



Spring 2.5

An Update for Spring 2.0 users

**Arjen Poutsma & Joris Kuipers
SpringSource**

Speaker's Qualifications



- Arjen is open source rock star
- Lead of Spring Web Services and developer on Spring 3.0
- Frequent speaker at various conferences
- Joris is senior consultant & trainer for SpringSource
- Talks about Spring for a living

Overall Presentation Goal

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

Agenda



- Supported platforms
- Configuration
 - **Annotations** & Namespaces, JCA, AspectJ
- The TestContext Framework
- Other Spring Portfolio news

Platforms

From Java 6 to Java EE 5 to OSGi

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 API's 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**

Java 6: JMX MBeans

- MBeans are new addition to JMX
- Better support for bundling related values
 - Standard MBeans require custom classes
- Can be registered by MBeanExporter
 - JMX Spec does not allow dynamic creation



Copyright 2007 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Java 6: JDK ServiceLoader API

- `java.util.ServiceLoader`
 - Registration of *service providers* for *services*
 - META-INF/services/my.service defines implementing classes for `my.service`
- Used by `Service (List) FactoryBean`

```
<bean id="sqlDriver"  
      class="org.sfw...ServiceFactoryBean">  
  <property name="serviceType"  
            value="java.sql.Driver"/>  
</bean>
```

Support for Built-in HTTP Server

- Spring 2.5 supports the Java 6 built-in HTTP Server
- HTTP-based Remoting using **SimpleHttpInvokerServiceExporter** & **SimpleHessian/BurlapServiceExporter**
- Set-up JRE 1.6 HttpServer using **SimpleHttpServerFactoryBean**

Support for new Platforms

New Platform support:

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

Java EE 5 support

- Support for Java EE 5
- Integrates seamlessly
- New Java EE 5 API's supported:
 - 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
 - e.g. BEA WebLogic 8.1 or higher
 - e.g. IBM WebSphere 5.1 or higher

Java EE 5: API's

- Support for unified expression language
- JSF 1.2: **SpringBeanFacesELResolver**
- Consistent use of JSR-250 annotations
- JTA 1.1: support new **TransactionSynchronizationRegistry**
- New JTA, JavaMail and JAX-WS support also available for stand-alone usage

Other J2EE enhancements: RAR file support

Deploy Spring app 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
- Can access app server services like JTA TransactionManager and MBeanServer

Other J2EE enhancements: IBM WebSphere 6.x

2.5 officially supported on IBM WAS 6.x

- `WebSphereUowTransactionManager`
- Or `<tx:jta-transaction-manager>`
- `WebSphereTransactionManager-FactoryBean` replacement
- No new features, but uses *supported* IBM API

Support for new Platforms

New Platform support:

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

Spring & OSGi

- Open Services Gateway initiative
- Dynamic module system
- *Bundle* as central packaging unit
 - Versioned jar
 - Exports types to expose
 - Started, stopped and updated *at runtime*
- 2.5 jars now are compliant bundles
 - Headers in MANIFEST.MF

Spring & OSGi

- Integration with OSGi provided by the *Spring Dynamic Modules for OSGi™ Service Platforms* project
- TPFKASO
- 1.1.2 released in October
 - **ApplicationContext** per bundle
- Integration with OSGi service registry

Spring on OSGi vs. Spring on Java EE

- Similar programming models
 - No dependencies on environment
 - Services are Spring managed components
- Very different deployment environment
 - Typically alternative runtimes
- Hard to mix
- Spring provides the common ground
- SpringSource dm Server aims to change this

Configuration

From JSR-250 to @Controller

New Configuration features

- Annotation-driven configuration
- JMS & JCA support
- Enhanced AspectJ support
- Annotation-driven MVC Controllers

Annotation-driven configuration

Spring 2.5 embraces annotations:

- JSR-250 Common Annotations
- Spring Autowiring Annotations
- Component scanning for autodetection of components
- Additional configuration option
 - XML is in no way deprecated!
 - Different from Spring JavaConfig



