Spring Framework 2.5: New and Notable

Ben Alex, Principal Software Engineer, SpringSource

GOAL>

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

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

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

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 Java EE 5 annotations
  - JSR-250 injection and lifecycle annotations

Other JEE enhancements: RAR support

Ability to deploy Spring application as a RAR file

- For J2EE 1.4 and Java EE 5 (JCA 1.5 `ResourceAdapter`)

- For non-web deployment units driven by messages, jobs etc
  - Instead of headless WAR
  - Add a **`META-INF/ra.xml`** file that references a Spring applicationContext.xml file
  - Put the required library JARs in the root of the RAR archive
  - Can access app server services like JTA `TransactionManager` and `MBeanServer`

Other JEE enhancements: IBM WebSphere 6

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*

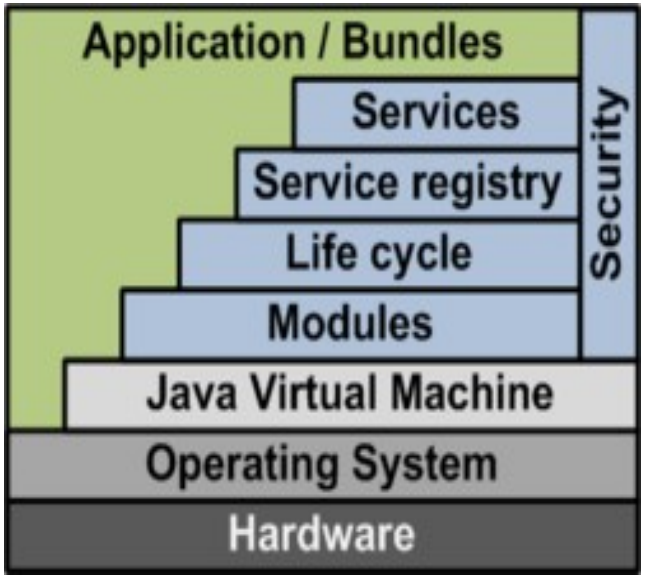
Support for new Platforms

New Platform support:

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

Spring and OSGi

- Open Services Gateway Initiative
- Dynamic module system for Java
  - Clean isolation of modules
  - Versioning
  - Hot deployment
- A *bundle* is the central packaging unit
  - Versioned JAR
  - Specifies types being exported
  - Specifies types that need to be imported
  - Can be dynamically changed at runtime



Spring is OSGi ready – today!

- Most recent Spring Portfolio similarly provide OSGi metadata
  - For example, Spring 2.5 JARs include OSGi metadata in the manifest
- Spring Dynamic Modules provides Spring-OSGi integration
- SpringSource Application Platform uses an OSGi kernel
- SpringSource Enterprise Bundle Repository provides bundles
- JEE remains fully supported by Spring
  - WARs, RARs, EARs and PARs with a *consistent* programming model

VIDEO>

SpringSource Application Platform

SpringSource Tool Suite

SpringSource Enterprise Bundle Repository

Agenda

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

Annotation-driven DI in Spring 2.5

- We've supported annotations in Spring since 2004
- @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

Annotation-driven 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

Resolving Dependencies: @Autowired

- Injection at constructor/field/method level
- Supports multi-argument methods
  - Concise

- 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

```
public class JdbcOrderRepositoryImpl
  implements OrderRepository {

  @Autowired
  public void init(
     @Qualifier("myDS")
     DataSource orderDataSource,
     @Qualifier("otherDS")
     DataSource inventoryDataSource,
     MyHelper autowiredByType) {
        // ...
}
```

Using your own @Qualifier annotations

```
public class JdbcOrderRepositoryImpl
  implements OrderRepository {

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

Using your own @Qualifier annotations

```
@Emea
public class EmeaOrderService
      implements OrderService {

      ...
}
```

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

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

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

Using your own @Qualifier annotations

```
<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
  - @Qualifier annotation avoids Spring annotations on target
- Cons
  - Same cons as mentioned earlier for annotation-based DI
  - Plus @Autowired is a Spring-specific mechanism
    - …but you can still invoke the methods as per usual

