to meet tomorrow's requirements
- Growing set of choices for optimal deployment of Spring based applications

For More Information

- Presentations at JavaOne
  - Dave Syer, *Spring Batch* presentation
  - Ben Alex, *Spring Security* presentation
- Online resources
  - Spring Framework home: www.springframework.com
  - SpringSource home: www.springsource.com
- Visit the SpringSource booth

THANK YOU

Rod Johnson, CEO, SpringSource

TS-6169