JSR-250 Common Annotations

- Part of Java EE 5 and JDK 1.6
 - Available as extra jar for JDK 1.5
- Annotations for lifecycle and DI
 - **@PostConstruct** & **@PreDestroy**
- cf. init- and destroy-methods
 - **@Resource**
- Injection of named beans or JNDI resources
- Can also be used on fields

JSR-250 Annotations Example



JSR-250 Annotations Example



```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
}
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource (name="myProcessor")
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
}
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}  
  
// No specific configuration is necessary!
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}  
  
// No specific configuration is necessary!  
<bean id="myService" class="mypackage.MyService" />
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}  
  
// No specific configuration is necessary!  
<bean id="myService" class="mypackage.MyService" />
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}  
  
// No specific configuration is necessary!  
<bean id="myService" class="mypackage.MyService" />  
  
<bean class="org.springframework.context.annotation.
```

JSR-250 Annotations Example

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    @PostConstruct  
    public void initialize() { ... }  
  
    @PreDestroy  
    public void shutdown() { ... }  
}  
  
// No specific configuration is necessary!  
<bean id="myService" class="mypackage.MyService" />  
  
<bean class="org.springframework.context.annotation.  
    CommonAnnotationBeanPostProcessor"/>
```

Further Java EE 5 Annotations

- Java EE 5 includes further, more specific annotations
 - **@WebServiceRef / @EJB**
- injecting a JAX-WS / EJB 3 service proxy
 - **@TransactionAttribute**
- EJB 3 transaction demarcation
 - **@PersistenceContext / @PersistenceUnit**
- JPA resource injection
- supported since Spring 2.0 already!
- All consistently supported by Spring 2.5

Annotation-driven configuration

- JSR-250 Common Annotations
- **Spring Autowiring Annotations**
- Component scanning



Autowiring Annotation

Autowiring Annotation

- New **@Autowired** annotation
 - Autowiring by type
 - Of fields, methods and constructors
 - **AutowiredAnnotationBeanPostProcessor**

Autowiring Annotation

- New **@Autowired** annotation
 - Autowiring by type
 - Of fields, methods and constructors
 - **AutowiredAnnotationBeanPostProcessor**
- Sweet spot for autowiring:
 - *By name* often too simplistic
 - *By type* often too extensive
 - **Specific** by type works a lot better!

Autowiring Annotation Example



Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}  
  
// No specific configuration necessary: driven by annotations..
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}  
  
// No specific configuration necessary: driven by annotations..  
<bean id="myService" class="mypackage.MyService" />
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}  
  
// No specific configuration necessary: driven by annotations..  
<bean id="myService" class="mypackage.MyService" />
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Autowired  
    public void injectServices(ServiceA a, ServiceB b) { ... }  
}  
  
public class MyService implements MyServiceInterface {  
    ...  
  
    @Autowired  
    public MyService(DataSource dataSource, ServiceA a) { ... }  
}  
  
// No specific configuration necessary: driven by annotations..  
<bean id="myService" class="mypackage.MyService" />  
  
<bean class="org.springframework.beans.factory.annotation.
```

Autowiring Annotation Example

```
public class MyService implements MyServiceInterface {

    @Autowired
    private DataSource dataSource;

    @Autowired
    public void injectServices(ServiceA a, ServiceB b) { ... }
}

public class MyService implements MyServiceInterface {
    ...

    @Autowired
    public MyService(DataSource dataSource, ServiceA a) { ... }
}

// No specific configuration necessary: driven by annotations...
<bean id="myService" class="mypackage.MyService" />

<bean class="org.springframework.beans.factory.annotation.
    AutowiredAnnotationBeanPostProcessor"/>
```

@Qualifier Annotation

- Autowiring by type may have too many candidates
- Provide *hints* using qualifiers!
- Through new `@Qualifier` annotation
- On fields / parameters or on custom annotations

@Qualifier Example: By Name



@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

```
<bean id="orderDataSource" class="example.DataSource">
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}

<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}

<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}

<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

```
<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

```
<bean id="irrelevant" class="example.DataSource">
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

```
<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

```
<bean id="irrelevant" class="example.DataSource">
    <qualifier value="inventoryDataSource"/>
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

