What's New in Spring 2.5

Rossen Stoyanchev
Senior Consultant, SpringSource

Agenda

- **Platform Updates**
  - JDK, Java EE
  - AspectJ, OSGi
- Annotation-Driven Configuration
- Spring @MVC
- The TestContext Framework

JDK Updates

- Dedicated support for JDK 1.6
  - JDBC 4.0
  - JMX MXBeans
  - JDK ServiceLoader API
  - Built-in HTTP Server
- JDK 1.5 and 1.4
  - fully supported
- JDK 1.3
  - EOF (End of Life)

Java EE Updates

- Dedicated support for Java EE 5
  - Servlet 2.5, JSP 2.1 & JSF 1.2
  - JTA 1.1
  - JAX-WS 2.0
  - JavaMail 1.4
  - RAR deployment
  - WebSphere Transaction API (UOW)
- Existing J2EE versions
  - 1.4 and 1.3 fully supported

AspectJ Support

- New bean name pointcut
  - e.g. “bean(*Service)”, “bean(*Controller)”
  - replaces BeanNameAutoProxyCreator
- Support AspectJ load time weaving
  - through Spring's LoadTimeWeaver
  - AspectJ compiler not needed
  - enable for @Transactional via tx namespace

OSGi

- All Spring jars packaged as OSGi bundles
  - headers in MANIFEST.MF
  - dependencies listed
- Ready for use in OSGi applications
- Spring Dynamic Modules (currently 1.0.1)
  - formerly Spring OSGi
  - provides OSGi integration

Packaging Changes

- Separated out from spring.jar
  - spring-webmvc-*.jar (mvc, portlets, struts)
- Merged
  - spring-tx.jar (dao, jca)
  - spring-orm.jar (jpa, jdo, hibernate, toplink, ibatis)
  - spring-context (jmx, remoting)
- Renamed
  - spring-mock.jar => spring-test.jar
  - spring-support.jar => spring-context-support.jar

Agenda

- Platform Updates
  - JDK, Java EE
  - AspectJ, OSGi
- **Annotation-Driven Configuration**
- Spring @MVC
- The TestContext Framework

Standard JSR-250 Annotations

- Java Standard DI & Lifecycle Annotations
  - `@Resource`
    - `Injection of JNDI resources or named beans`
    - `Setters/Fields`
  - `@PostConstruct` & `@PreDestroy`
    - Lifecycle callback methods
- Included with Java EE 5 and JDK 1.6
  - available as a jar for JDK 1.5
- Supported in Spring 2.5!

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {

  @Resource
  private DataSource dataSource;

  @Resource
  private Processor processor;

  @Resource(name="myProcessor")
  public void setProcessor(Processor processor) { … }

  @PostConstruct
  public void initialize() { … }

  @PreDestroy
  public void shutdown() { … }
}
```

```
<bean class="org.frammework.context.annotation
                    .CommonAnnotationBeanPostProcessor"/>
```

Annotation-Driven Configuration

- Spring 2.5 embraces annotations for DI
  - JSR-250 annotations
  - fine-grained DI with @Autowired
  - component scan
- Possible to remove XML entirely
  - however XML is not deprecated!
  - annotations: one more option!

Standard Spring Bean XML-based Configuration

```
public class MyService implements MyServiceInterface {

   private EmployeeDao employeeDao;
   private CustomerDao customerDao;

   public void MyService(EmployeeDao emplDao, CustomerDao custDao) {
      this.employeeDao = emplDao;
      this.customerDao = custDao;
   }
}
```

```
<bean class="MyService"/>
   <constructor-arg ref="employeeDao"/>
   <constructor-arg ref="customerDao"/>
</bean>

<bean class="employeeDao" class="JdbcEmployeeDao">
   <constructor-arg class="dataSource"/>
</bean>

<bean class="customerDao" class="JdbcCustomerDao">
   <constructor-arg class="dataSource"/>
</bean>
```

@Autowired

- Same purpose as JSR-250 @Resource
- But more powerful, more flexible
- Enables fine-grained DI
- Applies to
  - fields, methods, constructors

Add @Autowired

```
public class MyService implements MyServiceInterface {

  private EmployeeDao employeeDao;
  private CustomerDao customerDao;

  @Autowired
  public void MyService(EmployeeDao emplDao, CustomerDao custDao) {
     this.employeeDao = emplDao;
     this.customerDao = custDao;
  }
}
```

```
<context:annotation-config/>
<bean class="MyService”/>
<bean class="employeeDao” class="JdbcEmployeeDao”/>
<bean class="customerDao” class="JdbcCustomerDao”/>
```

Find the "Right" Match

- The default is a match "by type"
- But this may lead to ambiguity
  - multiple candidates of the same type
- Possible to provide "hints"
- Use @Qualifier
  - match by bean name
  - match by custom annotation

Add @Qualifier (if needed)

```
public class MyService implements MyServiceInterface {

  private EmployeeDao employeeDao;
  private CustomerDao customerDao;