@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;
   ...
}
```

@Resource Pros and Cons

- Pros
  - Supports Java EE 5 configuration style
  - May help portability
- Cons
  - Limited power
    - @Resource style is not as powerful as @Autowired
      - Can only resolve a single reference
      - No support for “qualifiers” or annotation resolution
      - Forced to import JEE annotations directly into your Java types

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

Out-of-the-box stereotype annotations

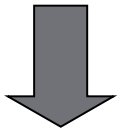
- @Service
  - Identifies a stateless service
- @Repository
  - Identifies a repository (DAO)
- @Aspect
  - @AspectJ aspect
- @Controller
  - Spring 2.5 @MVC controller

- Can define your own…
- @Component
  - *Meta-annotation*
    - Annotate your own annotation with @Component and receive component scanning
  - @Emea example earlier

Component Scanning

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

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



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

Component Scan Usage

- Specify package(s) to pick up
- Can coexist with XML bean definitions and namespaces
- Advanced component scanning syntax also available

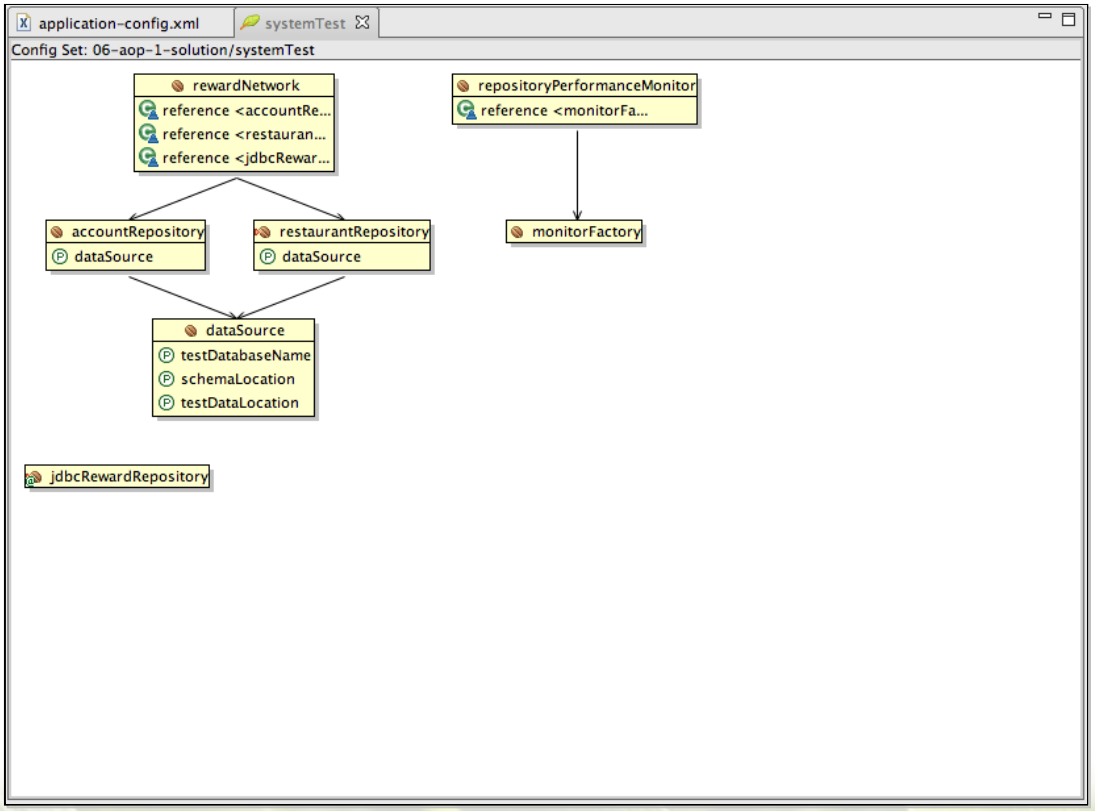
```
<context:component-scan
   base-package="com.mycompany.myapp"/>
```

Component Scan Pros and cons

- Pros
  - No need for XML unless you really need it
  - Changes (eg new classes) automatically discovered
  - Highly configurable if using Spring's @Autowired model
- Cons
  - Not a 100% solution – you'll still need XML sometimes
  - Avoid excessive classpath scanning
  - Lack simplified XML application structure
    - Unless you use Spring IDE!
- You can concurrently mix and match!

Spring IDE Visualization and Editing support

- Unified view of configuration

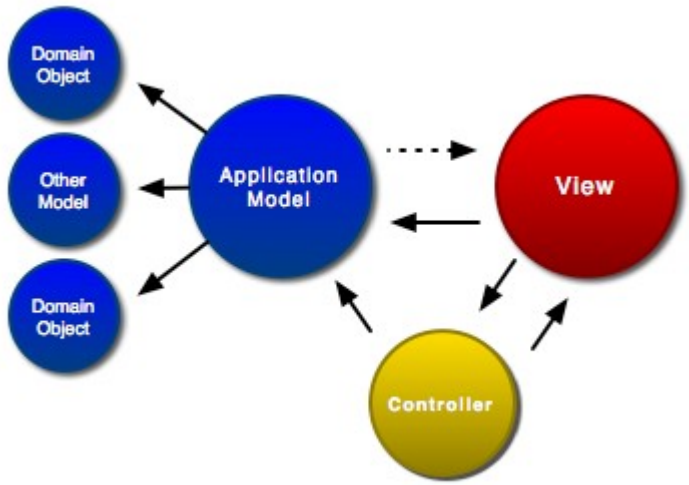


Agenda

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

Annotated @MVC Controllers

- Java 5 evolution of `MultiActionController`
  - Including form handling capabilities
- POJO-based
  - Just annotate your class
  - Works in servlet and portlet container
- Annotations provided
  - @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";
  }
}
```