```
<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

```
<bean id="irrelevant" class="example.DataSource">
    <qualifier value="inventoryDataSource"/>
    <!-- connection properties for Inventory DataSource -->
```

@Qualifier Example: By Name

```
public class JdbcOrderRepositoryImpl implements OrderRepository
{
    // autowire DataSources by type with qualifying names
    @Autowired
    public void init(
        @Qualifier("orderDataSource") DataSource orderDS,
        @Qualifier("inventoryDataSource") DataSource inventoryDS,
        MyHelper autowiredByType)
    {
        // ...
    }
}
```

```
<bean id="orderDataSource" class="example.DataSource">
    <!-- connection properties for Order DataSource -->
</bean>
```

```
<bean id="irrelevant" class="example.DataSource">
    <qualifier value="inventoryDataSource"/>
    <!-- connection properties for Inventory DataSource -->
</bean>
```

@Qualifier Example: Custom Annotation (1)



@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
        @Genre("Comedy") MovieCatalog comedyCatalog)
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
        @Genre("Comedy") MovieCatalog comedyCatalog)
    {
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
        @Genre("Comedy") MovieCatalog comedyCatalog)
    {
        this.comedyCatalog = comedyCatalog;
    }
}
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
        @Genre("Comedy") MovieCatalog comedyCatalog)
    {
        this.comedyCatalog = comedyCatalog;
    }
}
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
        @Genre("Comedy") MovieCatalog comedyCatalog)
    {
        this.comedyCatalog = comedyCatalog;
    }
    // ...
}
```

@Qualifier Example: Custom Annotation (1)

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

public class MovieRecommender {

    @Autowired @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(
        @Genre("Comedy") MovieCatalog comedyCatalog)
    {
        this.comedyCatalog = comedyCatalog;
    }
    // ...
}
```

@Qualifier Example: Custom Annotation (2)



```
<bean class="example.SimpleMovieCatalog">
    <qualifier type="Genre" value="Action"/>
    <!-- inject any dependencies required by this bean -->
</bean>
```

```
<bean class="example.SimpleMovieCatalog">
    <qualifier type="example.Genre" value="Comedy"/>
    <!-- inject any dependencies required by this bean -->
</bean>
```

```
<bean id="movieRecommender" class="example.MovieRecommender"/>
```

```
<context:annotation-config/>
```

@Qualifier Example: Custom Annotation (2)



```
<bean class="example.SimpleMovieCatalog">  
    <qualifier type="Genre" value="Action"/>  
    <!-- inject any dependencies required by this bean -->  
</bean>
```

```
<bean class="example.SimpleMovieCatalog">  
    <qualifier type="example.Genre" value="Comedy"/>  
    <!-- inject any dependencies required by this bean -->  
</bean>
```

```
<bean id="movieRecommender" class="example.MovieRecommender"/>
```

```
<context:annotation-config/>
```



New namespace!

The context XML Namespace

New configuration namespace: **context**

- **<context:property-placeholder location="...">**
 - PropertyPlaceholderConfigurer
- **<context:annotation-config>**
 - activating JSR-250, all common autowiring annotations, **@Required**
- **<context:mbean-export>**
 - activating annotation-driven MBean export (**@ManagedResource**, **@ManagedOperation**)

More Autowiring

- Autowiring of typed Collections:

```
Autowired  
private List<MovieCatalog> allCatalogs
```

- Replacing *Aware interfaces:

```
Autowired  
MessageSource messageSource  
  
Autowired  
ResourceLoader resourceLoader  
  
Autowired  
ApplicationContext applicationContext
```

Annotation-based Autowiring pros and cons

- Pros:
 - Self-contained: no XML config needed
 - Works in much more cases than generic autowiring (any method or field)
 - JSR-250 or custom annotations keep your code from depending on Spring
- Cons:
 - Requires classes to be annotated
 - Configuration only per class, not per instance
 - Changes require recompilation

Annotation-driven configuration

- JSR-250 Common Annotations
- Spring Autowiring Annotations
- **Component scanning**