  @Autowired
  public void MyService(
         @Qualifier("naEmployeeDao") EmployeeDao emplDao,
         @Qualifier("naCustomerDao") CustomerDao custDao) {
     this.employeeDao = emplDao;
     this.customerDao = custDao;
  }
}
```

```
<bean id="naEmployeeDao" class="JdbcEmployeeDao"/>
<bean id="naCustomerDao" class="JdbcCustomerDao"/>
...
<bean id="euEmployeeDao" class="JdbcEmployeeDao"/>
<bean id="euCustomerDao" class="JdbcCustomerDao"/>
...
```

More @Autowired...

- Typed Collections

```
@Autowired
private List<MovieCatalog> catalogs;
```

- Replace ***Aware** interfaces:

```
@Autowired
private MessageSource messageSource;

@Autowired
private ResourceLoader resourceLoader;


@Autowired
private ApplicationContext applicationContext;
```

Component Scan

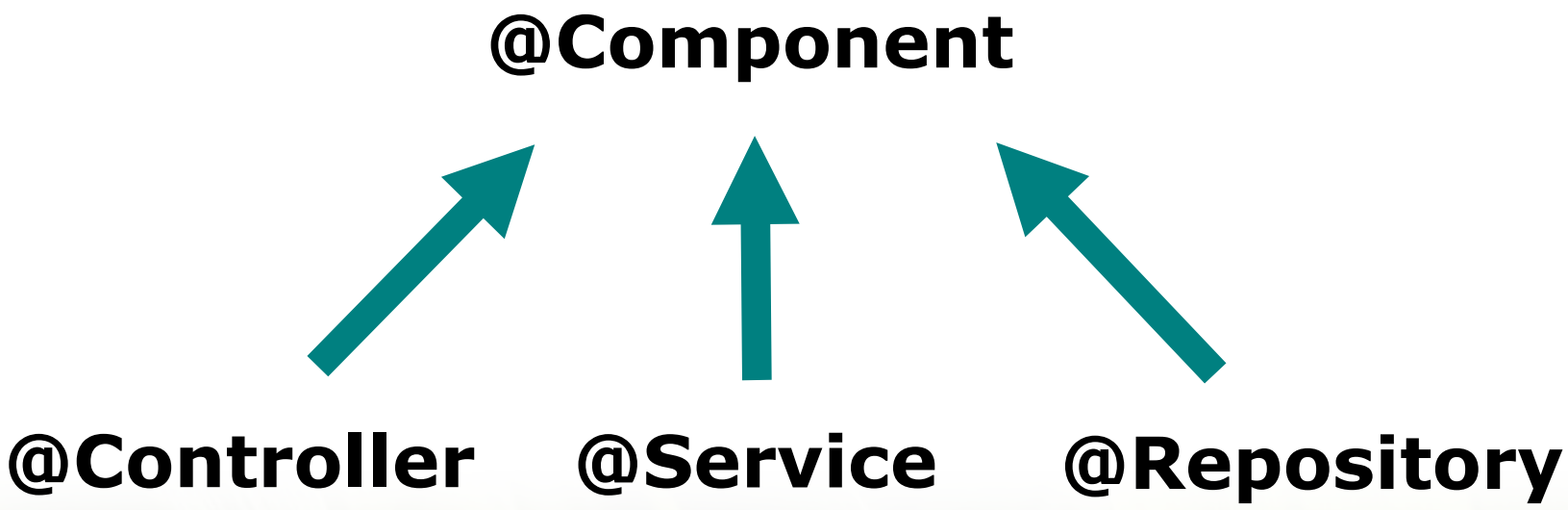
- @Autowired reduced the amount of XML
  - takes care of wiring among beans
- Yet we still need <bean /> tags
  - declare beans

```
<bean class="MyService”/>
<bean class="employeeDao” class="JdbcEmployeeDao”/>
<bean class="customerDao” class="JdbcCustomerDao”/>
```

- A component scan can “discover” beans

Stereotype Annotations

- Designate classes according to system layering
- All are logical extensions of @Component
- Easy to create your own extensions



Add Component Scanning

```
@Repository("myService")
public class MyService implements MyServiceInterface {

   private EmployeeDao employeeDao;
   private CustomerDao customerDao;

   @Autowired
   public void MyService(EmployeeDao emplDao, CustomerDao custDao){..}
}
```

```
@Repository("employeeDao")
public class JdbcEmployeeDao implements EmployeeDao {..}
```

```
@Repository("customerDao")
public class JdbcCustomerDao implements CustomerDao {..}
```

```
<context:annotation-config/>
<context:component-scan base-package="org.abc"/>
```

Add Component Scanning

- Component scanning is configurable
- Includes, excludes
- Match by class name, annotation, pointcut, etc.