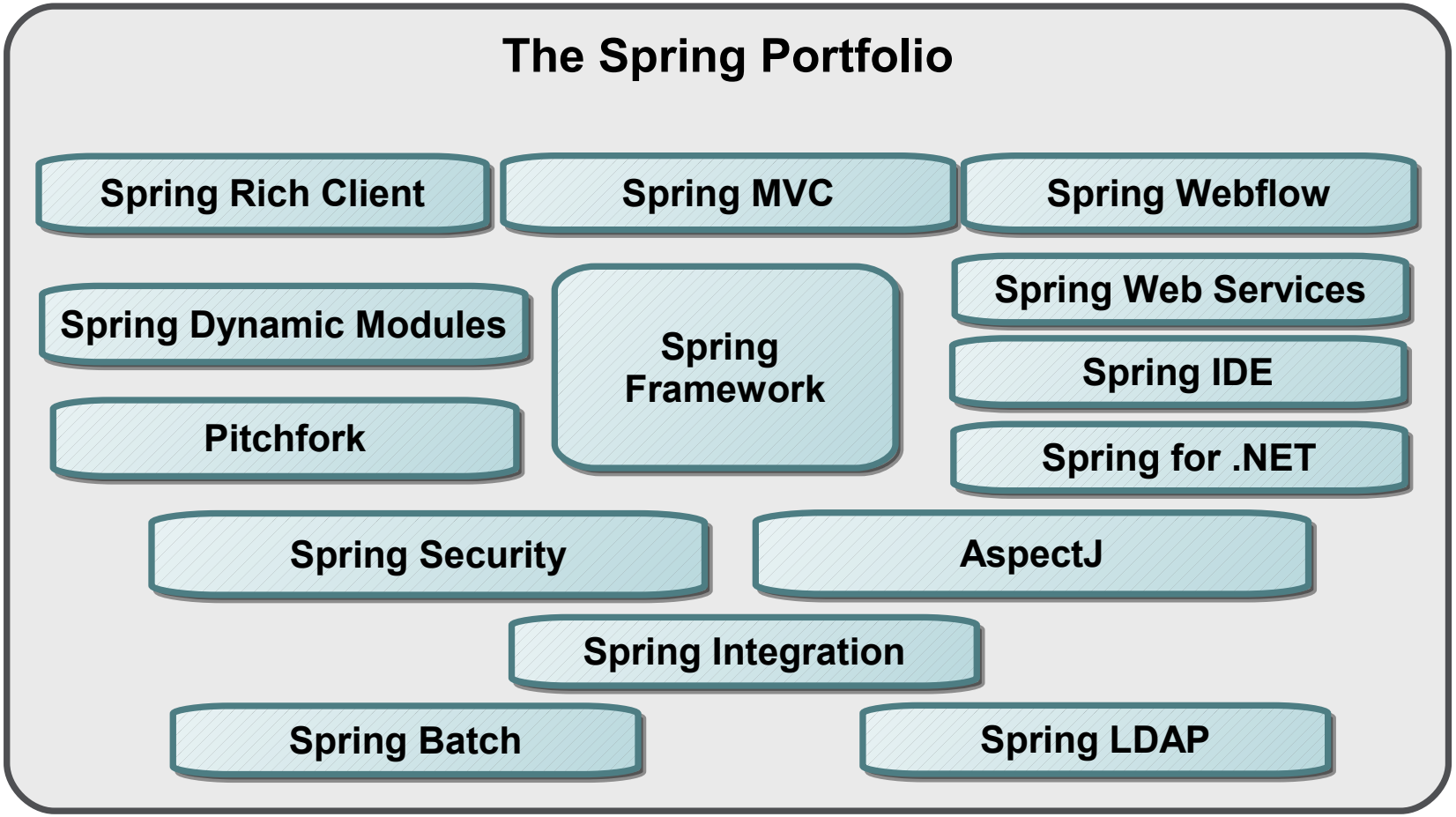
Advanced annotation-based MVC

- 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

Spring Open Source Ecosystem



Spring Commercial Product Ecosystem

- SpringSource Tool Suite
  - Free for personal use – download it today and quick-start with Spring 2.5!
- SpringSource Application Platform
  - The perfect way to build OSGi applications – today (and free under GPL!)
- SpringSource Enterprise
  - Pre-integrated Spring with 24/365 guaranteed fixes, support, indemnity
- SpringSource Application Management Suite
  - Managing and monitoring your production environment
- SpringSource Advanced Pack for Oracle Database
- SpringSource Enterprise Ready Server
- Enterprise Support for HTTPD, Tomcat and ActiveMQ

Spring 3.0

- Q3, 2008

- Moves to Java 5+ basis
- Further improvements in Spring MVC will provide a unified programming model between MVC and Web Flow
- Comprehensive REST support across MVC and 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

- Online resources
  - Spring Framework home: www.springframework.com
  - SpringSource home: www.springsource.com
- Visit the SpringSource booth

THANKS>

ben.alex@springsource.com