Annotated Components

- Annotate a type to make it a Spring bean
 - with **@Component**
 - Or annotation which has @Component
- Spring has @Repository, @Service and @Controller stereotypes
- You can create your own
- No <bean> tag needed anymore!
- Enabled with <context:component:scan>
 - Scans the classpath for beans

Annotated Component Example

```
package mypackage.services;

@Service
public class MyService implements MyServiceInterface {

    @Resource (name="myDataSource")
    private DataSource dataSource;

    @Autowired
    public void injectServices (ServiceA a, ServiceB b) { ... }

    @PostConstruct
    public void initialize() { ... }

    @PreDestroy
    public void shutdown() { ... }
}

<!-- Not even a plain XML <bean> tag is necessary! -->
<context:component-scan base-package="mypackage.services" />
```

@Component



@Component

- Provide bean name using value element:
 - @Component("myService")
 - Defaults to uncapitalized short class name:
`mypackage.MyService => myService`

@Component

- Provide bean name using value element:
 - @Component("myService")
 - Defaults to uncapitalized short class name:
mypackage.MyService => myService
- Create your own stereotype annotations:

```
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Component  
public @interface Manager {  
    String value() default "";
```

Component scanning configuration

- Component scanning is very configurable:
 - Use includes and excludes
 - Scan for annotations, assignable types or matching class names
 - Or even define your own strategy

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

Scoping and Qualifiers

- Can also define scope:

```
@Scope("session")  
public class ShoppingCart {
```

- Can also define custom qualifier:

```
@Genre("Action")  
public class ActionCatalog implements MovieCatalog {
```

Component Scanning Pros

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

Component Scanning Cons

- Not a 100% solution
- Requires classes to be annotated
- Don't scan a huge number of classes!
 - use Spring's filtering mechanism

New Configuration features

- Annotation-driven configuration
- **JMS & JCA Support**
- Enhanced AspectJ support
- Annotation-driven MVC Controllers



JMS Namespace

- New **jms** namespace
 - Define ListenerContainers with listeners per destination

```
<!-- standard JMS message listener container -->
<jms:listener-container
  connection-factory="myConnectionFactory"
  transaction-manager="myTransactionManager">

  <jms:listener destination="myQueue" ref="myListener"/>
  <jms:listener destination="myQueue2" ref="myListener2"/>

</jms:listener-container>
```

JMS Namespace Adapter Support

- Easily define MessageListenerAdapter using method attribute
 - Use POJO to process message contents

```
<jms:listener-container connection-factory="connFactory">  
  <jms:listener destination="myQueue" ref="myService"  
    method="handleRequest"/>>  
</jms:listener-container>
```

```
@Service("myService")  
public class MyServiceImpl implements MyService {  
    public String handleRequest(String request) {...}  
}
```

JCA Support

Spring 2.5 introduces full JCA 1.5 support

- generic JCA ResourceAdapter support
- generic JCA message endpoint support
 - JMS message listeners
 - CCI message listeners
 - Comparable to Message Driven Beans
- configurable backends
 - e.g. Spring 2.0's TaskExecutor abstraction

JCA Setup Example for ActiveMQ

```
<bean class=
  "org.sfw.jca.endpoint.GenericMessageEndpointManager">

  <property name="resourceAdapter" ref="myResourceAdapter"/>

  <property name="messageEndpointFactory">
    <bean class=
      "org.sfw.jca.endpoint.GenericMessageEndpointFactory">
      <property name="messageListener" ref="myListener"/>
      <property name="transactionManager"
        ref="myTransactionManager"/>
    </bean>
  </property>

  <property name="activationSpec">
    <bean class="org.apache.activemq.ra.ActiveMQActivationSpec">
      <property name="destination" value="myQueue"/>
      <property name="destinationType" value="javax.jms.Queue"/>
    </bean>
  </property>

</bean>
```

JMS Namespace support for JCA

- JCA can also be configured through jms namespace

```
<!-- JCA-based JMS message listener container -->  
<jms:jca-listener-container  
    resource-adapter="myResourceAdapter"  
    transaction-manager="myTransactionManager">  
  
    <jms:listener destination="myQueue" ref="myListener"/>  
    <jms:listener destination="myQueue2" ref="myListener2"/>  
  
</jms:jca-listener-container>
```