```
<context:component-scan base-package="org.abc">
    <context:include-filter type="regex"
                        expression=".*Stub.*Repository">
    <context:exclude-filter type="annotation"
                        expression="com.abc.MyAnnotation">
<context:component-scan/>
```

Annotations versus XML

- Pros:
  - Less configuration, very little XML
  - Remove minor Spring dependencies
    - *Aware interfaces, @PostConstruct, @PreDestroy
  - Support for refactoring
- Cons:
  - No central application “blueprint”
  - Configuration is per class, not per instance
  - Requires annotations in the code
  - Change requires recompilation

Summary of Annotation-Driven Configuration

- Component scanning
  - removes need for <bean/> tags
  - user filters to reduce start-up overhead
- Dependency injection
  - use standard JSR-250 annotations
  - @Autowired/@Qualifier for fine-grained DI
- XML is not deprecated
- Possible to mix and match!

Agenda

- Platform Updates
  - JDK, Java EE
  - AspectJ, OSGi
- Annotation-Driven Configuration
- **Spring @MVC**
- The TestContext Framework

Annotations in the Web Layer

- Web layer configuration adds up quickly
  - many controllers, properties, etc.
- Spring 2.5 introduces MVC annotations
  - informally known as "@MVC"
  - component scanning for controllers
  - request mappings
  - data binding
  - and more...

Existing Spring 2.0 MVC Controller Technology

- SimpleFormController
  - base class template for form pages
  - great for single page edits (Save/Cancel)
- MultiActionController
  - base class to have multiple handler methods
  - great for more advanced page edits (CRUD)
- AbstractWizardFormController
  - similar to SimpleFormController
  - used for multi-page input

Spring 2.5 Controller Technology

- Annotate class with @Controller
  - no base type or interface requirements
- Use @RequestMapping
  - class-level or method-level
- Results in an interesting combination
  - like MultiActionController
    - but with help for data binding, model set-up, etc.
  - like SimpleFormController
    - but without having to learn base class methods

Annotated Controller Example

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

    @Autowired
    private OrderService orderService;

    @RequestMapping("print.htm")
    public void printOrder(OutputStream httpResponseOutputStream){
        orderService.generatePdf(httpResponseOutputStream);
    }

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

Setting Up a Form Page

```
@Controller
@RequestMapping("/editPet.do")
@SessionMapping("pet")
public class EditPetForm {

   @ModelAttribute("types")
   public Collection<PetType> populatePetTypes(){
      return clinic.getPetTypes();
   }

   @RequestMapping(method=RequestMethod.GET)
   public String setupForm(
            @RequestParam("petId") petId, ModelMap model){
      Pet pet = clinic.loadPet(id);
      model.addAttribute("pet");
      return "petForm";
   }
}
```

Processing a Form Submit

```
    ...

    @RequestMapping(method=RequestMethod.POST)
    public String processSubmit(
              @ModelAttribute("pet") pet, BindingResult result){
        new PetValidator().validate(pet, result);
        if (result.hasErrors()) {
            return "petForm";
        }
        clinic.storePet(pet);
        return "redirect:/owner.do?ownerId=" +
            pet.getOwner().getId();
    }

}
```

Summary of Spring MVC Annotations

- POJO Controller Technology
- Intuitive request mapping
  - multiple handler methods per class
  - URL and request method matching
- Flexible method signatures
- Controller services via annotations
  - data binding
  - model set-up
  - request parameter extraction

Shameless Plug

- Much more on this tomorrow..

Agenda

- Platform Updates
  - JDK, Java EE
  - AspectJ, OSGi
- Annotation-Driven Configuration
- Spring @MVC
- **The TestContext Framework**

Testing in Spring 2.5

- The TestContext Framework
- Annotation-driven
- Test framework agnostic
  - Unit 4.4
  - JUnit 3.8
  - TestNG

TestContext Annotations

- @Transactional in test classes
  - Requires “transactionManager” bean
  - Automatic roll-back after each test
  - Code under test participates in transaction
- Other TestContext annotations
  - @IfProfileValue(name="", value="")
  - @ExpectedException(SomeException.class)
  - @NotTransactional
  - @Rollback(false)
  - @DirtiesContext

Test Helper classes

- SimpleJdbcTestUtils
  - Count table rows, empty tables
  - Run SQL script
- ReflectionTestUtils
  - Set non-public fields
  - Invoke non-public setters

- Independent of TestContext framework!

Summary of TestContext Framework

- Choice of test frameworks
- No need to extend from a base class
- Combines with @Autowired
- Better control over Spring test features
  - @Transactional, @NotTransactional
  - @DirtiesContext
- New test features
  - @Timed, @ExpectedException

Summary

- Platform Updates
  - Java 6, Java EE 5
  - OSGi, AspectJ
- Embraces annotations
  - Dependency Injection, Web Layer, Testing
  - an addition to existing alternatives
- Enhanced XML configuration
  - new custom namespaces

Spring 2.5 Sample Applications

- PetClinic
  - completely revised for Spring 2.5
  - annotation-driven configuration
  - annotation-driven MVC controllers
    - focus on simple form handling
  - annotation-driven tests
- imagedb
  - annotation-driven configuration
  - annotation-driven MVC controllers
    - focus on stateless multi action handling

For More Information

- Updated sample applications
- Spring Documentation
- Spring Forums
- Other
  - http://blog.springsource.com
  - http://www.springframework.org

Questions & Answers