New Configuration features

- Annotation-driven configuration
- JMS & JCA Support
- **Enhanced AspectJ support**
- Annotation-driven MVC Controllers

aspectj *crosscutting objects for better modularity*

AspectJ bean pointcut

- New `bean(name)` pointcut element
 - For use in AspectJ pointcuts
 - Matches beans by name
 - Supports wildcards

```
<aop:advisor pointcut="bean(*Service)  
              advice-ref="accessCounter"
```

- No more need for `BeanNameAutoProxyCreator`!

AspectJ Load-Time Weaving

- Support AspectJ load-time weaving through Spring's **LoadTimeWeaver**
- Driven by `META-INF/aop.xml` files
- For any supported platform
 - Generic Spring VM agent
 - Various app servers: Tomcat, Glassfish, OC4J

```
<context:load-time-weaver/>
```

@Configurable revisited

- Spring 2.0 introduced **@Configurable**
 - For non-Spring-managed objects
 - Dependency-inject *any* object using AspectJ
- Spring 2.5: now supported with load-time weaving
 - No AspectJ compiler needed
 - Good combination with annotation-driven config

```
<context:spring-configured/>
```

@Configurable applied

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->
<!-- spring-aspects.jar is required on the classpath -->
<!-- ClassLoader needs to be weaving-capable -->

<context:load-time-weaver aspectj-weaving="on"/>
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->
<!-- spring-aspects.jar is required on the classpath -->
<!-- ClassLoader needs to be weaving-capable -->

<context:load-time-weaver aspectj-weaving="on"/>

<context:spring-configured />
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

```
    @Autowired
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

```
    @Autowired
```

```
    private MyService myService;
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

```
    @Autowired
```

```
    private MyService myService;
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->  
<!-- spring-aspects.jar is required on the classpath -->  
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

```
    @Autowired
```

```
    private MyService myService;
```

```
}
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->
<!-- spring-aspects.jar is required on the classpath -->
<!-- ClassLoader needs to be weaving-capable -->
```

```
<context:load-time-weaver aspectj-weaving="on"/>
```

```
<context:spring-configured />
```

```
@Configurable
```

```
public class MyDomainObject {
```

```
    @Autowired
```

```
    private MyService myService;
```

```
}
```

```
// The following domain object will be configured by
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->
<!-- spring-aspects.jar is required on the classpath -->
<!-- ClassLoader needs to be weaving-capable -->

<context:load-time-weaver aspectj-weaving="on"/>

<context:spring-configured />

@Configurable
public class MyDomainObject {

    @Autowired
    private MyService myService;

}

// The following domain object will be configured by
// Spring!
```

@Configurable applied

```
<!-- configures AspectJ bean configurer aspect -->
<!-- spring-aspects.jar is required on the classpath -->
<!-- ClassLoader needs to be weaving-capable -->

<context:load-time-weaver aspectj-weaving="on"/>

<context:spring-configured />

@Configurable
public class MyDomainObject {

    @Autowired
    private MyService myService;

}

// The following domain object will be configured by
// Spring!
MyDomainObject obj = new MyDomainObject();
```

Used with transactions: Spring-AOP



- Traditional <tx:annotation-driven> with @Transactional:

```
<!-- automatically detects annotated components -->
<tx:annotation-driven transaction-manager="txManager" />

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

<!-- Proxied: only allowed to use @Transactional on public,
      externally called methods! -->
```

Used with transactions: AspectJ Load-time Weaving



- `<tx:annotation-driven>` and AspectJ weaving:

```
<!-- configures AspectJ bean configurer aspect -->
<!-- spring-aspects.jar is required on the classpath -->
<!-- ClassLoader needs to be weaving-capable -->
<context:load-time-weaver aspectj-weaving="on"/>

<tx:annotation-driven mode="aspectj"

<tx:jta-transaction-manager />

<!-- Woven: can use @Transactional on any method! -->
```

New Configuration features

- Annotation-driven configuration
- JMS & JCA Support
- Enhanced AspectJ support
- **Annotation-driven MVC Controllers**

Annotation-driven Controllers

- Java5 variant of MultiActionController
 - Including form handling capabilities
- POJO-based
 - Just annotate your class
 - Works in servlet and portlet container
- Several annotations:
 - @Controller
 - @RequestMapping / @RequestMethod
 - @RequestParam
 - @ModelAttribute
 - @SessionAttributes
 - @InitBinder

Example of Annotated 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";
    }
}
```

Annotated Controller from PetClinic (1)

- Session-based form setup:

```
@Controller
@RequestMapping("/editPet.do")
@SessionAttributes("pet")
public class EditPetForm {
    // ...
    @RequestMapping(method = RequestMethod.GET)
    public String setupForm(
        @RequestParam("petId") int petId, ModelMap model)
    {
        Pet pet = this.clinic.loadPet(petId);
        model.addAttribute("pet", pet);
        return "petForm";
    }

    // ...
}
```

Annotated Controller from PetClinic (2)

- Session-based form processing:

```
// EditPetForm continued ...
```

```
@ModelAttribute("types")
```

```
public Collection<PetType> populatePetTypes() {  
    return this.clinic.getPetTypes();  
}
```

```
@RequestMapping(method = RequestMethod.POST)
```

```
public String processSubmit(  
    @ModelAttribute("pet") Pet pet, BindingResult result,  
    SessionStatus status)  
{  
    new PetValidator().validate(pet, result);  
    if (result.hasErrors()) { return "petForm"; }  
    else {  
        this.clinic.storePet(pet);  
        status.setComplete();  
        return "redirect:owner.do?ownerId=" +  
            pet.getOwner().getId();  
    }  
}
```

Supported parameter types

- Parameter types that can be used for controller and @InitBinder methods:
 - request / response / session / WebRequest
 - DataBinder (@InitBinder only)
 - Locale
 - InputStream / Reader and OutputStream / Writer
 - @RequestParam annotated
- Including for example MultipartFile
 - Map / ModelMap
 - Command / Form Objects plus Errors / BindingResult
- Controller methods only
 - SessionStatus

More Annotated Controller functionality

- There's even more
 - e.g. the new `WebBindingInitializer`, using `RequestToViewNameTranslator`
- Will be expanded further in Spring 3.0 with REST support
 - URI Templates
 - HTTP Header value extraction
 - View selection based on Accept header or file extension in URL
 - New views: JSON, OXM, Atom, RSS

The Test Context Framework

**From AbstractDependencyInjection-
SpringContextTests to @ContextConfiguration**

Spring Test Context Framework

- Revised, annotation-based test framework
- Supports JUnit 4.4, TestNG as well as JUnit3.8
- Supersedes older JUnit 3.8 base classes
 - AbstractDependencyInjection-SpringContextTests & friends
 - They're still there for JDK 1.4
 - Will be deprecated in Spring 3.0

Spring Test Context Framework

- Convention over configuration
 - Use only annotations
 - Reasonable defaults that can be overridden
- Consistent support for Spring's core annotations
- Spring-specific integration testing functions:
 - Context management & caching
 - Dependency injection of tests
 - Transactional test management

Annotated Test Class Example

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
// defaults to MyTests-context.xml in same package
public class MyTests {

    @Autowired
    private MyService myService;

    @Test
    public void myTest() {...}

    @Test
    @Transactional
    public void myOtherTest() { ... }
}
```

Using the TestContext Framework

- Use the `SpringJUnit4ClassRunner` for JUnit 4.4 or instrument test class with `TestContextManager` for TestNG
- Or extend one of the new base classes
 - `Abstract(Transactional)`
`[JUnit38 | Junit4 | TestNG]`
`SpringContextTests`

Test Context Annotations

- **TestExecutionListeners**
 - `@TestExecutionListeners`
- **Application Contexts**
 - `@ContextConfiguration` **and** `@DirtiesContext`
- **Dependency Injection**
 - `@Autowired`, `@Qualifier`, `@Resource`, `@Required`, **etc.**
- **Transactions**
 - `@Transactional`, `@NotTransactional`,
`@TransactionConfiguration`, `@Rollback`,
`@BeforeTransaction`, **and** `@AfterTransaction`
- **Testing Profiles (JUnit only)**
 - `@IfProfileValue` **and** `@ProfileValueSourceConfiguration`
- **JUnit extensions**
 - `@ExpectedException`, `@Timed`, `@Repeat`

Copyright 2007 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

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

Summary



- Spring 2.5 builds on strong 2.0 foundation
 - With updated support for standards
 - And focus on ease of configuration
- Embraces Java 6, Java EE 5 as well as OSGi
 - Spring as 'traditional' Java EE framework
 - Or as application framework in OSGi environment
 - Same programming model!
- Embraces annotations for configuration
 - As an addition to existing alternatives
- Enhances XML configuration

For More Information

- Updated sample applications
- Spring Reference Manual
- blog.springframework.org
- www.springframework.org
- Spring Forums

DEMO

The revised PetClinic

Other Spring Portfolio News

There's more than just Spring!

The Spring Portfolio

■ Other Spring Portfolio Products:

- Spring Web Services
- Spring Web Flow
- Spring Security
- Spring Dynamic Modules for OSGi
- Spring Batch
- Spring IDE
- Spring JavaConfig
- Spring Integration

● So, what's new there?

- Don't worry, just the highlights 😊

Spring Web Services 1.5

- Current version 1.5.5
 - Contract-first web services framework
 - SOAP and POX
 - XML Marshallers
 - WSDL generation
 - WS-Security, integrated with Spring Security
 - WS-Addressing
 - HTTP(S), SMTP and JMS Transports

Spring Web Services 1.6

- New version 1.6 in development
 - Enhanced Annotation support
 - Extra transports (XMPP)
 - More control through additional callbacks

Spring Web Flow 2.0

- Current version 2.0.5
 - Model stateful user interactions in web apps
 - Reusable and composable modules: **flows**
- 2.0 is major overhaul:
 - Much improved syntax
 - New support for EL, Ajax, JSF (Spring Faces) and Dojo (Spring-JavaScript)
 - Includes Session/EntityManager in conversation support

Spring Security 2.0

- Current version 2.0.4
 - Flexible, Declarative Security Framework
 - Independent of J2EE security: very portable!
 - Integrates with existing authentication solutions
- JAAS, LDAP, SSO, ...
 - Easy to build or plug in your own
 - Lots of features
- Remember-me, Run-as replacement, ACL, ...

Spring Security 2.0

- Spring Security 2.0 MUCH easier to use
 - New namespace support, cutting down XML that's needed
 - Windows NTLM authentication
 - User management API
 - Hierarchical Roles
 - ACL enhancements
 - Portlet support
 - And more...

Spring Dynamic Modules for OSGi

- Currently 1.1.2
 - Basis for SpringSource dm Server
- Version 2.0 in development
- Will be reference implementation for RFC 124, "A Component Model for OSGi"

Other Portfolio Releases (1)

- Spring Batch
 - 1.1.3 is current release
 - Support for defining and running batch jobs
- Repeat, retry, chunking, etc.
 - 2.0 development is in progress
- M3 has been released
- New namespace and annotation support
- Chunking now 1st class citizen
- Steps don't have to be sequential

Other Portfolio Releases (2)

- Spring JavaConfig
 - 1.0M3 released
 - Will be partly incorporated into Spring 3.0
 - 1.0 released with or immediately following Spring 3.0 GA
- Spring IDE
 - At version 2.2.1
 - Many improvements and new features, incl. support for latest portfolio project versions
 - Remains Open Source, despite Spring Tool Suite

Other Portfolio Releases (3)

- Spring Integration
 - New addition, 1.0 just released
- Implements Enterprise Integration Patterns for use *within* your application
 - As opposed to through external ESB
 - Adapters to connect to external systems
- Allows event-driven architectures
 - Based on message abstraction

Q&A